An Introduction to Video Compression in C/C++

Fore June

Chapter 1

Image Prediction and Motion Compensation

9.1 Introduction

In previous chapters, we combined various well-developed techniques, which are shown in Figure 7-1 to compress image data. You may be amazed to see that the techniques can achieve high compression ratios with high-quality reproduced decompressed images. Actually, the best is yet to come. So far, we have only considered static images and have used .ppm files in our examples; the techniques employed in Figure 7-1 have only exploited the correlations between pixels (or spatial redundancy). These techniques are very similar to those used by the JPEG standard to compress static images. Actually, they can be applied to encode a sequence of images, compressing each image individually. This method of compressing each frame independently is known as motion JPEG or M-JPEG. Obviously, M-JPEG has not made use of the correlations between frames (temporal redundancy) to achieve higher compression. M-JPEG is usually used in very high quality video captures and the scenes are normally captured in the raw data format which are then edited and compressed into another format. Encoding each frame individually is also referred to as *intra-frame* coding, where at a certain instance, data processing is applied only to the data of the current frame but not to any other frame in the video sequence. This contrasts with *inter-frame* coding, where processing is applied simultaneously to the data of the current frame and the adjacent frames. If you watch a movie played by a DVD player and at some point change it to play at a slow-motion mode, you would notice that most consecutive frames within a sequence are very similar to the frames both before and after the frame of interest. By exploiting the inherent temporal, or time-based redundancies between frames, considerably more compression efficiency can be obtained.

Intra-frames are often referred to as I-frames and inter-frames are called P-frames. P here refers to ‘prediction’. This is because people use motion estimation and a technique known as block-based motion compensated prediction to exploit the temporal correlations between frames in order to reduce the redundancies. Indeed, in Chapter 2 we have discussed that the information conveyed by a set of data closely relates to its predictability. A perfectly predictable message conveys no information. Actually, besides temporal prediction, we can also use spatial prediction to reduce data redundancies.

9.2 Temporal Model

A temporal model exploits inter-frame correlations to reduce redundancies. Very often, a predicted frame is calculated or formed by certain procedures and is subtracted from the current frame, producing a residual or difference frame. The more accurate the prediction, the less information the residual data contain and the higher compression we can achieve. The residual data are encoded in the usual way as presented in Figure 7-1. Besides decoding the encoded residual data, the decoder has to recreate the predicted frame and adds it to the decoded residual to reconstruct the current frame (usually this is a lossy process; the reconstructed frame is not identical to the original one). The predicted frame is often created from one or more past or future frames which are referred to as 'reference frames'. Predictions in general can be improved by compensating for motion between the current frame and reference frames. We will discuss motion compensation in detail in the next section.

For instance, MPEG predicts images from previous frames (P frames) or bidirectionally from previous and future frames (B frames). After predicting frames using motion compensation, the coder calculates the residual which is then compressed using the methods shown in Figure 7-1.

9.3 Block Based Motion Estimation and Motion Compensation

A simple and commonly used method for temporal prediction is to use the previous frame as the predictor for the current frame. However, very often for many frames of a video scene, the main difference between one frame and another is the result of either the camera moving or an object in the frame moving. This means much of the information that represents one frame will be the same as the information used in the next frame. A direct subtraction of the previous frame (predictor) from the current frame could produce substantial nonzero residual values. For instance, consider an example where the images are shown in Figure 9-1 and Figure 9-2 (for simplicity, we assume that a white pixel has a zero value and a black pixel has a nonzero value). The objects in the two figures are identical except that one is displaced with respect to the other. Direct calculation of the difference between the two frames yields substantial residual values as shown in Figure 9-3. Actually, this example is quite extreme as the difference has more nonzero values than Frame 1 and Frame 2, resulting in a less efficient compression. However, if we make compensation for the motion by translating the object in Frame 1 to the position of the object in Frame 2 before carrying out the subtraction, then the residual will consist of all zeros, which result in a very efficient compression.

The residual can be compressed and decompressed as usual (Figure 7-1). *But how do we recover the current frame (Frame 2)?* To reconstruct the current frame, the decoder needs to know the the image data of the previous frame (Frame 1) and the displacement of the object, which is known as **motion vector**. The process of obtaining the motion vector is known as **motion estimation**, and using the motion vector to remedy the effects of motion is known as **motion compensation**.

Once we have reconstructed Frame 2, we can use it as the predictor for Frame 3 and the reconstructed Frame 3 is used as the predictor for Frame 4 and so on. Therefore, we only need to send one complete frame to be used as the predictor along with the residual and motion vectors at the very beginning. After that, the predictor (reference frame) is reconstructed from other information.

Of course in reality, the objects in two different frames are rarely identical. Moreover, many objects in a scene are deformable and it is very difficult to identify all the objects in a large number of frames. Then in practice, how do we do motion compensation?

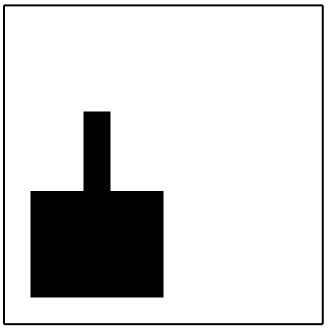
Figure 9-1. Frame 1

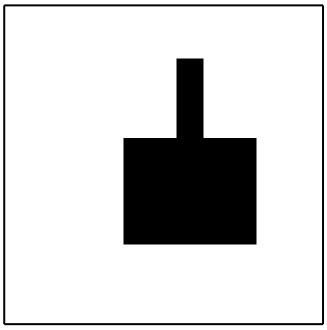
Figure 9-2. Frame 2

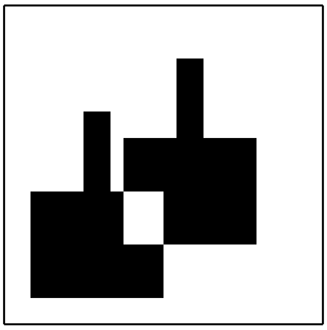
Figure 9-3. Residual = Frame 1 - Frame 2

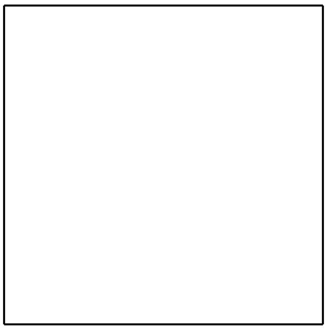
Figure 9-4. Motion Compensated Residual

The most commonly used technique in motion compensation is the block-based motion estimation. In this method, a current frame is divided into rectangular sections or ‘blocks’. We handle each block independently and search in the reference frame a block that matches this block best. The following procedures explain more precisely how the search is carried out for each block of size $M \times N$:

1. Search a region in the reference frame that best matches the $M \times N$ sample block. This is done by comparing the $M \times N$ block of the current frame with some or all of the possible $M \times N$ regions of the reference frame. The best matched region can be determined using an objective quality measure such as the minimum absolute difference (MAD) or sum of absolute difference (SAD) discussed below. That is, we want to find an $M \times N$ block in

the reference frame so that the sum of absolute difference (SAD) between it and the given sample block is minimized. This is the motion estimation procedure.

