TECHNICAL REPORT

ISO/IEC TR 14496-7

Second edition
2004-10-15

Information technology — Coding of audio-visual objects —

Part 7: Optimized reference software for coding of audio-visual objects

Technologies de l'information — Codage des objets audiovisuels —

Partie 7: Logiciel de référence optimisé pour le codage des objets audiovisuels

Reference number
ISO/IEC TR 14496-7:2004(E)

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

In exceptional circumstances, the joint technical committee may propose the publication of a Technical Report of one of the following types:

— type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;

— type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;

— type 3, when the joint technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example).

Technical Reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical Reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC TR 14496-7, which is a Technical Report of type 3, was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

This second edition cancels and replaces the first edition (ISO/IEC 14496-7:2002) which has been technically revised.

ISO/IEC TR 14496 consists of the following parts, under the general title *Information technology — Coding of audio-visual objects*:

— *Part 1: Systems*

— *Part 2: Visual*

— *Part 3: Audio*

— *Part 4: Conformance testing*

— *Part 5: Reference software*

— *Part 6: Delivery Multimedia Integration Framework (DMIF)*

— *Part 7: Optimized reference software for coding of audio-visual objects [Technical Report]*

— *Part 8: Carriage of ISO/IEC 14496 contents over IP networks*

— *Part 9: Reference hardware description [Technical Report]*

— *Part 10: Advanced Video Coding*

— *Part 11: Scene description and application engine*

— *Part 12: ISO base media file format*

— *Part 13: Intellectual Property Management and Protection (IPMP) extensions*

— *Part 14: MP4 file format*

— *Part 15: Advanced Video Coding (AVC) file format*

— *Part 16: Animation Framework eXtension (AFX)*

— *Part 17: Streaming text format*

— *Part 18: Font compression and streaming*

— *Part 19: Synthesized texture stream*

Introduction

Purpose

This part of ISO/IEC 14496 was developed in response to the growing need for optimized reference software that provides both improved visual quality and faster execution while compliance is preserved. The goal is to provide non-normative tools that are essential for implementations of the normative parts of the ISO/IEC 14496 specifications. For example, Part 5 of the ISO/IEC 14496 specifications uses a full search motion estimation which is theoretical optimum in coding efficiency but impractical for commercial implementation. In the past, the industry needs to create its own encoding tools for its target products. In this part, we provide a well-tested set of encoding tools that can enhance the performance but should not be standardized. The following recommended tools would be up to the individual organization to decide if it wishes to adopt or adapt these tools for its specific needs. This part provides significant reduction in the time-to-market and provides a reference benchmark for commercial ISO/IEC 14496 compliant products.

Information technology — Coding of audio-visual objects —

Part 7: Optimized reference software for coding of audio-visual objects

1 Scope

This part of ISO/IEC 14496 specifies the encoding tools that enhance both the execution and quality for the coding of visual objects as defined in ISO/IEC 14496-2. The tool set is not limited to visual objects but at this point all the recommended tools are visual encoding tools. There are four tools that have been described in this technical report.

- Fast Motion Estimation
- Fast Global Motion Estimation
- Fast and Robust Sprite Generation
- Fast Variable Length Decoder Using Hierarchical Table Lookup

These tools have been demonstrated as robust tools with source codes for both MoMusys and Microsoft implementations. In the current implementations, there is single software that includes all tools existed in the ISO/IEC 14496-2. This is obviously inefficient in terms of code size and execution speed. To address this issue, the optimized reference software has compilation switches such that only selected tools as defined by the profiles and levels are included. Such level of optimization is performed at high level programming language. The platform specific optimization is currently not addressed by this part.

2 Fast Motion Estimation

2.1 Introduction to Motion Adaptive Fast Motion Estimation

The optimization of fast motion estimation is essentially a multi-dimensional problem. The key dimensions concerned in this problem are: Rate, Quality (PSNR), Speed-up (or Computational Gain), Algorithmic Complexity, Memory Size and Memory Bandwidth (see Figure 1). There always exists a trade-off among all these five key dimensions. Therefore, it is highly desirable to have an adaptive fast motion estimation core algorithm with scalable structure, which can be adaptively optimized with respect to all or selected aspects for various coding environment and requirements. Since the rate control is used to fix the bit-rate, the optimization problem is reduced by one dimension to four dimensions.

Motion Vector Field Adaptive Search Technique (MVFAST) [1] is a generic algorithm of the family of *motion-adaptive* fast search techniques, originally proposed by Kai-Kuang Ma and Prabhudev Irappa Hosur from Nanyang Technological University (NTU), Singapore. The MVFAST offers high performance both in quality and speed and does not require memory to store the searched points and motion vectors. The MVFAST has been adopted by MPEG-4 Part 7 in the Noordwijkerhout MPEG meeting (March 2000) as the *core technology* for fast motion estimation.

A derivative of MVFAST, called *Predictive* MVFAST (PMVFAST) [2], is considered as an *optional approach* that might benefit in special coding situations. PMVFAST incorporates a set of thresholds into MVFAST to trade higher speed-up at the cost of memory size, memory bandwidth and additional algorithmic complexity. In PMVFAST, the threshold values are adjusted based on the 54 test cases specified by MPEG-4. However, the coding performance and sensitivity of PMVFAST using these thresholds for the video sequences and encoding conditions outside the MPEG-4 test set has *not* been studied and verified.

