ABSTRACT

The JVT/H.264 video coding standard can achieve considerably higher coding efficiency than previous standards. Unfortunately this comes at a cost in considerably increased complexity at the encoder mainly due to motion estimation and mode decision. In this paper, we propose a new scheme to jointly optimize motion estimation and mode decision. Simulation results show that we can achieve up to 90% complexity reduction while maintaining coding efficiency.

1. INTRODUCTION

The new JVT (or H.264, MPEG-4 AVC) [1] video coding standard has gained more and more attention recently, mainly due to its high coding efficiency and minor increase in decoder complexity. The encoder complexity, however, is greatly increased. Among all modules in the encoder, motion estimation (ME) and mode decision contribute most of the complexity, especially when Rate-Distortion Optimization (RDO) is used. In this paper, we propose new algorithms to alleviate the encoding complexity due to ME and mode decision, while maintaining coding efficiency. All the work is discussed within P (inter) frames, while this framework can be easily extended to B (bi-predictive) frames.

JVT supports multiple reference frames and various block sizes for ME. It uses tree-structured hierarchical macroblock partitions. Inter-coded 16x16 pixel macroblocks can be broken into macroblock partitions, of sizes 16x8, 8x16, or 8x8. 8x8 partitions are also known as sub-macroblocks. Sub-macroblocks can be further broken into sub-macroblock partitions, of sizes 8x4, 4x8, and 4x4. To simplify our explanation, we will call these block types as 16x16, 16x8, 8x16, 8x8, 8x4, 4x8, 4x4, where 8x8, 8x4, 4x8 and 4x4 is a partition of sub-macroblock 8x8. Assuming that we have M block types, N reference frames and that the search range for each reference frame and block type is the same and equal to ±W, this would imply that we need to examine $N \times M \times (2W + 1)^2$ positions compared to only $(2W + 1)^2$ positions for a single reference/block type. In addition, quarter pel motion vector precision is used. These three features have made ME in JVT very complicated. When dealing with mode decision for interframes, besides the multiple inter-modes with regards to block types, JVT also supports skip mode and intra mode. Intra mode has two block types: 4x4 and 16x16. We shall call them intra4x4 and intra16x16 for future reference. Intra4x4 supports 9 modes and intra16x16 supports 4 modes.

The current JVT reference software [2] is based on a RDO framework for both ME and mode decision. ME is separately considered from mode decision. ME is first performed for all block types of inter modes, then the mode decision is made by comparing the cost of each inter mode and intra mode. The mode with the minimal cost is selected as the best mode. In ME, the motion vector (MV) and reference frame are selected by minimizing the cost

$$J_{\text{motion}}(\overline{m}, \text{REF}) = \lambda_{\text{motion}} \cdot (SAD(s, c(\text{REF}, \overline{m})) + R(\overline{m} - \overline{p}) + R(\text{REF})),$$

where $\lambda_{\text{motion}}$ is the Lagrangian multiplier, $R(\overline{m} - \overline{p})$ represents the bits used for coding MV and $R(\text{REF})$ is the bits for coding RE. The SAD (Sum of Absolute Differences) is computed as:

$$SAD(s, c(\text{REF}, \overline{m})) = \sum_{x=1,y=1}^{B_1,B_2} |s[x,y] - c[x-mx,y-my]|,$$

where $s$ is the original video signal and $c$ is the referenced signal, $B_1$ and $B_2$ are the vertical and horizontal dimensions of block size, which can be 16, 8 or 4. The MV is found by searching all 7 possible block types within the N previous decoded pictures. This results in $7N$ combinations of block sizes and reference frames. For each block size and reference frame, the MV is determined by initially performing a full search on integer-pel positions followed by subpel refinement. The subpel motion search is performed in two steps: 1/2 pel search and 1/4 pel search. This is illustrated in Figure 1, where capital letters represent integer positions, Arabian numbers represent 1/2 pel positions and lower case letters represent 1/4 pel positions. Assuming that the best integer search position is E, in the first step 1/2 pel positions 1,2,3,4,5,6,7,8 are searched. Assume that 7 is the best position, in the second step 8 1/4 pel positions a,b,c,d,e,f,g,h are searched. so totally 16 subpel positions are checked. The mode decision is made by minimizing