2. Set the selected block in the reference frame to become the predictor for the $M \times N$ sample block of the current frame and subtract it from the current block to form a residual $M \times N$ block. Also, calculate the motion vector (displacement) between the two blocks. This is the motion compensation procedure.

3. Encode the residual block using methods discussed in previous chapters (Figure 7-1). We may also encode the motion vector using a pre-calculated Huffman code. Send the encoded values to the decoder.

The decoder first decodes the encoded motion vector and residual block. It uses the motion vector to identify the predictor region in the reference frame and adds the predictor to the residual to reconstruct a version of the original block.

From the above discussions, we see that encoding one frame usually involves more than one motion vector. This is the reason that we encode a motion vector with a Huffman code. Figure 9-5a presents the video compression that extends Figure 7-1 to include temporal prediction, motion estimation and motion compensation; the steps of RGB-YCbCr transformation and down sampling are omitted in the diagram. The corresponding diagram for decoding the encoded stream is shown in Figure 9-5b.

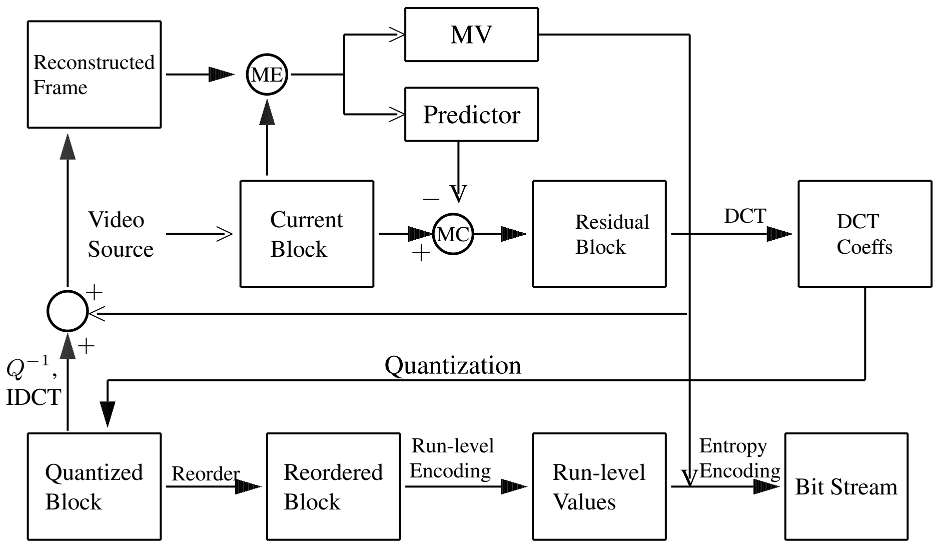

Figure 9-5a. Video Compression with ME and MC

In Figure 9-5, the terms MV, ME, MC, and Q^{-1} represent motion vector, motion estimation, motion compensation, and inverse quantization respectively. Block-based motion compensation (MC) is commonly used in video compression. This is because it is simple, and straightforward to implement. The algorithm could be efficient, depending on the choice of block size and search details. The method also fits well with commonly used rectangular video frames and the block-based Discrete Cosine Transform (DCT). On the other hand, there are some drawbacks in using block-based MC. Objects in a scene are often non-rectangular and their boundaries rarely match well with rectangular edges. Very often, an object may move by a fraction of the distance between pixels from one frame to another; the motion may not be simple translations but more complex motions like shrinking, rotation, and vibration. Also, objects like animals and liquid could be deformable and some objects such as clouds, smoke and fire may not even have any regular shape.

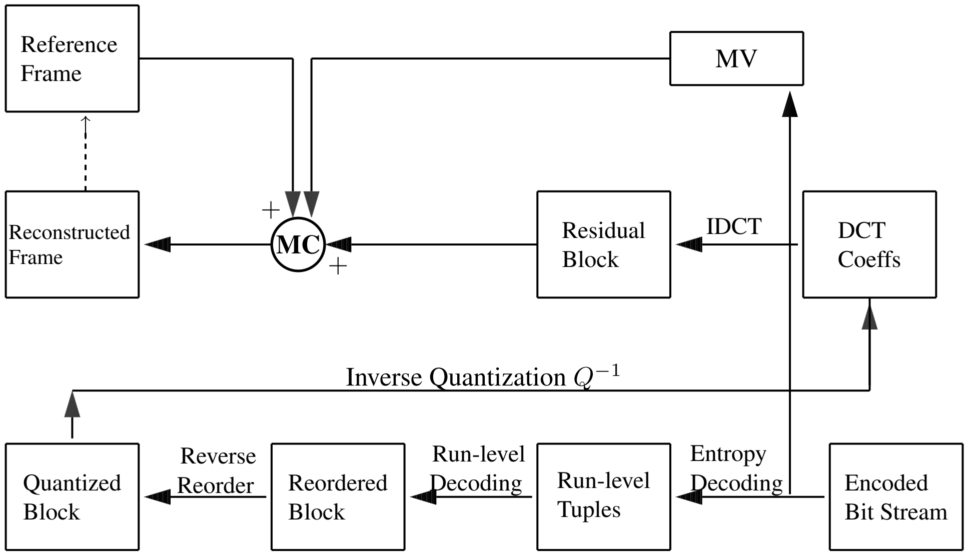

Figure 9-5b. Decoding of Compressed Stream with ME and MC

9.4 Matching Criteria

Before we perform any search, we need some criteria to determine which is the best match. Some of these criteria are simple to evaluate, while others are more involved. Different kinds of algorithms may use different criteria for comparison of blocks. The commonly used criteria are the "Sum of Differences" (SAD), the "Mean Absolute Difference" (MAD), and the "Mean Squared Difference" (MSD). If the compensation block size is $N \times N$ samples, and I_{ij} and I'_{ij} are the current and reference sample values respectively at location (i, j), the formulas for these criteria are presented in equations (9.1a), (9.1b), and (9.1c) below:

$$SAD = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} |I_{ij} - I'_{ij}| \tag{9.1a}$$

$$MAD = \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} |I_{ij} - I'_{ij}| \tag{9.1b}$$

$$MSD = \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} (I_{ij} - I'_{ij})^2 \tag{9.1c}$$