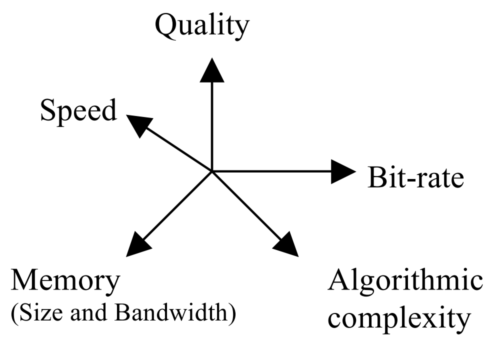


Figure 1 — Five dimensional optimization problem of fast motion estimation

2.2 Technical Description of Core Technology — MVFAST

2.2.1 Detection of stationary blocks

A large number of MBs in the video sequences (e.g., “talking head” video sequences) with low-motion content tend to have motion vectors equal to (0,0). Such MBs in the regions of no-motion activity can be detected simply based on the sum of absolute difference (SAD) at the origin. Therefore, we exploit an optional phase, called *early elimination of search*, as the first step in MVFAST as follows. The search for a MB will be terminated immediately, if its SAD value obtained at (0,0) is less than a threshold T, and the motion vector is assigned as (0,0). Through extensive simulations, we found that among those zero-motion blocks identified, about 98% of them have their SAD at position (0,0) less than 512. Hence, we choose T = 512 to enable the mechanism of early elimination of search. Since this early elimination of search phase is optional, it can be turned off or disabled by imposing $T = 0$.

2.2.2 Determination of local motion activity

The *local motion vector field* at a macroblock (MB) position is defined as the set of motion vectors in the *region of support* (ROS) of that MB. The ROS of a MB includes the n neighborhood MBs. In MVFAST, the ROS with n = 3 is shown in Figure 2. Let $V=\{V_0, V_1, \ldots V_n\}$, where $V_0 = (0,0)$, and V_i (and $i \neq 0$) is the motion vector of MB$_i$ in the ROS (see Figure 2). The cityblock length of $V_i=(x_i, y_i)$ is defined as $l_{vi} = |x_i| + |y_i|$. Let $L = MAX\{l_{vi}\}$ for all V_i. The motion activity at the current MB position is defined as follows.

$$\begin{aligned}\text{Motion Activity} &= \text{Low, if } L \le L1;\\ &= \text{Medium, if } L1 < L \le L2;\\ &= \text{High, if } L > L2;\end{aligned} \qquad (1)$$

where L_1 and L_2 are integer constants. We choose L_1 and L_2 as the cityblock distance from the center point of the pattern to any other point on the small and large search patterns (see Figure 3), respectively. Thus, L_1 =1 and L_2 =2.

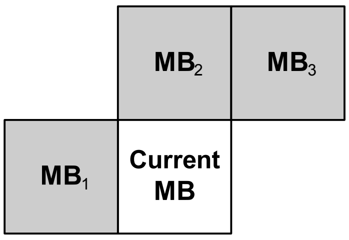


Figure 2 — Region of support (ROS) for the current MB consists of MB1, MB2 and MB3

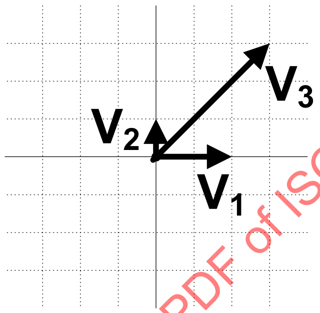


Figure 3 — Example of distribution of motion vectors belonging to set V. In this case, lv1 = 2, lv2 = 1, lv3 = 6; thus L = MAX{lv1, lv2, lv3} = 6

2.2.3 Search Center

The choice of the search center depends on the local motion activity at the current MB position. If the motion activity is low or medium, the search center is the origin. Otherwise, the vector belonging to set V that yields the minimum sum of absolute difference (SAD) is chosen as the search center.

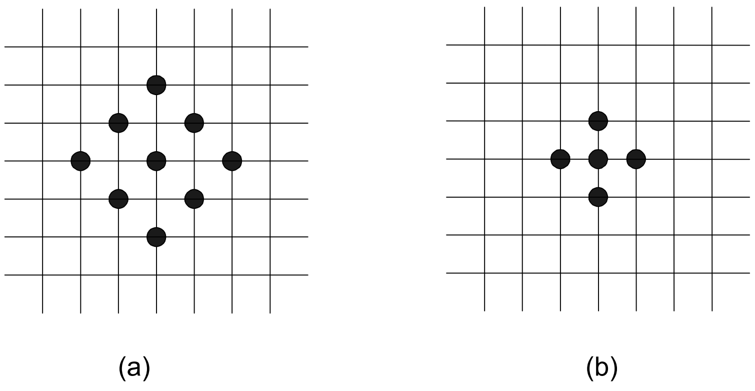


Figure 4 — (a) Large Diamond Search Pattern (LDSP) and (b) Small Diamond Search Pattern (SDSP)

2.2.4 Search Strategy

A local search is performed around the search center to obtain the motion vector for the current MB. The search patterns employed for the local search are shown in Fig. 4. Two strategies are proposed for the local search and their choice depends on the motion activity identified. If the motion activity is low or high, we employ small diamond search (SDS). Otherwise, we choose large diamond search (LDS).

i) *Small Diamond Search* (SDS)

Step 1: Small diamond search pattern (SDSP) is centered at the search center, and all the checking points of SDSP are tested. If the center position yields the minimum SAD (i.e., no motion), then the center represents the motion vector; otherwise, go to Step 2.

