FULL SEARCH CONTENT INDEPENDENT BLOCK MATCHING BASED ON THE FAST FOURIER TRANSFORM

Steven L. Kilthau, Mark S. Drew, and Torsten Möller
School of Computing Science, Simon Fraser University,
Vancouver, B.C., Canada V5A 1S6
{kilthau,mark,torsten}@cs.sfu.ca

ABSTRACT

In this paper, we present a new algorithm for solving the block matching problem which is independent of image content and is faster than other full-search methods. The method employs a novel data structure called the *Windowed-Sum-Squared-Table*, and uses the fast Fourier transform (FFT) in its computation of the sum squared difference (SSD) metric. Use of the SSD metric allows for higher peak signal to noise ratios than other fast block matching algorithms which require the sum of absolute difference (SAD) metric. However, because of the complex floating point and integer math used in our computation of the SSD metric, our method is aimed at software implementations only. Test results show that our method has a running time 13%-29% of that for the exhaustive search, depending on the size of the search range.

1. INTRODUCTION

The block matching problem is one that occurs in many areas of the image processing, multimedia, and vision fields. In this paper, our focus is on the application of block matching to the computation of motion vectors for video compression. Because of their widespread use, many motion vector compensation algorithms have been developed and are in use today.

Block matching algorithms all attempt to minimize some measure of similarity between a template block of pixels in the current image to all candidate blocks in the reference image within a given search range. The two most popular similarity measures used are the sum of absolute difference (SAD) and the sum of squared difference (SSD). Given a block size B and a displacement vector (u,v) for a candidate block relative to the template block, the metrics are defined as:

$$SAD_{(u,v)} = \sum_{j=0}^{B-1}\sum_{i=0}^{B-1} \left| f_t(i,j) - f_{t-1}(i+u, j+v) \right| \qquad (1)$$

$$SSD_{(u,v)} = \sum_{j=0}^{B-1}\sum_{i=0}^{B-1} \left(f_t(i,j) - f_{t-1}(i+u, j+v) \right)^2 \qquad (2)$$

Because of its lack of multiplications the SAD metric is far more convenient for use in hardware designs, and is therefore used almost exclusively. However, minimizing the SSD metric corresponds to maximizing the peak signal to noise ratio (PSNR), whereas minimizing the SAD metric does not. Therefore, if a maximum PSNR is desired, SSD should be the metric of choice.

All existing block matching algorithms can be roughly grouped into two categories. The first category consists of those algorithms that are not guaranteed to find the best matching block within a given search range, but instead use a heuristic approach to guide the search. These methods examine only a subset of the possible locations within the search range, and hence can be computed very efficiently. Some of the most popular methods are the three step search [8], the two dimensional logarithmic search [6], and their many successful variants such as the one found in [3]. Because of their speed, these suboptimal methods are of great interest. However, they are prone to getting trapped in local minima and thus are not appropriate for applications which require a maximum PSNR.

The second category, and the subject of our focus, consists of those algorithms which are guaranteed to find the optimal matching block within a given search range. In recent years, many algorithms have been developed for this type of search [2, 5, 9, 10, 11]. All of the algorithms in this category achieve their speedup through early elimination of candidate search positions; however they suffer from the fact that their performance depends largely on the content of the image sequence being encoded. Much of the recent research [2, 9, 11] eliminates search positions through application of the Minkowski Inequality:

$$\sqrt[p]{\sum_i |a_i + b_i|^p} \le \sqrt[p]{\sum_i |a_i|^p} + \sqrt[p]{\sum_i |b_i|^p} \qquad (3)$$

For the case $p = 1$, substitutions $a_i = x_i - y_i$, and $b_i = y_i$ yield the following common form:

$$\sum_i |x_i - y_i| \ge \sum_i |x_i| - \sum_i |y_i| \quad . \qquad (4)$$

Many algorithms make use of a pyramid version of Eq.4 for early elimination of candidates. Notice however that for $p = 2$, the Minkowski Inequality requires the computation of the square root. Since the square root operation is extremely expensive, techniques that rely on the Minkowski Inequality to eliminate search positions can not efficiently use the SSD metric, and are instead only able to use the SAD metric. Hence, all methods that require the Minkowski Inequality cannot guarantee a maximum PSNR value, while still maintaining computational efficiency.

One technique that doesn't fit into the above categorization is the phase-correlation method [1]. This technique works by computing the cross-correlation of the template block with the corresponding search range and identifying a set of candidate correlation peaks. The algorithm then evaluates a difference measure at those points and chooses the minimum as the solution to the block matching problem. Although this algorithm cleverly reduces the computational complexity it has been shown to

identify spurious solutions, and as such is not guaranteed to maximize PSNR.