The functions SAD, MAD, and MSD presented in (9.1) are also referred to as distortion functions. Because of its simplicity, SAD is probably the most commonly used measure to determine the best match. In subsequent discussions, we shall also use “minimizing SAD” as our criterion to find the best match.

In practice, instead of searching in the RBG space, we search in the YCbCr space. The residuals are actually the differences of the YCbCr components between two frames. Also, we need to send the motion vectors to the decoder. Very often, small vectors occur more frequently, and a pre-calculated Huffman coder ‘bias’ towards smaller values. Therefore, we usually encode smaller vectors with less number of bits than larger vectors. Consequently, it may be useful to ‘bias’ the choice of a vector towards the location of the current block which is assumed to be at (0, 0). We can accomplish this by subtracting a constant from the SAD at location (0, 0).

9.5 Choice of Block Size

The next question arises on motion compensation is what block size should we use in the search. Intuitively, a smaller block size would give smaller SADs, yielding ‘better’ residual results (more zeros and small values). However, a smaller block size requires more search operations and motion vectors encoding. Encoded motion vectors increase the overhead bits required for decoding and the increase in bits may outweigh the benefits of improved residual compression. For instance, consider the extreme case that we were to use a block consisting of only a single pixel (i.e. block size = 1). Then in the searching process, a pixel with the same intensity value would result in a perfect match and produce a motion vector. However, we would not gain any compression in this situation because instead of encoding a set of pixel values, we would encode the same number of two dimensional motion vectors, which could be less correlated and make the compression worse.

On the other hand, a large block size requires much less search and motion vectors. However, a large block may contain small moving objects, giving rise to a poor predictor block and thus poor residual results. The larger the block, the less likely we will find a predictor block that matches reasonably well with the current block.

One effective way to compromise between choosing a large block size and a small block size is to adapt the block size to the image characteristics. For example, we choose a large block size in flat, and homogeneous areas of a frame and choose a small block size in regions of rapid or complex motion. In practice, video compression standards, including MPEG-1, MPEG-2, MPEG-4, H.261, H.263 and H.264 use a macroblock as the basic block for motion compensation. H.264 also uses adaptive motion compensation block sizes which are organized in a tree structure. In Chapter 5, we discussed that a macroblock is a 16×16 region of a frame. It shows in Figure 5-1 that a 4:2:0 formatted macroblock consists of a 16×16 luminance (Y) sample block, an 8×8 blue chrominance (Cb) sample block and an 8×8 red chrominance (Cr) sample block. Motion estimation is done by searching a 16×16 sample region in a reference frame that best matches the current macroblock consisting of Y, Cb, and Cr samples. The Minimum Absolute Difference (MAD) criterion or the Sum of Absolute Differences (SAD) is usually used to determine the best-match. Motion compensation is done by subtracting the chosen best matching region in the reference frame from the current macroblock to generate a residual macroblock, which is encoded in the usual manner (Figure 7-1 of Chapter 7) and sent to the decoder along with an encoded motion vector that describes the position of the selected region relative to the current macroblock. Within the encoder, we have to decode encoded residual macroblock and make use of the motion vector to add it to the matching region to form a reconstructed macroblock frame which is stored as a reference for future motion compensation operations (see Figure 9-5). You may wonder why we obtain the reference frame by reconstructing the macroblock rather than using one from the video source. This is because the encoder and decoder need to use an identical reference frame for motion compensation otherwise errors will accumulate over time. Sometimes, if there is a rapid scene change, causing a significant difference between adjacent frames, it is better not to use motion compensation in the encoding. To carry out the compression effectively, one can allow the encoding to switch between *intra* mode that encodes without motion compensation and *inter* mode that uses motion compensation for each macroblock. Also, objects may move by a fraction of the distance between two pixels rather than an integral value. In this case, we may be able to find better predictions by first interpolating the reference frame to sub-pixel positions before searching a best-match of these positions.

9.6 Motion Estimation Algorithms

Often motion estimation (ME) is the most computationally intensive part of a video encoder. Reduction of the required computational complexity is one of the most challenging issues for motion compensation. It is the state of the art in video encoding and is particularly critical in real-time video compression. On the other hand, nonreal-time applications can do the compression offline and can afford to spend more time on optimizing the compression. The decoder does not need to do any motion compensation and in general runs a lot faster than the encoder. Moreover, in many applications such as the distribution of a video clip, compression has to be done only once but decompression will be done many times. Therefore, these applications may afford to pay the cost of computation even if they are very high during the compression process.

As mentioned in the previous section, a common way to do motion estimation is to find a region in the reference frame that best matches the current macroblock and compute the motion vector between the two regions. *How do we find the best-matched region?* A model that is commonly used in searching is the *block-translation* model, where an image is divided into non-overlapping rectangular blocks. Each block in the predicted image is formed by translating a similar source region from the reference frame. This model does not consider any rotation or scaling of the block. The brute-force exhaustive search, which is also known as full search is one of such models.