$$J_{\text{mode}}(s, c, \text{MODE}) = \lambda_{\text{mode}} \cdot \text{SSD}(s, c, \text{MODE}) + R(s, c, \text{MODE}).$$
In the above equation, SSD denotes Sum of Square Difference between the original signal and reconstructed signal. MODE indicates a mode out of a set of potential macroblock modes: [SKIP, 16x16, 16x8, 8x16, 8x8, 4x8, 4x4, Intra4x4, Intra16x16], and $R(s, c, MODE)$ is the number of bits associated with choosing $MODE$, including the bits for the macroblock header, the motion, and all DCT coefficients.

$$J_{mode}(16, 16) = J_{mode}(8, 8) = J_{mode}(4, 4)$$

**Fig. 1. Subpel Motion Estimation**

In this paper, we propose a new algorithm to jointly optimize mode decision and ME. Unlike the reference software where ME is separated from mode decision[2], in our scheme, ME is considered as a part of mode decision. That is, no ME is required for a particular mode if that mode is eliminated by the mode decision algorithm. For inter-modes, since ME takes most of the time, we also propose new schemes to reduce its complexity. Various fast ME algorithms have been proposed in the past, but most of them are focused on integer pel motion search [3], while little work has been done for subpel motion search [4]. In our paper, we shall focus on subpel ME.

The rest of the paper is organized as follows. We first discuss fast mode decision in Section 2. The algorithm related to fast ME for the inter-modes is given in Section 3, followed by simulation results and the conclusion. The entire discussion is focused on inter-frames (P frames) only.

**2. FAST MODE DECISION**

In JVT, for P frames, we need to select the best mode from a set of potential modes: [SKIP, 16x16, 16x8, 8x16, 8x8, 8x4, 4x8, 4x4, intra4x4, intra16x16]. Examining all possible combinations of modes, especially due to ME on various block types and reference pictures, can be a big burden on the encoder. In this section, we shall propose a new algorithm to alleviate the complexity on mode decision. The algorithm is to reduce the number of potential modes and to restrict the set of past coded reference picture for ME. In this way, we will eliminate ME for some block types and reference pictures. In addition, our algorithm also decreases the number of tested intra modes. To simplify the explanation, we divide the modes into two categories: inter modes and intra modes, where inter modes include SKIP mode and different block types (in particular, 16x16, 16x8, 8x16, 8x8, 8x4, 4x8, 4x4) and intra modes consist of intra4x4 and intra16x16.

In our algorithm, ME is jointly done with mode decision. ME is only performed within a particular inter-mode if that mode is checked according to our scheme. For inter modes, SKIP mode refers to the 16x16 mode where no motion and residual information is encoded. So no motion search is required and it has the lowest complexity. In our algorithm, we will differentiate it from other block types and give it the highest priority. As for the decision on block types, our algorithm is based on whether the error surface versus block size is monotonic, that is, whether the current macroblock has the same tendency of using smaller block size (sub-macroblock partition) or larger block size. The error surface is built by initial 3 modes (block sizes): 16x16, 8x8, and 4x4. Here 8x8 means that the entire macroblock is examined using only 8x8 partitions, and 4x4 means that the entire macroblock is examined using only 4x4 partitions. We call the error surface monotonic if $J_{mode}(16 \times 16) < J_{mode}(8 \times 8) < J_{mode}(4 \times 4)$ or $J_{mode}(16 \times 16) > J_{mode}(8 \times 8) > J_{mode}(4 \times 4)$. The decision of whether to test other modes 16x8, 8x16, or finer sub-macroblock partition (optimal selection of 4x4, 8x8, 8x4 and 4x4 partition within sub-macroblocks) is based on the error surface built by these 3 modes. If the error surface is not monotonic, all other modes need to be tested. If the error surface is monotonic, only modes (block sizes) between the best two modes are tested. For example, if the best two modes are 16x16 and 8x8, which implies that the macroblock tends to use larger block partitions, only 16x8 and 8x16 are further tested; if the best two modes are 8x8 and 4x4, this implies that the macroblock tends to use smaller block partitions (or sub-macroblock partitions), and only 8x4 and 4x8 modes are further tested. The algorithm related to block type partitioning is as follows, where $T_1$ and $T_2$ are thresholds which will be explained afterwards.