Since our method uses the SSD metric to find the minimum matching block within the given search range, we are guaranteed to find the blocks that maximize the PSNR of the predicted image. Furthermore, our method achieves its speedup regardless of the content of the image sequence being encoded. The details of our algorithm are given in §2, experimental results are given in §3, and we discuss conclusions in §4.

2. THE FFT BLOCK MATCHING ALGORITHM

In order to maximize the PSNR, our algorithm minimizes the SSD metric given in Eq.2. Following a trivial expansion, the mathematical definition of our per-block computation is given by:

$$\min_{\forall u,v} \sum_{j=0}^{B-1}\sum_{i=0}^{B-1}\left[f_t(i,j)^2 + f_{t-1}(i+u, j+v)^2 - 2 f_t(i,j) f_{t-1}(i+u, j+v)\right] \quad (5)$$

Since the term $f_t(i,j)^2$ appears across the entire minimum, it can be removed from the sum without affecting the resulting solution. Removing this term and separating the sum leaves us with the following equation:

$$\min_{\forall u,v} \sum_{j=0}^{B-1}\sum_{i=0}^{B-1} f_{t-1}(i+u, j+v)^2 - 2\sum_{j=0}^{B-1}\sum_{i=0}^{B-1} f_t(i,j) f_{t-1}(i+u, j+v) \quad (6)$$

The FFT Block Matching Algorithm (FFTBMA) that we propose computes Eq.6 using three basic steps:

1. Resize input image to include a zero pad
2. Compute the *windowed sum squared table*
3. Compute a per-block convolution sum

Step 1 is simply to allow convenient calculation of the SSD metric without using conditionals for those search locations that lie outside of the dimensions of the original image. Given a search range of $\pm P$ we apply a zero pad of P pixels around the entire image. This simple preprocess eliminates the need for conditionals within the innermost loops of our algorithm and greatly increases its speed. Similarly, this also improves the performance of the exhaustive search, and as such is used in our implementation of that algorithm as well. For convenience, we assume here that the original dimensions of the image are a multiple of the block size, B. If this is not initially true, the dimensions of the image are increased to compensate for this prior to the application of the zero pad.

Steps 2 and 3 are discussed in §2.1 and §2.2 respectively.

2.1 Windowed Sum Squared Table (WSST)

To compute the first term of Eq.6, we use a variant of the well known summed area table (SAT), introduced in [4]. Given an input image f, a summed area table is a new image f_{SAT} such that

$$f_{SAT}(i,j) = \sum_{k\le i}\sum_{l\le j} f(k,l) \quad (7)$$

Summed area tables can be very easily computed by applying the following recurrence, being careful to set $f(i,j)$ to zero when either of the indices is negative:

$$f_{SAT}(i,j) = [f(i,j) + f_{SAT}(i-1, j) + f_{SAT}(i, j-1) - f_{SAT}(i-1, j-1)] \quad (8)$$

The WSST differs from the SAT in that each pixel needs to represent a *sum of squares*, where the sum extends only over the last $B\times B$ sub-image (window), with B the block size.

Our approach to creating the windowed sum squared table consists of two steps. In the first step we compute a sum squared table (SST), and in the second step we confine the sum to the last $B\times B$ sub-image. Using a variant of Eq.8 would imply that for an 8 bit image of size $W\times H$ we may need to store values as large as $255^2 WH$. For large video streams such as those used in HDTV, this can easily exceed the maximum value representable by a 32 bit integer. So although this algorithm seems to follow directly from the recurrence of Eq.8, care is needed to prevent overflow of intermediate calculations.

To solve the overflow problem, we initially divide the image into blocks of size $B\times B$ and compute an SST over each block. By assumption, the image size is divisible by B, and we assume that $255^2 B^2$ can be represented with a 32 bit integer. The result of this computation is a new image where each block constitutes a sum squared table defined by the following recurrence:

$$f_{SST}(i,j) = [f(i,j)^2 + f_{SST}(i-1, j) + f_{SST}(i, j-1) - f_{SST}(i-1, j-1)] \quad (9)$$

We now need to combine the individual SST's to create the final Windowed SST. For this we note that the sum of squares over any rectangle confined to a single block, with lower left corner (i,j) and upper right corner (k,l) is given by:

$$\sum_{s=l}^{j}\sum_{t=k}^{i} f(s,t)^2 = f_{SST}(k,l) - f_{SST}(i,l) - f_{SST}(k,j) + f_{SST}(i,j) \quad (10)$$