9.6.1 Full Search

In a full search (or exhaustive search), we test every block within a defined range against the block it is defined to match (target block). In terms of minimizing SAD, exhaustive search always finds the optimal motion vector because the method compares all possible displacements within the search range. The following C-like pseudo code shows this algorithm:

```
for ( int i = 0; i < 256; ++i ) {
  x[i] = pixel in current macroblock
  y[i] = pixel in a 16x16 block in reference frame
  for each k = 16x16 block in reference frame {
    D[k] = sum of distortion( x[i], y[i][k] )
  }
}

return k-th 16x16 block in reference frame that minimizes D
```

If the search window size is 2S + 1, and its center (0, 0) is at the position of the current macroblock, the search window is typically a rectangular region with lower-left and upper-right corners defined by the coordinates $(-S, -S\,)$, and $(+S, +S)$. Full search evaluates SAD at each point of the window, for a total of $(2S+1)^2$ points. A simple full search strategy is to start from the upper-left corner at $(-S, +S)$ and proceed in raster scan order, from left to right and top to bottom until SADs at all positions have been evaluated. However, in a typical video sequence, movements of objects between frames are small and small SAD positions are concentrated around (0, 0). It is not likely that we find a minimum SAD near the corners of the search window. Therefore, we can simplify a full search by starting the search at the center (0, 0) and proceeding to evaluate points in a spiral pattern as shown in Figure 9-6. Of course, if we evaluate the SAD at every position as usual, we do not gain an advantage. However, we can make a short cut by adopting an early termination strategy, in which the evaluation of a SAD terminates once its value is larger than the previous SAD minimum (note that we just terminate the SAD evaluation at the position but we do **not** terminate the search until all positions in the window have been covered). By using this early termination strategy, it is increasing likely that the search will terminate early as the search pattern expands outward, and thus saving a significant amount of time in calculating SADs.

There are various strategies to decide on the search range. The search range usually depends on how fast objects move in the image sequence, and how well we want to track the fastest objects. If our reference frames are adjacent frames, it is not possible to track an object that moves more than an image width or height between successive frames. It might be reasonable to track an object that moves about half an image width between successive frames but in terms of TV, this corresponds to an object that appears in a frame and might disappear in the subsequent frame in one tenth of a second, which is faster than what we need. If we aimed at tracking an object that traverses a frame in about half a second, and the video dimension is about 512×512 and is played at a rate of 40 frames per second (fps). we would need to accommodate a movement of about 12 pixels per frame. If we wish to make predictions from two successive frames instead of one, we would need to double the search range to about 24 pixels.

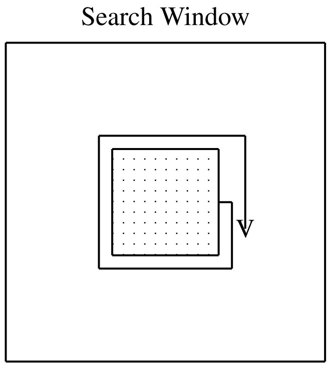

Figure 9-6. Full Search with Spiral Scan

Also, in real-world scene, there are usually more and faster movements in the horizontal direction than in the vertical direction. People found that it was best to search the width about twice as much as the height.

Full search is rather easy to implement. However, the best match as determined by a criterion like minimum SAD may not represent the real match of the object that appears in two frames. Good matches tend to minimize the residual errors, resulting in good compression, but if the matches do not represent true motion, the motion vectors of subsequent matching blocks may not correlate well, and encoding these vectors become inefficient.

9.6.2 Fast Searches

Though full search gives optimal results for a given criterion, it is computationally intensive. There are suboptimal search algorithms, usually referred to as fast search algorithms that trade the quality of the image prediction with the efficiency of searching. These algorithms evaluate the search criterion at a subset of the locations of the search window.

Three Step Search (TSS)

The Three Step Search (TSS),introduced by Koga et al in 1981 is a popular and robust search algorithm that gives near-optimal results. The N-Step Search (NSS) is a modified version of the TSS that calculates the SADs at a specified subset of locations within the search window. It searches for the best motion vectors in a coarse to fine search pattern. Figure 9-7 illustrates the search positions of this algorithm. Suppose the origin (0, 0) is at the center of the figure. We start by choosing the origin as our center of searching and pick a step size b, which is 4 in Figure 9-7. In the first stage, in addition to the origin, we choose eight locations of blocks at a “distance” of b from the center for comparison; in Figure 9-7 these first 9 search locations are labeled ‘1’. We pick the location that gives the smallest SAD as our new center of search, which is marked by a concentric circle in Figure 9-7. In the second stage search, the step size is halved and a further 8 locations around the new center are chosen with the new step size which is 2 in this example. The search locations for this stage is labeled ‘2’ in Figure 9-7. Once again, we pick the best-matched location to be the new search center and repeat the procedure until the step size cannot be subdivided further.

In an N-step search, the step size in general is $2^{(N-1)}$ and there are N searching levels. The number of searches is only $8N + 1$ as compared to $(2^{N+1} - 1)^2$ in a corresponding full search. However, NSS uses a uniformly allocated checking point pattern in the first step, which could be inefficient for small motion estimation.

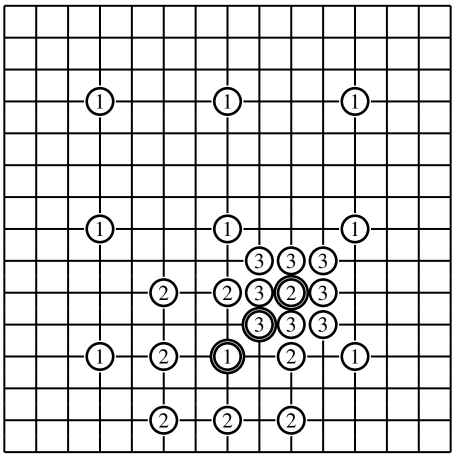

Figure 9-7. Three Step Search (TSS)

Hierarchical Search

Hierarchical search employs the coarse-to-fine approach to reduce computations at the coarse levels. At a coarse level, a 'large' block may contain many pixel positions but only a small number of searching locations; the sample value at a location may be the average (filtered value) of many adjacent pixel values. The algorithm first searches a large block to obtain a first approximation of the motion, which can be successively refined by searching smaller regions using smaller blocks, each block being appropriately filtered. The method has a better chance of getting the "real" motion vector because we first establish the general trend of motion using large filtered blocks and accomplish more accurate measurement using small blocks. It has been shown that motion vectors obtained by hierarchical search has significantly lower entropy, implying that they require less number of bits to encode.