Step 2: The center of SDSP moves to the point where the minimum SAD was obtained in the previous step, and all the points on SDSP are tested. If the center position yields the minimum SAD, then the center represents the motion vector; otherwise, recursively repeat this step.

ii) *Large Diamond Search* (LDS)

Step 1: Large diamond search pattern (LDSP) is centered at the search center, and all the checking points of LDSP are tested. If the center position gives the minimum SAD, go to Step 3; otherwise, go to Step 2.

Step 2: The center of LDSP moves to the point where the minimum SAD was obtained in the previous step, and all the points on LDSP are tested. If the center position gives the minimum SAD, go to Step 3; otherwise, recursively repeat this step.

Step 3: Switch the search pattern from LDSP to SDSP. The point that yields the minimum SAD, is the final solution of the motion vector.

Table 1 summarizes the methodology for selection of search center and search strategy depending on the motion activity at the current MB position.

Table 1 — The search modes for MVFAST

Motion Activity	Search Center	Search Strategy
Low	Origin	SDS
Medium	Origin	LDS
High	The position of the vector in set *V* that yields minimum SAD	SDS

2.2.5 Perspectives on implementing MVFAST

The MVFAST algorithm can be structured in terms of *profiles*. The MVFAST itself as described above can be viewed as the **main profile**. The low, medium and high motion activity cases in Table 1 can be considered individually as three other different profiles of MVFAST. Depending on the video coding applications, any one of these individual profiles can be turned "ON" simply by adjusting the two parameters, L_1 and L_2, in Equation (1). If we set $L_1 = L_2 =$ Search Range, we obtain "low motion activity" profile. The "medium motion activity" profile (which is the same as Diamond Search, as described in VM Version 14) can be obtained, if we set $L_1 = -1$ and $L_2 =$ Search Range. For "high motion activity" profile, we can set $L_1 = L_2 = -1$. Note that in this case, Search Range = 2*N, if the search in either coordinate is in the range [–N, N-1].

Although MVFAST is implemented in an intelligent way such that the overlap of search points is minimized when the search pattern moves, few search points are visited more than once. This overlap can be avoided by keeping the record of all the search points visited and testing if the current search point is visited earlier. Thus further improvement over speed-up can be achieved.

The search point (0,0) is always tested in MVFAST. However, some improvement in computational gain is obtained by testing (0,0) point only, if any of the motion vectors in the ROS has motion vector = (0,0).

Through extensive experiments using MVFAST, it is found that further improvement in objective quality can be achieved when interlaced CCIR sequences with high global motion are coded in progressive mode, by including the motion vector of collocated block on the previously coded non-intra frame in the set *V*. During the motion estimation of interlaced pictures, each frame prediction of macroblock motion is performed before field motion estimation. Therefore, for field motion estimation of current macroblock, its frame motion vector is included in set *V*.

From hardware implementation viewpoint, to restrict the total number of search points for a block in the worst case to be *N*, an additional stopping criterion — "stop the search when the number of search points visited so far is equal to *N*", can be included in SDS and LDS given in subclause 2.4.

2.2.6 Special Acknowledgements

Kai-Kuang Ma and Prabhudev Irappa Hosur would like to sincerely acknowledge tremendous support from Professor Meng Hwa Er, Dean, School of Electrical and Electronic Engineering, and Deputy President of Nanyang Technological University, Singapore, who plays a vital role on promoting and directing all Singapore MPEG activities. For independent verification efforts, the following individuals are greatly acknowledged: Dr. Weisi Lin, Mr. Chengyu Xiong, Dr. Ee Ping Ong, all from Institute of Microelectronics (IME), Singapore.

CONTACT PERSON:

Dr. Kai-Kuang Ma, School of Electrical and Electronic Engineering, Nanyang Technological University, Block S2, Nanyang Avenue, Singapore 639798. Tel: +65-790-6366; Fax: +65-792-0415; Emails: ekkma@ntu.edu.sg and kaikuang@hotmail.com.

2.3 Technical Description of PMVFAST

2.3.1 Introduction

This section provides the technical description of the *Predictive Motion Vector Field Adaptive Search Technique* (PMVFAST) which adds some techniques from the *Advance Predictive Diamond Zonal Search* (APDZS) [2] proposed by the Hong Kong University of Science and Technology (HKUST) to the MVFAST core mentioned above to achieve larger speed up. The PMVFAST was contributed by Prof. Ming L. Liou, Dr. Oscar C. Au, and Alexis Tourapis of HKUST. PMVFAST is faster than MVFAST at the expense of higher hardware complexity

Several independent parties, Optivision Inc., Sarnoff Co., Mitsubishi Electric Information Technology Center America, National Technical University of Athens (NTUA), and Beijing University of Aeronautics and Astronautics (BUAA), conducted evaluation throughout the entire adoption process. For independent verification efforts, the following individuals are greatly acknowledged: Dr. Weiping Li (from Optivision), Dr. Hung-Ju Lee and Dr. Tihao Chiang (from Sarnoff), Mr. Anthony Vetro and Dr. Huifan Sun (from Mitsubishi), Mr. Gabriel Tsechpenakis, Mr. Yannis Avtithis and Prof. Stefanos Kollias(from NTUA), and Prof. Bo Li, Yaming Tu (from BUAA).