Using Eq.10 we can easily derive the sum of squares (SS) for an arbitrary $B\times B$ region as an SS over rectangles in 4 neighboring blocks. By using coordinates that are local to the block containing the upper right corner we arrive at the following equation:

$$\begin{aligned} f_{WSST}(i,j) &= \sum_{s=j-B+1}^{j}\sum_{t=i-B+1}^{i} f(s,t)^2 \\ &= [f_{SST}(i,j)] \\ &\quad + [f_{SST}(-1,j) - f_{SST}(i-B,j)] \\ &\quad + [f_{SST}(i,-1) - f_{SST}(i,j-B)] \\ &\quad + [f_{SST}(-1,-1) + f_{SST}(i-B,j-B) \\ &\quad - f_{SST}(i-B,-1) - f_{SST}(-1,j-B)] \end{aligned} \quad (11)$$

Fig.1 depicts Eq.11 visually. By applying this equation for every pixel we complete the computation of the windowed sum squared table for the entire frame.

2.2 Per-Block Convolution Sum

In this section we show that computation of the second term in Eq.6 amounts to the evaluation of a correlation sum for each template block, which we evaluate as a convolution sum. For a single candidate block, the second term of Eq.6 is just a dot product with the template block. However, computing this dot product for each of the $(2P+1)^2$ candidate blocks in the search range amounts to a correlation of the template block with the $(2P+B)\times(2P+B)$ region corresponding to the square containing all pixels of all blocks in the search range. In order to efficiently compute the correlation, we will first convert it to a

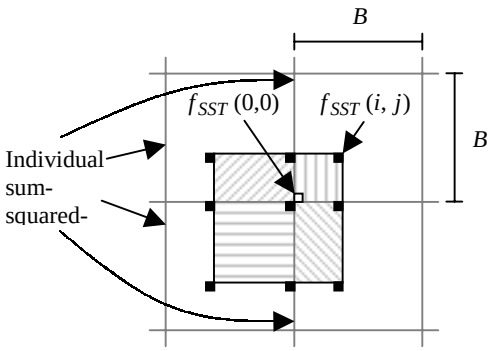

Figure 1. Computation of WSST relative to multiple SST's

convolution and then use the fast Fourier transform (FFT).

For each template block, we create two images of size $(2P+B)\times(2P+B)$. The first image, *template*, corresponds to the template block and is computed by simply multiplying the block by 2, reversing the pixels, and zero padding to the correct size. The pixel reversal effectively changes the correlation sum into an equivalent convolution sum. The second image, *candidates*, corresponds to the square containing all pixels of all candidate blocks in the search range. This square can be copied directly from the reference image. Given these two images, we can compute a new image, *result*, according to the following formula:

$$result = FFT^{-1}\left(FFT(template) \bullet FFT(candidates)\right) \quad (12)$$

Since the convolution that we have performed is cyclic, the first $B-1$ rows and columns of *result* will contain wrap-around data that should be discarded. This leaves us with a usable portion of the image of size $(2P+1)\times(2P+1)$. Notice that this corresponds to one solution for each of our $(2P+1)^2$ search locations, and is exactly what we desire.

Given that we have previously computed the windowed sum squared table, we can now easily find the minimum. Assuming that our template block is at offset (x, y) we simply perform a linear pass over the *result* image, evaluating the minimum matching block according to the following formula:

$$\min_{i,j}\left(f_{WSST}(x+i-P, y+j-P) - result(i,j)\right) \quad (13)$$

where $i, j \in [B-1, 2P+B-1]$. It is important to note that $result(i, j)$ must be rounded to the nearest integer before it can be combined with a value from the windowed sum squared table. As a last step, we simply need to correct the offset to account for the cyclic nature of the convolution. Given that Eq.13 identifies the pair (i, j) as the location of the minimum match, the resulting motion vector corresponding to the minimum matching block is then given by:

$$(mvi, mvj) = (i-P-B+1, j-P-B+1) \quad (14)$$

2.3 Running Time Analysis

In our discussion of running time, we will assume without loss of generality that the dimension of the original image is $N \times N$, where N is divisible by B.

