

In choosing a search algorithm to implement motion estimation for video compression, the major issues are the computational complexity and the data I/O.

Estimating Motion for Video Compression

By Faramarz Azadegan

Communicating information among multiple parties is a fundamental property of any society. Today, we communicate via text, speech, audio, and video in everyday situations. However, the amount of information that needs to be communicated becomes huge very fast. For example, video on regular TV has a dimension of 720 samples by 480 lines per frame, which changes 30 times a second. If each color component (red, green, and blue) is represented by 8 bits, we will need to transmit $3 \times 8 \times 720 \times 480 \times 30 = 248,832,000$ bits per second. The only way we can handle such an enormous amount of data is through compression.

The principle underlying compression is based on the fact that there's an inherent redundancy in any source. The role of any compression system is to remove this redundancy without noticeably affecting the content. In other words, the compression system must retain enough information in the compressed part so that the original

source can be recovered within some acceptable level of error.

The crucial question is, What is an acceptable level of error? The answer isn't obvious. It depends on computational complexity, memory requirements, data I/O requirements, compression and decompression delay constraints, available bandwidth for compression, and the actual cost of the compression/decompression device at each end, not to mention market conditions.

The major reason for focusing on motion estimation when considering compression is that, even when it's done properly, it consumes more than 60% of the computational power in a compression system, which is a significant portion of the computation. By reducing this portion, we'll be able to do a lot more with our processing power. Here, we'll narrow the focus to motion estimation for video and, further, to block matching,

one of the most computer and data I/O-intensive operations in video compression.

There are various ways of performing motion estimation. In the case of video compression, you need to keep two things in mind.

BASIC CONSIDERATIONS

First, most motion estimation techniques are concerned only with the translation of objects. Little attention is paid to other forms of motion, such as rotation, zooming, fading, or deformation. Second, in general the motion estimation techniques used in video compression don't actually estimate motion. Instead, they attempt to minimize the cost of a function, which in most cases is determining the absolute error.

When you consider the actual implementation issues, you need to take into account numerous parameters in your decision process. Among them are the cost in bits of transmitting of motion information, the cost of computation (scalar and vector processing), and the data I/O (to deliver previous frame data into local memory for computation).

BLOCK MATCHING

All the compression methods specified in the current video standards use block-matching schemes. In these schemes, each frame is partitioned exhaustively into (overlapping) blocks, usually rectangles. (You can, however, use any other geometric shape that you like for this purpose.) Each block is then compared with the blocks in the previous frame to find the best match based on some distortion measure. Various distortion measures can be used for the comparison. The most common is the "sum of the absolute difference" (SAD).

If a good match exists, it will result in very few nonzero elements in the block difference. That, in turn, will require much less computation to compress the block and a small number of bits to represent it and therefore to be transmitted.

If a good match isn't found, the resulting block difference will be larger. That will require more computation to compress the block and more bits to be transmitted. In other words, you pay a penalty in both computation and bit rate.

There's a trade-off in the size of the block for matching. You can find better matches with smaller blocks, simply because the probability of a match is higher. However, the cost of sending motion vectors for small

blocks is expensive, since the ratio of bits per sample increases. Large block sizes, on the other hand, require fewer bits per sample to transmit the best match, but it's harder to minimize the difference between the current block and the previous frame. In general, an adaptive system that uses both small and large blocks is the best choice.

Next, you have to decide how to perform the search. The major issues are the computational complexity and the data I/O.

COMPUTATIONAL COMPLEXITY

Two components determine the computational complexity: the size of the block and the search region.

Consider that you're interested in matching an $N \times M$ block. To obtain the total SAD value, you need to do $N \times M$ difference calculations and $N \times M - 1$ additions. Let's refer to this as one block distortion calculation.

In its general format, for each block distortion calculation, if you indicate the compared block of the previous frame with the leftmost corner at (rowPt, colPt), you need to do the following double loop when calculating the SAD:

```
SAD = 0;
for (row = 0; row < M; row++){
    for (col = 0; col < N; col++){
        SAD += abs(block[row][col] -
            prev_frame[row+rowPt][col+colPt]);
    }
}
if (SAD < min_SAD) min_SAD = SAD;
    // comparison for update
```

Here, we assumed that min_SAD represents the best match before the current calculation.