2.3.2 Technical Description of PMVFAST

PMVFAST combines the 'stop when good enough' spirit, the thresholding stopping criteria and the spatial and temporal motion vector prediction of APDZS and the efficient large and small diamond search patterns of MVFAST. Let the *refBlock* be the block in the reference frame at the same spatial location as the current block. Without loss of generality, the distortion criterion is assumed to be the Sum-of-Absolute-Difference (SAD), though it can be other measures. The predicted motion vector in PMVFAST is the median of the motion vectors of three blocks spatially adjacent to the current block (left, top and top right), as in MPEG motion vector predictive coding.

Firstly, the PMVFAST computes the SAD of the predicted motion vector (PMV), and stops if any one of two stopping criteria is satisfied. The first criterion is that the PMV is equal to the motion vector of refBlock and the SAD of PMV is less than that of refBlock. The second criterion is that the SAD of PMV is less than a threshold.

Secondly, the PMVFAST computes the SAD of some highly-probable motion vectors (MV of left, top and top right spatially neighboring blocks, MV of (0,0) and MV of refBlock) and stops if any one of two stopping criteria is satisfied. The first criterion is that the best motion vector so far is equal to the MV of refBlock and the minimum SAD so far (MinSAD) is less than that of refBlock. The second criterion is that the MinSAD is less than a threshold.

Thirdly, the PMVFAST selects the MV associated with minSAD and performs a local search using techniques of MVFAST. If PMV is equal to (0,0) and the motion vectors of the three spatially adjacent blocks are identical with large associated SAD, the large diamond search of MVFAST is applied. Otherwise, if the motion vectors of the three spatially adjacent blocks are identical and are the same as the MV of refBlock, small diamond search is applied with the simplication that only one small diamond pattern is examined. Otherwise, the small diamond search of MVFAST is applied.

Here is the step-by-step algorithm of PMVFAST: The variables *thresa*, *thresb* are integers used as thresholds in the stopping criteria.

(Initialization)

Step 1: Set thresholding parameters (*thresa* & *thresb*). These are set as follows:
If first row and column, *thresa* = 512, *thresb* = 1024
Else *thresa* = minimum value of the sad of left, top and top-right blocks. *thresb* = *thresa* + 256;
If *thresa<512, thresa = 512.* If *thresa* > 1024, *thresa* = 1024.
If *thresb* > 1792, *thresb* = 1792.
Set *Found*=0 and *PredEq*=0
Compute the predicted MV according to the Median rule.
Select previous MV, above, and above-right and calculate median.
If block is an edge block, depending to the position, do the following:
If block is on the first column, assume previous MV to be equal to (0,0).
If block is on the first row, select previous MV as the prediction.
If block is on the last column, assume above right MV to be equal to (0,0).
If left MV = top MV = top-right MV then set *PredEq*=1;

(Initial prediction calculation)

Step 2: Calculate *Distance*= $|\text{MedianMV}_X| + |\text{MedianMV}_Y|$ where MedianMV is the motion vector of the median.
If *PredEq*=1 and $\text{MV}_{\text{predicted}}$ = Previous Frame MV, set Found=2
Step 3: If *Distance*>0 or *thresb*<1536 or *PredEq*=1.
Select small Diamond Search. Otherwise select large Diamond Search.
Step 4: Calculate SAD around the Median prediction. MinSAD=SAD
If Motion Vector equal to Previous frame motion vector and MinSAD<PrevFrmSAD goto Step 10.
If SAD<=256 goto Step 10.
Step 5: Calculate SAD for motion vectors taken from left block, top, top-right, and Previous frame block. Also calculate (0,0) but do not subtract offset.
Let MinSAD be the smallest SAD up to this point. If MV is (0,0) subtract offset.

Step 6: If MinSAD <= *thresa* goto Step 10.
If Motion Vector equal to Previous frame motion vector and MinSAD<PrevFrmSAD goto Step 10.

(Diamond Search)

Step 7: Perform Diamond search, with either the small or large diamond. If *Found*=2 only examine one Diamond pattern, and afterwards goto step 10

Step 8: If small diamond, iterate small diamond search pattern until motion vector lies in the center of the diamond. If center then goto step 10.

Step 9: If large diamond, iterate large diamond search pattern until motion vector lies in the center. Refine by using small diamond and goto step 10.

(Final step. Use best MV found.)

Step 10: The motion vector is chosen according to the block corresponding to MinSAD.
By performing an optional local half-pixel search, we can refine this result even further.

2.3.3 Special Acknowledgement

M.L. Liou, O.C. Au, A.M. Tourapis would like to sincerely acknowledge the tremendous support from Prof. Ishfaq Ahmad, the Department of Electrical and Electronic Engineering of the Hong Kong University of Science and Technology (HKUST), the Hong Kong Telecom Institute of Information Technology (HKTIIT), the Hong Kong Reseach Grants Council (RGC).

CONTACT PERSON: Dr. Oscar C. Au, Department of Electrical and Electronic Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, China. Tel: +852 2358-7053; Fax: +852 2358-1485; Emails: eeau@ust.hk.

2.4 Conclusions

The comparison of MVFAST vis-à-vis PMVFAST is given in Table 2.