We will first develop an expression for the running time of the exhaustive search. Since full search algorithms are generally content dependent, there are certain cases where they will all exhibit the same worst case running time. For each block, the exhaustive search requires the computation of the SAD or SSD metric at $(2P+1)^2$ locations. This results in an $O\left(B^2(2P+1)^2\right)$ algorithm for each block. There are a total of N^2/B^2 blocks, so the exhaustive search has a total per-frame running time of:

$$O\left(N^2(2P+1)^2\right) \quad (15)$$

The FFT block matching algorithm that we have presented consists of two steps. In the first step we construct the windowed sum squared table. Since this amounts to only two linear passes over the entire image, this step takes only $O\left(2N^2\right)$ per frame. In the second step we perform a per-block convolution. Each convolution consists of three applications of a $(2P+B)\times(2P+B)$ real FFT, as well as $(2P+B)^2/2$ complex multiplications. Therefore this step has a running time of $O\left((2P+B)^2 \log_2(2P+B)\right)$ per block. A precise statement of the total running time of the FFT block matching algorithm is then:

$$O\left(N^2\left(\left((2P+B)^2 \log_2(2P+B) + 3(2P+B)^2\right)/B^2 + 2\right)\right) \quad (16)$$

For practical scenarios, asymptotic analysis is of no interest. To determine the cases for which our algorithm outperforms the exhaustive search, we analyze the following equation:

$$(2P+1)^2 \geq C\left(\left((2P+B)^2 \log_2(2P+B) + 3(2P+B)^2\right)/B^2 + 2\right) \quad (17)$$

We can safely assume that $R = 2P/B \in [1,4]$. Substituting R into Eq.17 and simplifying results in the following expressing:

$$B^2/C \;\geq\; 2/R^2 + (1+1/R)^2(\log_2[(1+R)B] + 3) \quad (18)$$

In §3 we show that the FFT block matching algorithm outperforms the exhaustive search for all relevant block sizes and a range of C, including values that characterize inefficient FFT implementations.

3. EXPERIMENTAL RESULTS

Our experiments test the performance of the FFT block matching algorithm against that of the exhaustive search. Both algorithms have been implemented using C/C++. For the FFT computation we use a publicly available package called the Fastest Fourier Transform in the West (FFTW) [7]. None of the code or libraries that we have used contain any machine specific instructions or assembler routines that would give either algorithm an unfair advantage. Our test platform is an AMD Athlon 900 MHz with 512 MB of RAM. All timings include reading the images, constructing the necessary data structures, and computing all motion vectors for each frame of the input image sequence.

Data Set	Frames	Width	Height	Motion
Mother	199	176	144	slow
Carphone	74	176	144	medium
Football	160	360	240	fast

Table 1. Data sets used in the experiments.

Both algorithms have been tested on three data sets. The dimensions, number of frames, and type of motion classification of each are given in Table 1. The motion classification is shown as slow, medium, or fast, and indicates the degree of motion contained within the image sequence. For all tests, a block size of 16 is used, and only the luminance channel is considered.

The performance of each algorithm is measured using

execution time, PSNR, and the number of errors, E. The number of errors needs to be considered because although the FFT block matching algorithm is mathematically exact, the round-off error produced by the many computations used in the convolution sum will infrequently lead to a non-optimal match. We have observed that this occurs less than 0.5% of the time in practice, and when the result is non-optimal, the computed motion vector is usually matched to the next most optimal block. All data sets were tested using multiple values of the search range, P .

	Exhaustive Search			**FFT BM Algorithm**		
P	Time	PSNR	E	Time	PSNR	E
+/-8	14.06	38.20	0	4.01	38.20	74
+/-16	52.13	38.31	0	10.89	38.31	86
+/-24	114.51	38.34	0	15.19	38.34	52
+/-32	201.19	38.36	0	33.03	38.36	74

Table 2. Test results for *mother* image sequence.

	Exhaustive Search			**FFT BM Algorithm**		
P	Time	PSNR	E	Time	PSNR	E
+/-8	5.19	32.00	0	1.49	32.00	27
+/-16	19.21	32.02	0	4.01	32.02	36
+/-24	42.22	32.02	0	5.59	32.02	31
+/-32	74.18	32.03	0	12.36	32.03	25

Table 3. Test results for *carphone* image sequence.

	Exhaustive Search			**FFT BM Algorithm**		
P	Time	PSNR	E	Time	PSNR	E
+/-8	36.91	23.13	0	10.52	23.13	3
+/-16	136.65	23.46	0	28.38	23.46	2
+/-24	300.14	23.55	0	40.14	23.55	0
+/-32	527.42	23.60	0	87.10	23.60	2

Table 4. Test results for *football* image sequence.

As Tables 2, 3, and 4 show, the FFT block matching algorithm greatly outperforms the exhaustive search while still obtaining the same PSNR value, with running times of 13%-29% of that of the exhaustive search, dependent on P . Fig.2a shows a plot of the timings for the *football* image sequence, as well as the analytic curves in the inequality of Eq.17 (here we used $C = 2.75$). In Fig.2b, we have solved Eq.18 for B , varying C and R . Since we have measured C to be between 2.2 and 3.0 in our implementation, Fig.2c shows a slice through the plot in Fig.2b for $C = 3$. With such a value of C , it is clear that the FFT block matching algorithm convincingly outperforms the exhaustive search for our previously defined range of $R \in [1, 4]$. Even for much less efficient implementations of the FFT, it is clear that our algorithm is faster.