You can improve the match with the following:

```
SAD = 0;
for (row = 0; row < M; row++){
    for (col = 0; col < N; col++){
        SAD += abs(block[row][col] -
            prev_frame[row+rowPt][col+colPt]);
        if (SAD < min_SAD) {
            min_SAD = SAD;
            break;
        }
    }
}
```

This approach reduces the calculation, but you need also to consider the following in your design.

In some of the architectures, a vector SAD calculates the SAD of one row (or part of it) in one cycle. Therefore if your comparison also takes one cycle, you could be wasting a lot of cycles for comparison. As a compromise, you might want to unroll one of the double loops and start comparing at a later stage.

If you do that, one conclusion is that the sooner you get to the (nearly) optimum search point, the less computation you need to do. A number of strategies, to be discussed later, try to achieve this.

Assume that your block size is

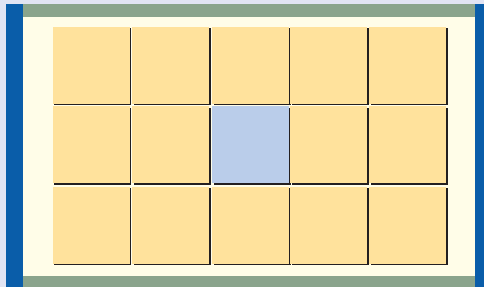


Figure 1: For the current block (blue) in the current frame, the search region in the previous frame, shown here, is $[-32, 31]$ in the horizontal direction and $[-16, 15]$ in the vertical. To obtain the best motion vector, every displacement of the current block in the horizontal and vertical direction must be examined. The examination requires calculating the sum of the absolute difference (SAD) for each displacement.

16×16 and your search region is $[-16, 15]$ in the horizontal and $[-16, 15]$ in the vertical direction. As a result, you'll have $32 \times 32 \times 16 \times 16 = 262,144$ difference calculations and $32 \times 32 \times 255 = 26,120$ additions for only one block. Recall that a D-1/NTSC resolution contains 1,350 blocks per frame at 30 frames per second, and now you can begin to appreciate the burden of motion estimation!

In this example we assumed that only one reference frame—that is, one previously coded frame—is used for prediction (hence the designation “P frame”) in the standards. If you use both the past and the future

ADAPTIVE DIGITAL TECHNOLOGIES, INC.

ipPhoneChip™

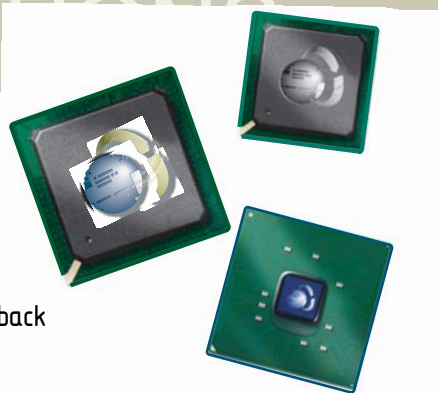
- Conference and/or speakerphone
- ITU Vocoders
- RTP Payload packetization
- Voice Quality Enhancement

High Density Conference Chip

- Port counts up to 512 per chip
- TDM and Packet interfaces
- Noise reduction, voice record and playback
- DTMF Tone detection and generation

VoIP Gateway Chip

- 4, 8, 16, & 32 port options
- Powered by G.PAK™



CHIP SOLUTIONS

chip solutions



Adaptive Digital is a strategic member of the TI's TMS320™ Third Party Program



ADAPTIVE DIGITAL TECHNOLOGIES, INC.
1100 East Hector Street, Suite 210
Conshohocken, PA 19428 USA
www.adaptivedigital.com
610.825.0182

PRODUCT LINE

- Voice Quality
 - G.168-2002 Line EC
 - G.168-2002 Network EC
 - Acoustic Echo Canceller
 - Automatic Gain Control
 - Noise Reduction
 - Acoustic Beamformer
- Telephony
 - DTMF
 - VAD/CNG
 - Tone Relay
 - Conferencing
 - Tone Detect
 - Tone Suppress
 - Tone Generate
- Speech
 - G.729
 - G.729A
 - G.729B
 - G.729AB
 - G.728
 - G.726
 - G.723.1
 - G.722
 - G.711 with appendices 1 & 2
 - ADT 4800
 - ADT 9600
 - GSM AMR
 - MELP
 - LPC

ASK about our Video Products

FREE TOOL go to <http://wizard.adaptivedigital.com>

Adaptive Digital Algorithms run on TI's TMS320C5000™ & TMS320C6000™ DSP Platforms