Table 2 — Comparison of MVFAST and its derivative algorithm PMVFAST

	MVFAST	PMVFAST
Threshold comparisons	No threshold or one optional threshold	A set of compulsory thresholds. Coding performance and sensitivity of PMVFAST using these thresholds for the video sequences and encoding conditions outside the MPEG-4 test set has *not* been studied and verified.
Algorithmic complexity	Less complex	Higher complexity
Memory size	No need to store either search points or motion vectors	Memory is compulsory. Up to 4 Mbytes of memory for a search window size of +- 1024 to store search points and up to 1.3 kilobytes for storing motion vectors
Memory bandwidth	Not applicable (since no memory is needed)	Memory bandwidth is wasted for accessing the memory when each search point is visited
Objective quality (PSNR)	On average, about 0.2 dB less than Full Search	On average, about 0.1 dB less than Full Search
Speed-up		About 50% faster than MVFAST
Scalability	Scalable to three different profiles; where each profile is obtained by simply assigning values to two parameters in the beginning of the algorithm.	Not scalable

The MVFAST is recommended as the core technology for fast motion estimation, since it is a generic solution suitable for all encoding environments. However, if issues such as memory, algorithmic complexity and threshold sensitivity are not of concern, then PMVFAST algorithm can be used. Therefore, MVFAST is integrated as the core mode in Part 7.

3 Fast Global Motion Estimation

3.1 Introduction to Feature-based Fast and Robust Global Motion Estimation Technique

Sprite coding is an important technology in MPEG-4 encoder, but sprite coding could be hardly applied in real-time application because the global motion estimation (GME) used in sprite coding is a time-consuming task. In order to overcome this problem a feature-based fast and robust GME technique (FFRGMET) is proposed by Tsinghua University, which improves the original GME method [3]. Comparison experimental results show that FFRGMET improves the speed substantially. There are three significant improvements of FFRGMET compared with original GME method.

(1) More Accurate Outlier Detection on the Base Level

Three-level pyramid is applied in the GME calculation. Local motion will affect GME when there is local motion. Pixels undergoing local motion are the outliers in GME. Residual block based outlier detection method is used in the base level of GME in FFRGMET. The original outlier detection method in VM is residual histogram based, which could not represent the spatial distribution of the residuals. At the base level of the estimation, the pixels belonging to the foreground object appear to show large residuals and concentrate together to a compact region, so the residual block based outlier detection could help to locate the outliers more accurately.

(2) More Robust Object Function for Parameter Estimation

The robust object function is used to replace the quadratic object function in VM. This object function is adaptive to the variance of all pixels to be estimated, and is more robust to outliers than the quadratic form. Robust object function is very important in FFRGMET because there are fewer pixels in calculation compared with GME method in VM. Robust object function restrains the effect of outliers in GME. Robust object function considers the residual and variance of the pixels set in the estimation.

(3) Fewer Pixels Used in GEM Calculation

On the intermediate level and base level of pyramid, it does not need to use all the pixels in the object region to participate the GME calculation. There is much redundant information in whole background. So in FFRGMET, only some feature pixels are selected as representatives in GEM. The feature pixels are selected based on the spatial edge and the temporal difference, which could contribute more to the motion estimation than other pixels. The speed of GME is accelerated because fewer pixels are used.

3.2 Technical Description of FFRGMET

3.2.1 Outlier Exclusion

Residual-block based method to exclude the outliers in FFRGMET calculation. The image is divided into 16×16 sized blocks. The block is regarded as a potential block to be rejected if the SAD of this block belongs to the top 30% ordered by the SAD of a block.

$$SAD = \sum_{i=0}^{16}\sum_{j=0}^{16}\left|\gamma_{ij}\right|;\quad \gamma_{ij} \text{ is the residual of pixel (i,j).}$$

There are two steps to determine whether to reject the blocks in GME calculation or not. (a) Firstly if there is at least four potential blocks to be rejected in the eight nearest neighbor blocks of current potential block to be rejected, then the current potential block to be rejected will be rejected, otherwise the current potential block to be rejected is reserved. (b) Secondly if there is a rejected block in the eight nearest neighbor of remainder potential block to be rejected, then this potential block to be rejected will be rejected in the calculation of global motion estimation.

3.2.2 Robust Object Function

The following robust object function is used in the Levenberg-Marquadet calculation of FFRGMET:

$$F = \sum_i \frac{r_i^2}{(\sigma^2 + r_i^2)},\quad \sigma = 1.253\ \mathrm{E}(r)$$

E(r) is the mean of absolute value of residual r. The weight function for the object function is:

$$\text{weight}(r) = 1/(\sigma^2 + r^2)$$

3.2.3 Feature Selection

Feature pixels are selected in GME calculation according to the following conditions (Condition 7-1).

$$\{(x, y) \mid (UPTHRESH \cdot E(|I_x|+|I_y|)) > (|I_x|+|I_y|) > (DOWNTHRESH \cdot E(|I_x|+|I_y|)) \quad \text{AND} \quad (|I_t| > THRESHOLD_t \cdot E(I_t))\} \tag{7-1}$$

I_x and I_y are the spatial gradient components of luminance. I_t is the temporal gradient of luminance. *E*(x) is the mean value of the set of x. The pixel that belongs to motion edge must meet the following two conditions. The first is that its spatial gradient value must be larger than a predefined down threshold and less than a predefined up threshold. Differential operator is sensitive to noise when the gradient value is small. So large gradient value can reduce the influence of noise. And large gradient value means that there is an edge of luminance. The second condition is that the absolute value of temporal gradient value must be larger than a predefined threshold. I_t of that pixel will be small if the direction of global motion is perpendicular to the direction of luminance gradient of the point. The pixels belonging to motion edge are used in the intermediate and base levels of pyramid calculation of global motion estimation.