In our tests, using $P = 24$ consistently gives the best running times. This occurs because P is relatively large and the dimension of the FFT is then 64×64 , which runs extremely fast because it is a power of 2. As we have mentioned, the FFT block matching algorithm occasionally fails to identify the best matching block due to round-off error, but as the PSNR indicates, the effect of this error is negligible.

4. CONCLUSIONS

Our new FFT-based block matching algorithm employs a novel data structure, the Windowed-Sum-Squared-Table, and exploits the FFT in its computation of the SSD metric. Because it is independent of image content, our algorithm runs faster than existing full search algorithms, with speeds of 13%-29% of the exhaustive search in practice. The FFT block matching algorithm is not heuristic-based and thus can consistently identify the best matching blocks, maximizing PSNR. The algorithm is well suited for software implementations requiring very low bit rates.

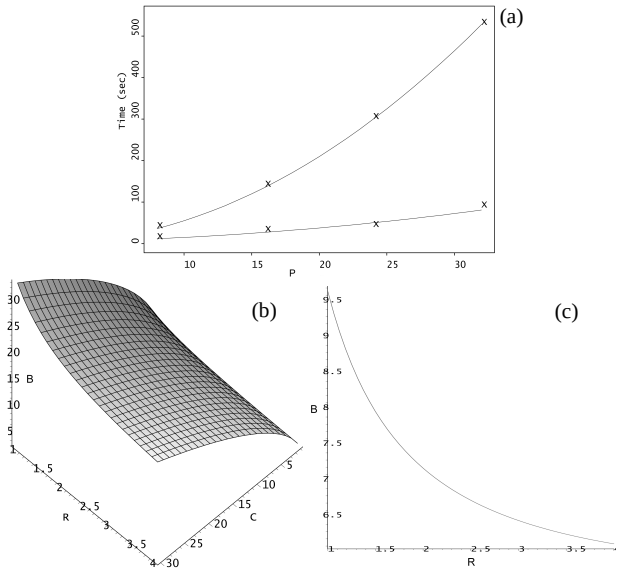

Figure 2. (a) Timings for *football* image sequence. **(b)** Solutions of Eq.18. **(c)** Slice through Fig.2b at $C = 3$.

5. REFERENCES

[1] C. D. Kuglin and D. C. Hines, "The phase correlation image alignment method," in *Proceedings of the 1975 IEEE International Conference on Systems, Man and Cybernetics*, pp 163-165, 1975.

[2] C.-H. Lee and L.-H. Chen. "A fast motion estimation algorithm based on the block sum pyramid," *IEEE Transactions on Image Processing*, 6(11):1587-1591, 1997.

[3] D. W. Zhang, I. Ahmad, M. Liou. "Adaptive motion search with elastic diamond for MPEG-4 video encoding," *Proceedings of International Conference on Image Processing*, 2001.

[4] F. C. Crow, "Summed-area tables for texture mapping," *Computer Graphics (Proc. of Siggraph)*, 18(3):207-212, 1984.

[5] H.-C. Huang and Y.-P. Hung, "Adaptive early jump-out technique for fast motion estimation in video coding," *Graphical Models and Image Processing*, 59(6):388-394, 1997.

[6] J. R. Jain and A. K. Jain. "Displacement measurement and its application in interframe image coding," *IEEE Transactions on Communications*, COM-29(12):1799-1808, 1981.

[7] M. Frigo and S. Johnson, "FFTW: An adaptive software architecture for the FFT," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, vol. 3, pp 1381-1384, 1998.

[8] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro. "Motion-compensated interframe coding for video conferencing," *Proceedings of National Telecommunications Conference*, vol. 4, pp G5.3.1-G5.3.5, 1981.

[9] W. Li and E. Salari. "Successive elimination algorithm for motion estimation," *IEEE Transactions on Image Processing*, 4(1):105-107, 1995.

[10] Y.-C. Lin and S.-C. Tai, "Fast full-search block-matching algorithm for motion-compensated video compression," *IEEE Transactions on Communications*, 45(5):527-531, 1997.

[11] Y.-S. Chen, Y.-P. Hung, and C.-S. Fuh. "A fast block matching algorithm based on the winner-update strategy," in *Proceedings of the Fourth Asian Conference on Computer Vision*, vol. 2, pp 977-982, 2000.