- **step1**: check SKIP mode. if $J_{mode}(SKIP) < T_1$, select SKIP as best mode, stop; otherwise go to step2;
- **step2**: check 16x16 and 8x8. if $J_{mode}(SKIP) < J_{mode}(16 \times 16)$ & & $(J_{mode}(8 \times 8) < J_{mode}(4 \times 4))$, go to step7; otherwise, go to step3;
- **step3**: check 4x4; if $(MinJ_{mode} = J_{mode}(8 \times 8)) \& \& (MaxJ_{mode} = J_{mode}(16 \times 16))$, go to step4; if $MaxJ_{mode} = J_{mode}(4 \times 4)$, go to step5; if $MaxJ_{mode} = J_{mode}(16 \times 16)$, go to step6;
- **step4**: check 16x8, 8x16, 8x4, 4x8; go to step7;
- **step5**: check 16x8, 8x16; go to step7;
- **step6**: check sub-macroblock partition; go to step7;
- **step7**: select the best inter mode. if the energy of the residue for best inter mode is $> T_2$, check intra modes; goto step8;
- **step8**: choose the best mode among all tested modes.

In step1, threshold $T_1 = Nbits \cdot \lambda_{mode}$, where $Nbits$ equals the minimum number of bits required for non SKIP inter modes. In step2, by comparing SKIP with 16x16 and 8x8, we assume that if the RD cost for SKIP is the minimal, then the probability for other modes to have cost less than SKIP will be very small, so no other modes need to be checked. We check the monotonic condition in step3, where
Min J_{mode} = \min(J_{mode}(16 \times 16), J_{mode}(8 \times 8), J_{mode}(4 \times 4)) \) and \( \max J_{mode} = \max(J_{mode}(16 \times 16), J_{mode}(8 \times 8), J_{mode}(4 \times 4)) \). In step 6, an additional decision is performed for each 8x8 partition to decide which type shall be used among the 4 sub-macroblock partitions. Only 8x4 and 4x8 need to be tested. The initial result of 8x8 and 4x4 can be reused. In step 7, we assume that inter modes always have higher priority than intra modes for inter images. So if the inter modes have optimal performance, no intra modes need to be checked. The inter mode that has minimal cost is selected as the best inter mode. In our algorithm, the performance of inter modes is measured by the energy of the residue, which is simply computed as the sum of the absolute value of the block transformed coefficients \( (T_2 = 0 \) is used in our simulation). In step 8, the mode that has minimal cost is selected as the best mode.

When performing ME in step 4, 5 and 6, not all reference frames need to be checked. The empirical statistics show that 8x4 and 4x8 modes only need to be checked within the best reference picture of 8x8 and 4x4, while 16x8 and 8x16 modes only need to be checked within that of 8x8 and 16x16.

For intra modes, intra4x4 needs to be checked on 9 modes and intra16x16 needs to be checked on 4 modes. Further simplification on intra mode decision is desired. In our algorithm, the energy of the residue for the best inter mode is further compared with thresholds \( T_3 \) and \( T_4 \). If it is below \( T_3 \), only the DC mode for intra4x4 and intra16x16 is checked; else if it is below \( T_4 \), only the 3 most probable modes (vertical, horizontal and DC mode) for intra4x4 and intra16x16 are checked; otherwise, all modes are checked. We select \( T_3 = 5 \) and \( T_4 = 8 \) in our simulations. The threshold can be selected adaptively based on spatial and/or temporal neighboring conditions.

The mode decision can be further simplified by taking the neighborhood mode into consideration. The relationship of motion vectors among different block type partitions can also be used to simplify our algorithm. A mixed inter-intra mode decision is also worth trying.