3.2.4 Algorithm Description

Following is the complete description of FFRGMET algorithm (Note that only Y-component data is used in the following steps). A 6-parameter affine model is used in the FFRGMET.

Step1: Set parameters:

Maximum iterative steps for GN (Gauss-Newton) and LM (Levenberg-Marquadet) calculation (MAX_TIMES=32)
Resistant factor of LM (RESISTANCE = 0.002) and amplificatory factor of LM (AMPLIFIER = 10.0)
Number of parameters to be estimated (PARNUM = 6)
Threshold of gradient value (DOWNTHRESH =1.25, UPTHRESH = 1.65)
Threshold of temporal gradient value (THRESHOLD$_t$=1.0)

Step2: Generate the three-level pyramid:

Use Gaussian down-sampling filter [1/4, 1/2, 1/4] on the original image to generate the images of the three-level pyramid.

Step3: Calculate the motion parameters of top level:

Use three-step search method to calculate the initial two translational parameters of 6-parameter affine motion model.
Exclude the top 10% of total pixels in the residual histogram.
Estimate the parameters based on the left pixels using GN method.

Step4: Project the parameters of the top level onto the intermediate level.

Step5: Calculate the motion parameters of the intermediate level:

Exclude the top 10% of total pixels using residual histogram.
Select the pixels according to condition 7-1.
Estimate the parameters of second level with the selected pixels using LM optimizing method.

Step6: Project the parameters of the intermediate level onto the base level.

Step7: Calculate the motion parameters of the base level:

```
If (VOP.Shape = Rectangle) Then
    Exclude the residual blocks with top 1/3 SAD value using the residual-block based method.
Else
    Exclude the top 10% of total pixels using residual histogram.
```

```
        End If
        Select the pixels according to condition 7-1.
        Estimate the parameters of base level with the selected pixels using LM optimizing method.
```

Note: GN means Gauss-Newton optimizing method, LM means Levenberg-Marquadet optimizing method.

3.2.5 Perspectives on implementing FFRGMET

FFRGMET is faster than the original GME method in MPEG-4 VM, but it is more complex and the PSNR of coding will have a little loss. The original GME method can be used if there is no requirement for speed, otherwise FFRGMET can be used. All those predefined parameters can be changed, which is gotten from the experimentation.

3.2.6 Special Acknowledgements

Shiqiang Yang, Yuzhuo Zhong, Wei Qi and Yuwen He would like to sincerely acknowledge tremendous support from Professor Tihao Chiang and Dr. Huifang Sun, who are charge of Encoder Optimization. For independent verification efforts, the following individuals are greatly acknowledged: Dr. Oscar C. Au, Dr. Alexis M. Tourap, from the Hong Kong University of Science and Technology, China; Dr. Feng Wu, from Microsoft Research China.

CONTACT PERSON: Prof. Shiqiang Yang, Department of Computer Science and Technology, Tsinghua University, Beijing100084, China. Tel: +8610 6278-4141; Fax: +8610 6277-1138; Email: yangshq@mail.tsinghua.edu.cn.

3.3 Conclusions

The feature-based fast and robust global motion estimation technique (FFRGMET) is about 7 times faster than the original global motion estimation (GME) algorithm in MPEG-4 verification model (VM). But the PSNR decreases about 0.06 dB on the average for luminance component. The total GMC coding speed is accelerated about 3.5 times. FFRGMET for sprite coding is much faster than then original global motion estimation method in MPEG-4 VM from the comparison at the cost of a little loss in PSNR, which is negligible. Therefore, FFRGMET is integrated in Part 7.

4 Fast and Robust Sprite Generation

4.1 Introduction to Fast and Robust Sprite Generation

Just as those techniques for fast motion estimation and fast global motion estimation, the technique for sprite generation is also very important for sprite coding. This section dedicates to describe the algorithm for fast and robust sprite generation. Firstly, the described algorithm can significantly speed up the sprite generation compared with the method provided in MPEG-4 video VM [3], meanwhile only the little extra memory is necessary. Secondly, the described algorithm can provide better subjective visual quality as well. The visual quality is another key point for static sprite coding because the background object is reconstructed by directly warping the sprite according to the definition of static sprite coding in MPEG-4 standard. Furthermore, when no auxiliary mask information is available, a rough image segmentation technique is applied to this algorithm. It can not only accelerate the motion estimation process but also improve the visual quality of generated sprite.

4.2 Algorithm Description

4.2.1 Outline of Algorithm

The described algorithm bases on that in Appendix D of MPEG-4 Video VM. However, in order to achieve fast and robust sprite generation, some novel features are introduced as shown in Figure 5. Instead of estimating the global motion of the current image directly from the previous sprite, the described algorithm first warps the previous sprite and then calculates the global motion referencing the warped sprite image. This long-term motion estimation method can greatly decrease the error accumulations caused by the individual frame. The extra cost in memory is also reasonable because the size of warped sprite is the same as that of current frame. Static sprite coding is normally used for object-based video coding, however sometimes auxiliary segmentation information is either unavailable or not accurate enough to mask out all moving objects from the scene. The rough segmentation technique is incorporated into the proposed sprite generation, which is usually used in this algorithm when no auxiliary segmentation masks are available.