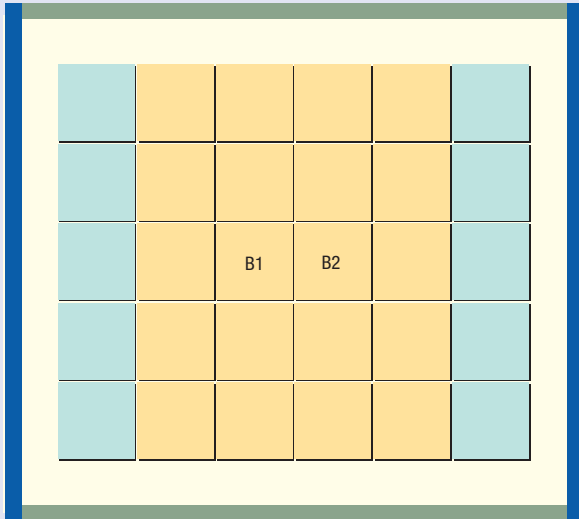


Figure 2: The search region in the previous frame for two consecutive horizontal blocks in the current frame, B1 and B2, is centered on each block. Note that the area of overlap (yellow) is two thirds of the search area. With the introduction of a new block in the horizontal direction, the column of blocks is flushed and the new column of blocks is brought in from memory.

frame (two previously coded frames) for prediction, which is the case for a bidirectionally interpolated frame (B frame), you have twice as many computations.

There are ways to reduce this calculation further.

In one approach, reduction in spatial resolution, the algorithms try to reduce the number of computations by reducing both the size of the block and the number of search points. Generically, we can classify these algorithms under the rubric “hierarchical search.”

In a hierarchical search, a low-pass filtering and sub-sampling process reduces the resolution for both the block and the search area, yielding fewer pixels—say, p pixels. Thus the computation is reduced to $p/(N \times M)$. The penalty will be in the suboptimality of the motion estimation. (You can obtain the subsampled results any way you choose.)

SEARCHING THE REGION

The next step is to decide how big the search region should be. The search region is the area of the previous frame (or, for a B frame, of the previous and the following coded frame) in which we’d like to do the matching.

This area is centered on the current block (Figure 1).

Various techniques exist to perform the search. Examples are the three-step, logarithmic, steepest-descent, hierarchical, and diamond or spiral searches. All of them attempt to reduce the number of points searched by making intelligent decisions about the search strategy for finding the best match. More specifically, a sequence of search points in the search area is adaptively obtained that will give the best estimate possible. The best match found by these techniques, however, isn’t necessarily the optimum match, since the techniques aren’t exhaustive. Instead, these searches provide a compromise between the optimum match and the level of complexity.

The diamond/spiral search is slightly different. It is actually a group of exhaustive search methods that examine each point in the search area. The difference between these methods and the other search techniques is in the path they take to search the region to reach the (nearly) optimum point first.

For example, the starting point for a spiral search could be the point (0, 0), that is, no motion point, since it has the highest probability. The search then continues through the lower probability points.

CANDIDATE MOTION VECTORS

The starting point of your search region could also be a different point, chosen carefully based on prior knowledge of the video. In that case, you would use a candidate, or predictive, motion vector approach.

In these approaches, several candidate motion vectors are examined first. The idea is that these motion vectors are the best choices and will therefore provide the optimum choice fastest. As an argument for these approaches, note that MPEG-2 uses delta coding of motion vectors, indicating that the previous motion vector is a very good candidate for the current block.

You could choose the candidates from the previous blocks or previous frames. The choices become a system designer’s responsibility, and most of these algorithms are proprietary.

Once you select a starting point from your candidate set, the rest of the search could follow a diamond or a spiral pattern centered on each candidate motion vector.

Another approach to reducing the computational requirements is to allocate a threshold for the SAD. Once this value is reached, you declare a good-enough match and exit the double loop. Of course, there’s no guarantee that this value will be reached.