For example, the mean pyramid method constructs different pyramidal images by subsampling, and estimates motion vectors starting from higher levels (coarse levels) and proceeding to lower ones. We can reduce the noise at higher levels by constructing the image pyramids using low pass filters, and use a simple averaging to construct the multiple-level pyramidal images. For instance, suppose $g_L(x, y)$ is the gray level value at the position (x, y) of the L-th level and $g_0(x, y)$ represents the original image at level 0. We obtain the pixel value at a level by averaging the four pixel values in a nonoverlapping window of the next lower level as shown in Equation (9.2):

$$g_L(x,y) = \lfloor \frac{1}{4} \sum_{i=0}^{1} \sum_{j=0}^{1} g_{L-1}(2x+i, 2y+j) \rfloor \tag{9.2}$$

where $\lfloor x \rfloor$ denotes the floor function of x, which truncates x to the nearest integer. If there are totally three levels in the pyramid, one pixel at level 2 corresponds to a 2×2 block and a 4×4 block at level 1 and level 0 respectively. Therefore, a block of size 16×16 at level 0 produces a block of size $16/2^L \times 16/2^L$ at level $L > 0$. After constructing the mean pyramid, we can search the images starting at level 2 using the minimum SAD (Sum of Absolute Differences) criterion; we select the motion vector with the smallest SAD as the coarse motion vector at that level. We send the motion vector detected at the higher level to the next lower level (level 1), which uses the received motion vector to guide the refinement step at that level. We repeat the motion estimation process once more down to level 0. Since SAD's are computed at the highest level based on relatively small blocks, the same values are likely to appear at several points. To solve this problem, we can use more than one candidate at the highest level (level 2 for our special case). A number of motion vectors at level 2 are propagated to the lower one. Full search in a small window around the candidates is used at level one to find the minimum difference location as the search center at layer 0. Figure 9-8 shows the search locations of the algorithm. At level 2 of the figure, three best matched points are selected as centers for search windows in the next level, which are shown at level 1 of the figure. Level 0 shows the search window where we search for the best match.

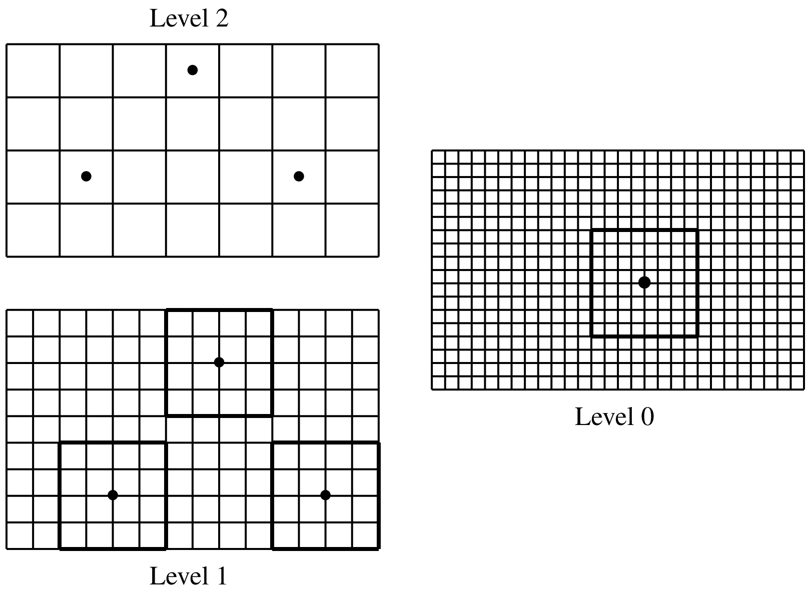

Figure 9-8. Search Locations for Hierarchical Search

Nearest Neighbours Search

One problem of the TSS, Hierarchical Search and many other suboptimal searches is that the searches may quickly get trapped in local minima, as the distortion function does not necessarily increase monotonically when we move away from the global minimum distortion candidate. This could result in a significant loss in estimation accuracy, and hence compression performance as compared to Full Search. This problem can be alleviated by incorporating prediction into a fast-search algorithm. We can predict the the current motion vector (MV) from previously coded motion vectors that represent spatially or temporally neighbouring macroblocks. Localizing the search origin in this way reduces the possibility of getting trapped in local minima, as the predicted motion vector is usually closer than the vector (0, 0) to the global minimum candidate. If the predicted MV is accurate, we can quickly find the “optimal” MV by searching a relatively small neigbourhood.

Nearest Neighbours Search (NNS) makes use of the above-mentioned motion vector prediction method and highly localized search pattern to give estimation accuracy approaching that of Full Search in many applications but with much lower computational complexity. The algorithm offers a more reliable MV prediction that detects potential non-motion changes in the video sequence. In the algorithm, we predict an MV based on previously coded MVs and transmit the difference (MVD) between the current MV and the predicted MV. NNS exploits this property by giving preference to MVs that are close to the predicted MV, and thus minimize the MVD. Figure 9-9 shows an example of search positions of NNS, which first evaluates the SAD at the search origin (labeled ‘0’) and then calculates the SAD at the locations of the predicted MV as well as the SADs of the surrounding points in a diamond shape (labeled ‘1’). If the SAD at ‘0’ or the center of the diamond is lowest, the search terminates, otherwise the location that gives the smallest SAD is chosen (double-circled ‘1’ in Figure 9-9), and becomes the center of a new diamond-shaped search pattern (labeled ‘2’) and the search process continues. In the example, the next search center is double-circled ‘2’ and the final selected location is double-circled ‘3’.

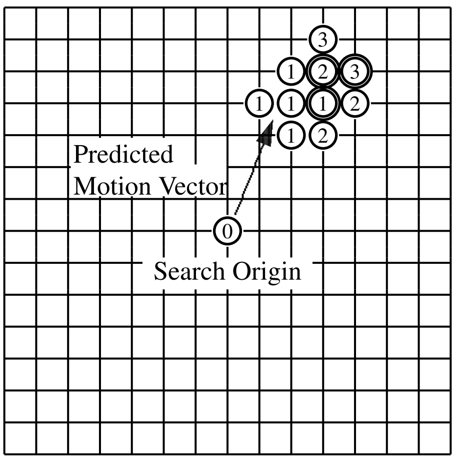

Figure 9-9. Nearest Neighbours Search (NNS)

Others

Many other fast search algorithms have been proposed. More popular ones include Binary Search (BS), Two Dimensional Logarithmic Search (TDL), Four Step Search (FSS), Orthogonal Search Algorithm (OSA), One at a Time Algorithm (OTA), Cross Search Algorithm (CSA), and Spiral Search (SS). In each case, we can evaluate its performance by comparing it with Full Search. We can compare the time an algorithm used and the compression ratio achieved against that of Full Search. Sometimes an algorithm may be good for certain applications but not for others. For example, algorithms such as Hierarchical Search are more easily to be implemented with customized hardware than others.