3. FAST MOTION ESTIMATION

With our fast mode decision scheme proposed in Section 2, even though ME does not need to be performed for some block types and reference pictures, it still involves a lot of computations at the encoder. In this section, we propose a fast ME algorithm to reduce the complexity. The ME process involves two steps: a fast integer pel motion search and a fast subpel motion refinement. We shall mainly discuss the subpel ME. For completeness, the fast integer motion search algorithm will be briefly introduced. The integer motion search is based on a well-known predictive motion estimation algorithm: the Enhanced Predictive Zonal Search (EPZS) [3]. It has three steps: (1) ME around initial predictors. The predictors are selected based on spatial-temporal correlation, different block type partitioning and/or a fixed pattern. (2) ME refinement around the best predictor using diamond or square patterns according to reference picture, block type and complexity of the sequence. (3) If necessary, another refinement is performed around the second best predictor or the median predictor. At each step, an adaptive threshold is used to decide if the algorithm will stop. The above algorithm can avoid being trapped into local minimum which is often the case of other fast motion search algorithms by employing multiple predictors and dual pattern refinement and further improve ME speed by using early stopping criterion.

As we discussed earlier, 16 checking positions are required for full subpel ME, which takes a trivial part in the total computation load when full search is used for integer ME. But when fast integer ME is used, such as EPZS, which can decrease the checking points down to the order of ten, the computational load for subpel ME can be no longer ignored. This process can also have a considerable impact to memory usage and access. In this paper, we propose two strategies to improve the subpel MV refinement: (1) subpel fast search pattern; (2) early stop criterion. The issue of subpel interpolation is also studied.

Two patterns will be proposed to reduce the searching points for subpel MV search. The basic idea behind them is that by assuming that the error surface is monotonic, MV refinement is only required for the positions around the points which have minimal errors. The two patterns will be illustrated using Figure 1. For ease of explanation, we call E as center position, B, D, F, H as diamond positions, and A, C, I, G as corner positions.

The first pattern is under the assumption that the diamond positions are generally more important than corner positions for motion search. For half pel search, we first examine the four diamond half pel positions. If the best position is the center point, no more search is required, otherwise, the two adjacent corner points are also examined. For example, assuming E is the best integer point, we first check half pel 2.5.7.4. If E is the best, the search is terminated; otherwise, if 2 is the best, we will continue by checking 2’s adjacent corner positions, i.e., 1 and 3. The same search strategy is used for quarter-pel. In this pattern, the number of worst case checking points is 6+6=12 points.

As for the second pattern, we need to store the position for the best integer and its best diamond position. For half pel search, 3 half pels between these two integer positions will be tested. For example, in Figure 1, if best integer position is E and its best diamond position is H, then half pel 6,7 and 8 will be tested. For quarter pel search, the best position and second best position in half pel search are chosen. We need to consider 3 cases.

Case 1: if the two best pels are integer-pel, two quarter pels on the same row/column will be tested. For example, if two best pels are E and H, b and g will be tested.

Case 2: if not case 1, if two best pels are at the same row/column, 3 quarter pels between these two pels will be tested. For example, if two best pels are 7 and 8, c, e and h will be tested.
Case 3: if the two best pels are in a diagonal position, for example, E and 8, then 3 quarter-pel (two diamond positions and one corner position) around best pel will be searched. For example, if best position is E, b, c and i will be tested; if best position is 8, e, c, and j will be tested.

The number of worst case checking points is 3+3=6 points.

In our algorithm, the above two patterns are adaptively selected by comparing the RD cost $J_{motion}$ of the integer ME with the spatial adjacent (left, top, top-left, top-right) blocks. If the current search is for reference frame 0 or the ratio of $J_{motion}$ for integer search of the current block over the minimal cost of its neighbors is larger than a threshold, the first pattern is used, otherwise the second pattern is adopted.