In any motion estimation procedure, the data from

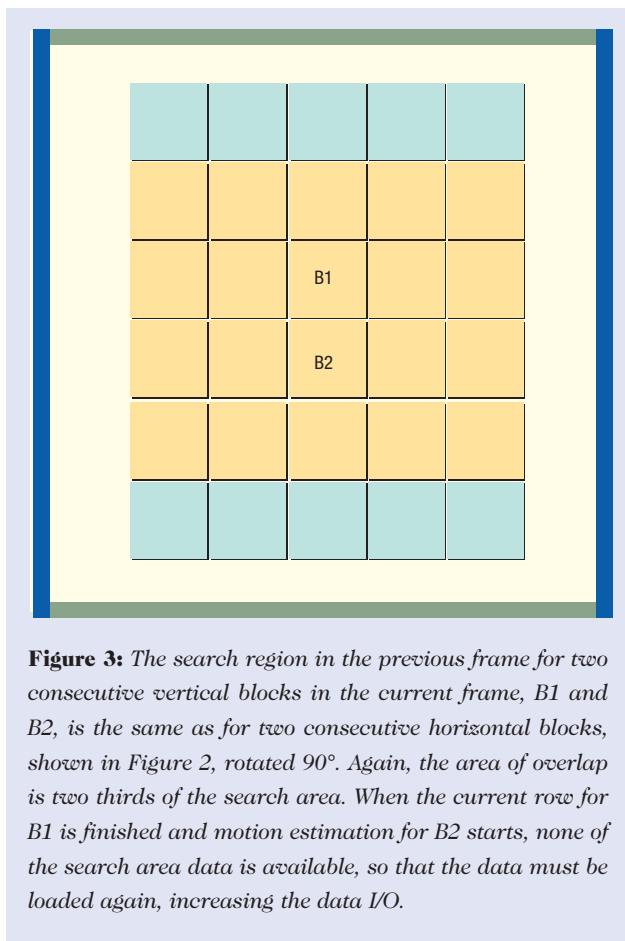


Figure 3: The search region in the previous frame for two consecutive vertical blocks in the current frame, B1 and B2, is the same as for two consecutive horizontal blocks, shown in Figure 2, rotated 90°. Again, the area of overlap is two thirds of the search area. When the current row for B1 is finished and motion estimation for B2 starts, none of the search area data is available, so that the data must be loaded again, increasing the data I/O.

the previous frame should be available for each motion estimation calculation. As a result, the data I/O issues for transferring the previous frame data into local memory for the calculation become very important. In other words, you may have the best computational power possible, but if the data isn't ready to be processed, your processor will be idling.

BUS BANDWIDTH

Therefore it's important to factor in the bus bandwidth of your processor and the local memory size in developing your motion estimation strategy. The problem becomes more difficult if you have multiple processors sharing the same bus to access the remote storage area to bring the required data into the local processor for processing. Furthermore, you have to consider the overhead associated with various requests from the processors and the latency for the responses. Obviously, if your bus accom-

modates a burst mode, it's much better for your design.

There are various techniques for reducing the traffic on the bus. In general, they try to minimize the bus traffic by bringing the required data into local memory as few times as possible. Consider the following case.

You start your motion estimation for block n , and you bring into local memory all the data necessary to perform the motion estimation for this block. After this process is done, you move to block $n + 1$. But there's a huge overlap between the data required for motion estimation for block $n + 1$ and the one for block n (Figure 2). Therefore you need to bring in only a few more columns of data from the previous frame and flush the unwanted data. As you proceed through this process, you bring in data for block $n + 2$, $n + 3$, and so on, and remove the unnecessary data from the local memory, possibly using a circular buffer or an appropriate pointer.

Now consider what happens when you go to the next row of blocks. Since you were previously at the end of the first row, you don't have the necessary data anymore and you must get it again. But, again, a huge overlap exists, this time between two vertical blocks (Figure 3). Unless you have a huge local memory, you've flushed all the data for the upper block, and now you need to bring it back again. This happens over and over, increasing the bus traffic for no good reason.

You might therefore consider various tricks to eliminate this continuous data I/O issue by following a different path in your motion estimation technique. A vertical or a zigzag path, for example, might result in less data I/O.

In general, almost all the techniques employed by various companies follow a variation of those described here, except possibly for very few purely proprietary techniques.

BIBLIOGRAPHY

- P. Kuhn, *Algorithms, Complexity Analysis and VLSI Architecture for MPEG-4 Motion Estimation*, Kluwer Academic Publishers, Boston, London, Dordrecht, The Netherlands, 1999.
- A. Netravali and B. Haskell, *Digital Picture Representation and Compression*, Plenum Press, New York, 1988.

Faramarz Azadegan (faramarz.azadegan@mmxmission.com) is the CEO of MultimediaXmission, Inc. in La Jolla, Calif. He spearheaded the development of the current U.S. standard for HDTV at Philips and, at Toshiba, led the development of the first DVD authoring system, for which his team received an Emmy award. He has held senior management positions at Conexant, Terayon, and Konami.