9.7 Frame Types

There are two types of video frames concerning motion compensation. An **intraframe** or **I-frame** is a frame that is encoded using only the information from within that frame. On the other hand, **inter-** or non-intra frames are encoded using information from within that frame as well as information from other frames. Inter-frames can be further classified as P-frames and B-frames.

9.7.1 Intraframes (I-frames)

Intra frames are coded without motion estimation and compensation. That means it is encoded spatially with no information from any other frame. In other words, no temporal processing is performed outside of the current picture or frame. Coding an I-frame is similar to coding an image in JPEG. Compressing a video using I-frames only is similar to the techniques in motion JPEG. The compression ratio obtained is in general significantly lower than that of compressing with inter-frame coding.

Spatial prediction can be used in intra-frame coding. In spatial prediction, we make use of previously-transmitted samples to predict an image sample in the same image or frame. Figure

9-10 shows a pixel d that is to be encoded. If the pixels in the frame are processed in the order from left to right and top to bottom, then the neighbouring pixel values at a, b, and c have been processed and are available in both the encoder and decoder. The encoder predicts a value $P(d)$ for the current pixel d based on some combinations of previously coded pixel values, $I(a), I(b)$, and $I(c)$. It then subtracts the actual pixel value $I(d)$ of d from $P(d)$ and encodes the residual (the difference resulted from the subtraction). The decoder uses the same prediction formula and adds the decoded residual to construct the pixel value.

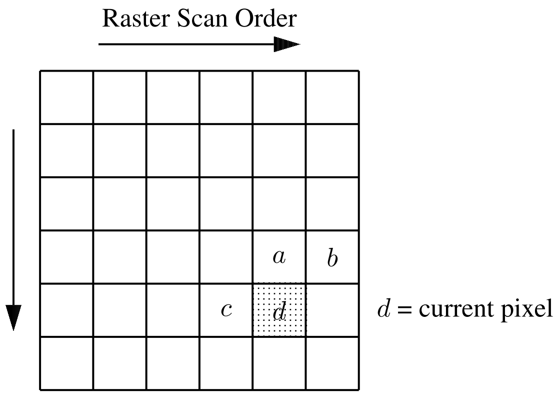

Figure 9-10. Spatial Prediction

The following is an example of spatial prediction for Figure 9-10. In the example, pixel c has a larger weight than b and a because it is the most recently scanned pixel. We also assume that the encoding of the residuals is a lossless process, which means that the pixel values can be recovered exactly from the decoded residuals.

Example 9-1 Spatial Prediction for Figure 9-10.

Encoder prediction: $P(d) = \frac{2I(c) + I(b) + I(a)}{4}$
Residual: $R(d) = I(d) - P(d)$ is encoded and transmitted.

Decoder decodes $R(d)$, and
obtain $P(d)$ using same prediction formula: $P(d) = \frac{2I(c) + I(b) + I(a)}{4}$
Reconstruct sample value at d: $I(d) = R(d) + P(d)$

If the residual encoding involes lossy operations such as quantizations, the decoded sample values, $I'(a), I'(b)$, and $I'(c)$ may not be identical to the original sample values, $I(a), I(b)$, and $I(c)$ of pixels a, b, and c. In this case, the decoder does not know the values of $I(a), I(b)$, and $I(c)$ and if we still use $I(a), I(b)$, and $I(c)$ to calculate $P(d)$ in the encoder as it does in the example, the process could lead to a cumulative error between the encoder and decoder. Therefore, in this situation, the encoder should first reconstruct the sample values $I'(a), I'(b)$, and $I'(c)$ from the previous residuals before calculating the current prediction $P(d)$. The encoder uses the reconstructed sample values to form the prediction in the above example:

$$\begin{aligned} P(d) &= \frac{2I'(c)+I'(b)+I'(a)}{4} \\ R(d) &= I(d)-P(d) \\ R'(d) &= Decode(Encode(R(d))) \\ I'(d) &= P(d)+R'(d) \end{aligned} \qquad (9.3)$$

In this way, just like what we discussed in Section 9.5, where we obtain the reference frame by reconstructing the macroblock rather than using one from the video source, the encoder and decoder use the same sample values in calculating $P(d)$, and the cumulative error can be avoided. The prediction equation of (9.3) is a special case of the more general linear prediction:

$$P(d) = C_aI'(a) + C_bI'(b) + C_cI'(c) \qquad (9.4)$$

where C_x is a constant for pixel x. In the example of (9.3), the constants for pixels a, b, and c are 0.25, 0.25, and 0.5 respectively. For things to work properly, the sum of the constants in the linear prediction of (9.4) must be equal to 1. If the predicted values I' have been scaled in the process, the original sample value $I(d)$ of the current pixel d must be scaled accordingly before subtracting $P(d)$ from it to form the residual $R(d)$.

In practice, we may perform intra-prediction after DCT transformation at a block-based level with block size 8×8. Very often, the low-frequency DCT coefficients of neighbouring blocks are correlated. Therefore, we can predict the DC coefficient (i.e. F_{00} of Chapter 6) and AC coefficients of the first row and column ($F_{0i}, F_{i0},\ 0 < i < 8$) from neigbouring coded blocks. Table 9-1 shows the DCT coefficients for each of the four luma 8×8 block of a macroblock. The data are obtained from the same PPM file, "beach.ppm" that we have used for testing in previous chapters. The DC coefficients (629, 637, 653, and 662) are clearly similar but it is less obvious whether there is correlation between the first row and first column of the AC coefficients in the blocks.