During motion refinement, in many cases, we have observed that the half-pel or quarter-pel motion refinement is not necessary. Thus, we apply an early stopping criterion to eliminate unnecessary refinement. In general, half pel refinement is more important than quarter pel, so the rule for half pel is more restrictive than quarter pel. The procedure is described as follows.

- step1: after integer search, examine the current minimum SAD, if it is below a threshold $T_1$, stop; otherwise, go to step2;
- step2: examine half pel positions, examine the current minimum SAD, if it is below a threshold $T_2$, stop; otherwise, go to step3;
- examine the quarter pel positions.

As for the threshold, currently $T_1$ is set equal to the number of pixels of the examined block type. The threshold $T_2$ is defined as

$$T_2 = a \times \min(MinJ_1, MinJ_2, ..., MinJ_n) + b,$$

where a and b can be fixed values ($a = 1.1, b = T_1/4$) and $MinJ_1, MinJ_2, ..., MinJ_n$ correspond to minimum distortion values of previously examined blocks and an upper bound($= 3T_1$). In our case, we use the 4 spatial adjacent blocks (left, top, top-left and top-right). In addition, the threshold is further applied to subpel refinement with regards to reference picture. In particular, if the reference frame is not 0 and its distortion is 2 times larger than the minimum distortion given by another reference frame, then there is a high probability that this reference frame is not the best, so the subpel refinement can be completely skipped.

In JVT, MV accuracy can be 1/4 pel. In the encoder, same sub-pel values will be used many times, mainly due to various block types and multiple reference frames. A study between the subpel interpolation complexity and memory requirement is needed. We investigated three solutions. The first solution is to precompute the quarter-pel and store it in the buffer for reuse as is done in the reference software [1]. This solution decreases the complexity of recomputing the same subpel value, but increases memory usage to a large extent: 16 times the memory of the original non-interpolated image is required. The second solution goes to another end: compute quarter-pel on the fly. In this way, no extra memory is required, but the complexity is increased, especially for multi-reference pictures. The third solution is a trade-off of the first and second solution: pre-compute half-pel and store it, compute quarter-pel on the fly. This requires only 4 times the memory, while simulation results show that combined with our fast subpel algorithm, the complexity is comparable to the first solution.

4. SIMULATION RESULTS

Our proposed scheme was integrated within version 4.2a of the reference JVT software [1]. Even though we have examined several different resolution sequences, we have selected to only present five relatively difficult sequences in this paper, thus emphasizing on the stability/performance of our schemes. These are CIF sequences Foreman, Stefan, Bus, Flowergarden and Mobile. In the simulation, we encoded the sequences at 30fps. The CAVLC entropy coder was used for all our tests, with quantizer values of 28, 32, 36, and 40, a search range of 32, and 5 references. To simplify our comparison, we have used average PSNR gain ($\Delta$PSNR) and bitrate reduction ($\Delta$bitrate) results, based on the above quantizers, as is recommended by [5]. The complete results are shown in Table 1, where the speed up compared to the reference software is also shown. From these results we observe that our proposed scheme can greatly simplify the encoder complexity. 85% to 90% complexity reduction can be achieved versus the reference software, while coding efficiency is only slightly decreased.

<table>
<thead>
<tr>
<th>sequence</th>
<th>foreman</th>
<th>stefan</th>
<th>bus</th>
<th>flower</th>
<th>mobile</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta$bitrate(%)</td>
<td>2.83</td>
<td>2.44</td>
<td>1.31</td>
<td>1.72</td>
<td>0.78</td>
</tr>
<tr>
<td>$\Delta$PSNR(dB)</td>
<td>-0.13</td>
<td>-0.12</td>
<td>-0.06</td>
<td>-0.09</td>
<td>-0.04</td>
</tr>
<tr>
<td>SpeedUp</td>
<td>9.91</td>
<td>8.94</td>
<td>8.52</td>
<td>9.51</td>
<td>8.07</td>
</tr>
</tbody>
</table>

5. CONCLUSION

In this paper, we present a new algorithm to jointly optimize motion estimation and mode decision in JVT. Simulation results demonstrate that our scheme can maintain coding efficiency at considerably lower complexity.

6. REFERENCES