The main goal of the described algorithm is to rapidly generate the background sprite with better visual quality. Assume that the video sequence comprises *n* frames, $\mathbf{I}_k$, $k = 0, 1, \ldots, n-1$. The sprite $\mathbf{S}_k$ is generated using $\mathbf{I}_i$, $i = 0, 1, \ldots, k-1$. $\mathbf{P}_k$ denotes the motion parameter estimated at the *k*th time instant. The complete sprite generation algorithm is described using pseudo C as following:

```
1.  $\mathbf{S}_0 = \mathbf{I}_0$; $\mathbf{P}_0 = 0$;
2.  For ( $k = 1$; $k < n$; $k$++ ) {
        Divide $\mathbf{I}_k$ into reliable (R), unreliable (UR), and undefined (U) regions;
        Fast and robust estimate global motion parameter $\mathbf{P}_k$ between $\mathbf{I}_k$ and $\mathbf{S}_{k-1}$;
        If no auxiliary segmentation is available, then segment $\mathbf{I}_k$;
        Warp image $\mathbf{I}_k$ towards sprite using $\mathbf{P}_k$;
        Blending the warped image with $\mathbf{S}_{k-1}$ to obtain $\mathbf{S}_k$.
    }
```

There are five modules used for processing each frame in the sprite generation, including Image Region Division, Fast and Robust Motion Estimation, Image Segmentation, Image Warping, and Image Blending. Bilinear interpolation is used for image warping, which is the same as that in MPEG-4 Video VM. Each module of the described algorithm is discussed in detail.

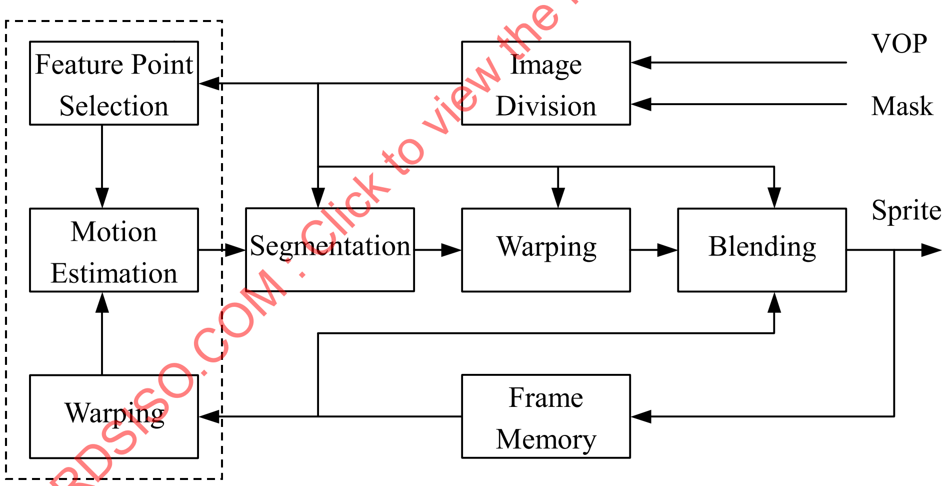


Figure 5 — Block diagram of fast and robust sprite generation

4.2.2 Image Region Division

According to the visual part of MPEG-4 standard, static sprite coding is normally used for object-based video coding. The described algorithm first derives the reliable mask from the segmentation mask by excluding some pixels along the borders of background object, as well as the frame borders. The excluded areas are defined as unreliable image region (**UR**); the rest region in background object is defined as the reliable image region (**R**). Furthermore, the areas masked as foreground objects are defined as undefined image region (**U**). The core technique of image division is to mask **UR** image region, which can be implemented by scanning the background region from four directions, i.e., left to right, top to bottom, right to left, and bottom to end. After each scanning, start points of the background region can be singled out, and then the subsequent *n* pixels are masked as **UR** image region. The sprite image is correspondingly divided into **R**, **UR**, and **U** regions as well. Reliable sprite region has been constructed from **R** image region. Unreliable sprite region was the visible part of **UR** image region. And undefined sprite region was not yet visible part of background object in previous images. An example of image division is shown in Figure 6. The image division can contribute to sprite generation on two aspects. Firstly, only **R** region participates the motion estimation, which can speed up the motion estimation processing and eliminate the effect of foreground objects and frame borders. Secondly, **R**, **UR**, and **U** regions are differently dealt with in image blending, which improves the visual quality of generated sprite.