Table 9-1 Four 8×8 luma blocks of a DCT Macroblock

629	-5	-1	-1	1	-1	-2	-1	637	-5	0	1	2	0	0	-1
-8	-5	-1	0	0	-1	-2	-2	-4	2	-1	-2	-1	1	0	0
2	-3	-1	-1	-4	-4	-3	-2	4	-1	1	-2	1	2	1	1
-3	-4	-3	-1	1	-2	-4	-3	2	0	-3	-1	0	1	0	0
-1	-6	0	-1	-1	1	0	0	2	-1	-1	0	-1	1	-1	3
-4	-5	-3	0	2	-1	-1	0	2	-2	-1	-1	-1	0	-1	0
-1	-2	-2	-1	-1	-1	-1	-2	0	0	1	2	1	-1	0	-2
1	-1	0	0	0	-2	-2	-1	-1	-1	1	1	0	0	0	-1
653	1	0	0	-1	1	-1	0	662	-6	1	-1	0	-2	0	0
-9	1	1	-2	-1	0	-1	2	-7	1	1	-2	0	1	0	1
0	1	0	0	0	0	0	-1	1	1	1	1	-1	0	-1	-1
-2	0	-2	1	1	-1	2	0	0	-1	-1	1	-1	-2	0	0
0	-1	-1	0	-1	0	0	0	2	1	1	0	-1	-1	1	0
1	-1	1	-1	0	0	-1	1	0	0	-1	-1	-1	0	0	1
0	-1	1	0	0	0	0	0	-1	-1	-2	-1	0	2	-2	0
0	0	1	2	2	0	1	-2	1	0	-1	0	1	1	-2	0

Suppose F^X_{ij} represent the coefficient of block X at location (i, j). Figure 9-11 presents an example of prediction for the current block D from neigbouring blocks. We predict F^D_{00}, the DC coefficient of the current block from the DC coefficients of the previously coded neighbouring 8×8 blocks, A, B, and C. The question is which block should we choose to form the prediction. *Should we choose A, B or C, or a combination of them?* A simple solution is to use either B or C as they are adjacent to the current block D. This is shown in Figure 9-11. In determining whether we should choose B or C, we can examine the *gradients* of the DC values between the blocks. We choose the block that gives the smaller gradient. That is, the choice of prediction is determined by:

$$\begin{array}{l} \text{if } \; |F^A_{00} - F^C_{00}| < |F^A_{00} - F^B_{00}| \\ \qquad \text{Predict from Block B;} \\ \text{otherwise} \\ \qquad \text{Predict from Block C;} \end{array} \tag{9.5}$$

The prediction value P^D_{00} of block D is set to the DC coefficient of the chosen block and is subtracted from the actual DC value of the current block. For example, if block C is chosen,

$$\begin{array}{l} P^D_{00} = F^C_{00} \\ R^D_{00} = F^D_{00} - P^D_{00} \end{array} \tag{9.6}$$

The residual R^D_{00} is then coded and transmitted.

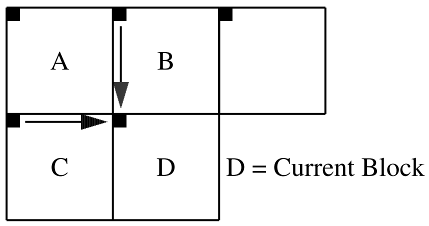

Figure 9-11. Prediction of DC of 8×8 DCT Block

We predict the AC coefficients in a similar way as shown in Figure 9-12, with the first row or column predicted in the direction determined by gradients of DC coefficients of (9.5). For example, if the prediction direction is from Block B, we predict the first row of AC coefficients of Block D from the first row of Block B. If the prediction direction is from Block C, we predict the first column of Block D from the first column of Block C.

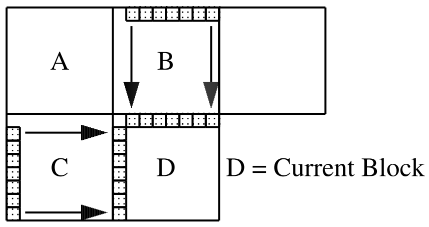

Figure 9-12. Prediction of AC of 8×8 DCT Block

9.7.2 Inter-frames

In addition to the techniques used in intra-frame coding, Inter-frame (or nonintra-frame) coding utilizes motion estimation and compensation to improve the compression of videos; it exploits the temporal correlations to make good predictions to reduce data redundancies. There are two types of inter-frames, **predicted frames** (**P-frames**) and **bidirectional frames** (**B-frames**). The difference between P-frames and B-frames is that they use different kinds of reference frames to make predictions. These two kinds of frames and I-frames usually join in a GOP (Group of

Pictures), which is often required to synchronize the encoder and decoder, when the encoded data are transmitted over a network and errors may occur. We can use an I-frame as a reliable reference for synchronization as it does not require information from other frames to be decoded. I-frames are also important for random access of compressed video files. Because of these, I-frames are also known as key frames or access points.

P-frames

Starting with an I-frame, we can forward-predict a future frame, which is commonly referred to as a P-frame. We can also use a P-frame to predict other P-frames in the future. Therefore, a P-frame is always predicted from a past frame, a frame that has been coded earlier.

As an example, consider a GOP that has 6 frames. The ordering of the frames will be:

$$I, P, P, P, P, P, I, P, P, P, P, P, ...$$

We predict each P-frame in the sequence from the frame immediately preceding it, whether it is an I-frame or a P-frame. The I-frame can be used for synchronization.

B-frames

B-frames are bidirectionally predicted frames. They can be predicted or interpolated from earlier and/or later frames. That is, we not only search a past frame to find the best match but also a future frame to find the optimal result. It is easy to understand making predictions using information of a previously coded frame. *But how do we make predictions from a future frame, something that has not happened?* The trick is that the terms “past” and “future” are artificial; we call the frames occurred before the current frame the past frames and frames occur after the current one, the future frames. To make predictions from a future frame, we must look ahead and buffer it, encoding and decoding it for it to be used as a reference frame. This is called backward prediction. Therefore, if video compression and decompression is to be done in “real time”, there must be a time delay between the compression and decompression process if B-frames are used. Figure 9-13 shows various prediction modes for inter-frames.

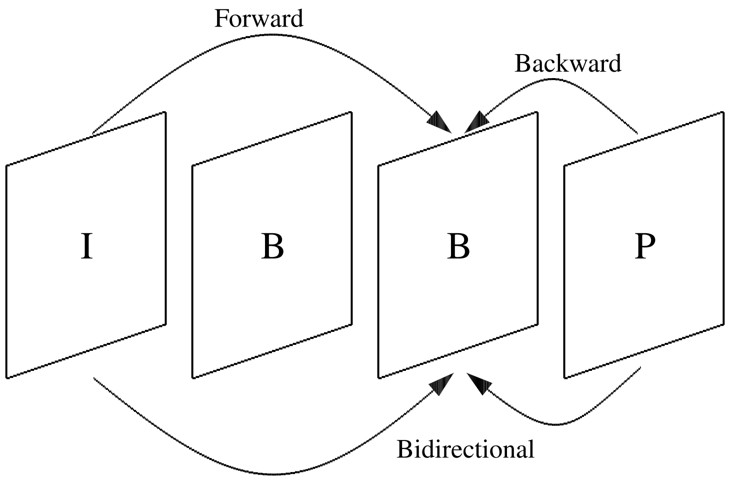

Figure 9-13. Prediction Modes of Inter-frames

9.8 Typical GOP Structure

Figure 9-14 shows a typical group of picture (GOP) structure, “IBBPBBP...”. We can categorize the I-frames and P-frames as *anchor frames*, because they may be used as reference frames in the coding of other frames. On the other hand, B-frames are not anchor frames as they are never used as a reference. Note that the structure starts with an I-frame; it is essential to start coding with an I-frame as there is no previous information that can be used for reference in motion estimation.

Figure 9-15 shows a prediction scheme of the GOP structure. We use the I-frame to predict the first P-frame, and use these two frames (I and P) to predict the first and second B-frame. Then we use the first P-frame to predict the second P-frame. The first and second P-frames are joined to predict the third and fourth B-frames.

As you can see from the prediction scheme, we need the fourth frame (P-frame) to predict the second and the third (B-frames). So we need to transmit the P-frame to the decoder before the B-frames and consequently we have to delay the transmission in order to keep the P-frame or to buffer the frames.

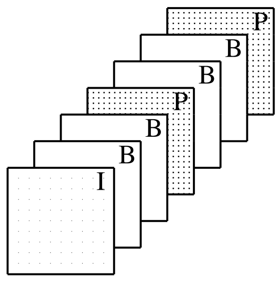

Figure 9-14. A Typical Group of Pictures (GOP)

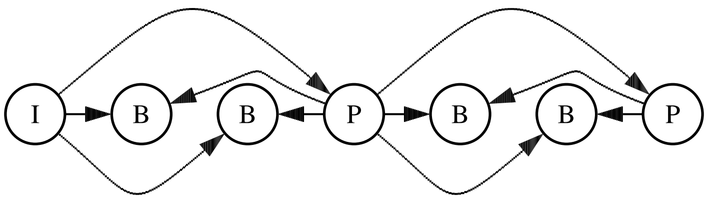

Figure 9-15. A Prediction Scheme for GOP of Figure 9-14

9.9 Rate Control

In many video compression applications, compressed data are sent over a network as bitstreams. The traffic in a large network is usually bursty. It may be idle for some of the time but may have congestion at some other time. This may make available bandwidth to clients change from time to time. Also, if the encoding parameters of a video codec are kept constant, the number of coded bits may change for each macroblock, depending on the video content; this will lead to variations of the output bit rate of the encoder. Typically, high-motion scenes or scenes with fine details generate more bits and low motion or coarse-detail scenes produce fewer bits. These can cause problems in delivering video data and it is necessary for the video encoder to adjust the bit rate to match the available bit rate of the transmission channel.

One simple way to control bit rate is to buffer the encoded data before transmission, which can 'smooth' the impact of fluctuations in the available bandwidth. Figure 9-16 shows a block diagram explaining this mechanism. The variable bit rate output is buffered by a queue, which is first-in first-out (FIFO). The data in the queue are deleted at a constant rate to match the available bandwidth of the delivering channel. Another queue at the receiving end of the channel is used to buffer data to the decoder which deletes the data from the queue at variable bit rate; this is because for the same number of input bits, the decoder may produce a different number of output bits depending on the context of the data. If the decoder needs to produce data at a constant frame rate, it has to consume input bits at a variable rate to counter the variations in the amount of data produced for a given number of bits. Of course, if the fluctuations in the channel capacity or the data context is too large, queues with limited size may not be able to maintain a constant bit rate, but at least the queues can 'smooth' the fluctuations.

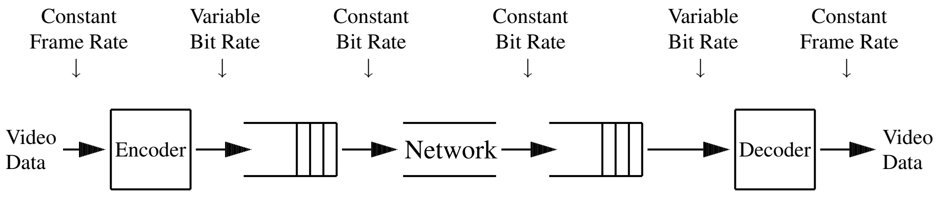

Figure 9-16. Controlling Bit Rate Using Queues

Another way to control the bit rate of the bitstream generated by a video encoder is to vary the encoding parameters from time to time in order to maintain a target bit rate. One important parameter that can change the encoding bit rate is the quantization parameter (Q_p). The encoder can control the bit rate by analyzing the rate at which the encoder is producing data and comparing it with the desired target bit rate. If the encoder is producing too much data, it simply raises Q_p. If too little data is being produced, it lowers Q_p. Note that the larger the Q_p, the better the compression but the lower the quality. There are many different algorithms for varying Q_p depending on the criteria of performance. Some of these algorithms also use the option of dropping frames of video as well as adjusting Q_p. The trade-off of the quality of each frame with the jerkiness in the video caused by frame dropping determines the extent to which we adjust Q_p and the number of frames we want to drop. Alos, the bit rate profiles and characteristics of these algorithms will differ, and often the choice of algorithm is dependent on the network and target application.

9.10 Implementations

We shall discuss the implementation of motion estimation and motion compensation in the next few chapters. We do not intend to cover many of the search algorithms and the prediction methods. We shall only discuss two simple cases, the encoding of the residual between two frames without any motion estimation, and the use of Three Step Search (TSS) to improve upon the encoding.