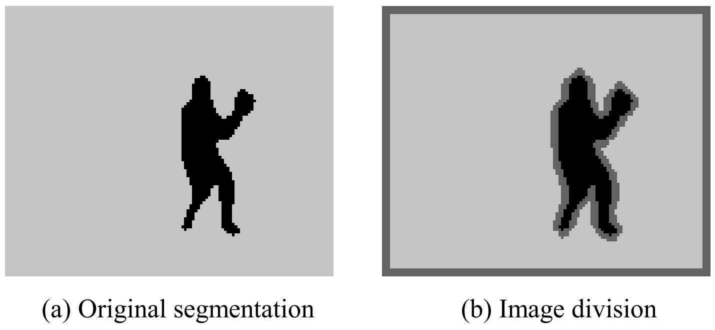


Figure 6 — Illustration of image division. Light gray: reliable region (R). Dark gray: unreliable region (UR). Black: undefined region (U)

4.2.3 Fast and Robust Motion Estimation

For the purpose of sprite generation, motion estimation aims at obtaining the motion parameters between the current image and sprite image. The sprite normally severs as the reference image in the motion estimation. In the described algorithm, instead of directly estimating the motion parameters between the current image and sprite, the pre-processing is used to accelerate motion estimation and eliminate the effect of foreground objects. Figure 7 illustrates the processing of motion estimation.

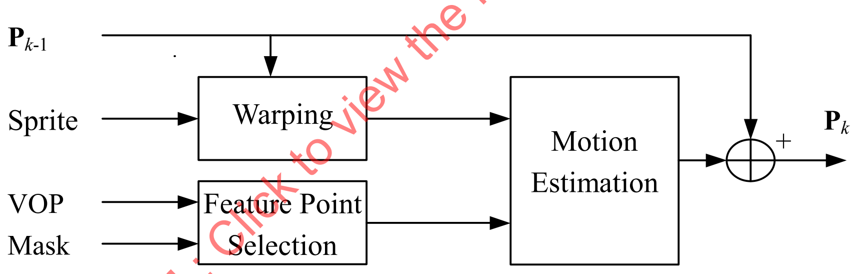


Figure 7 — Illustration of motion estimation

The pre-processing mainly focuses on the following two aspects. Firstly, feature point selection is performed in order to decrease the pixels involving in motion estimation. The selection is based on the fact that pixels with large gradient values dominantly contribute to motion estimation in the background object, rather than those located in smooth areas. The Sobel operator is first done on the current raw image, and then top 10% of total pixels located in **R** image regions are applied to motion estimation where the image magnitudes have the largest values in the Sobel image. Secondly, the sprite is warped using the previous estimated motion parameters. The great displacement between the current image and the sprite may decrease the accuracy of motion estimation, or even leads to wrong estimation. Using the warped sprite as reference, the current image can increase the robustness of global motion estimation, and even speed up motion estimation. Therefore the proposed fast and robust motion estimation consists of four steps:

Step1: Extract feature points **F** from **R** region of the current image $\mathbf{I}_k$. Sobel operator is performed, and then 10% pixels of the **R** image region are selected.

Step2: Warp previous sprite $\mathbf{S}_{k\text{-}1}$ using motion parameter $\mathbf{P}_{k\text{-}1}$ to get reference image **S'**.

Step3: Estimate motion parameter **P'** based on gradient decent method using **F** and **S'**.

Step4: $\mathbf{P}_k = \mathbf{P}_{k\text{-}1} + \mathbf{P}'$.

$\mathbf{P}_k$ is the final result of global motion estimation. The gradient decent method in step 3 is the same as that described in MPEG-4 Video VM [3].

4.2.4 Image Segmentation

Segmentation information is an essential part for sprite generation and compression. However, the auxiliary segmentation masks are sometimes unavailable. A rough image segmentation technique is developed in here for the purpose of sprite generation. Figure 8 illustrates the processing of image segmentation. The proposed algorithm can be described in detail using the following steps:

Step1: Warp previous sprite $\mathbf{S}_{k\text{-}1}$ using motion parameter $\mathbf{P}_k$ to get predicted image **S'**.

Step2: Subtract current image $\mathbf{I}_k$ from **S'** to get the absolute difference image **D**.

Step3: Filter **D** using *open* operator to detect the motion zones.

Step4: Extract the final segmentation masks by binarizing the filtered image. A threshold is dynamically selected to retain 80 % points in background object.

A simple *open* operator with the window size of 9x9 is used to detect the motion zones from the difference image. Although the segmentation is not accurate with some pixels in the background masked, it is enough only for the purpose of eliminating the effect of foreground objects in the proposed sprite generation. After image segmentation, the background areas are marked as reliable (**R**) image region, and the foreground areas are marked as unreliable (**UR**) image region. The image division results will contribute the following image blending process.

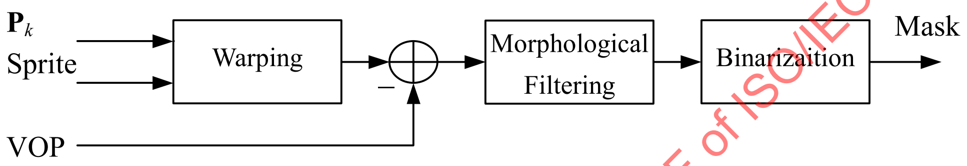


Figure 8 — Illustration of image segmentation

4.2.5 Image Blending

In the described algorithm, **R** and **UR** regions are differently dealt with in the sprite updating. The pixels located in **R** region are weighted to update the corresponding sprite pixels. However, for those pixels located in **UR** region, the corresponding sprite pixels are updated only when they have never been updated by **R** region pixels. **U** region doesn't participate in this process. The proposed algorithm can provide the sprite with better visual quality. Firstly, the **R** region ensures that reliable image information contributes more to sprite updating because **R** region normally corresponds to the background object. Secondly, the **UR** region tackles the aperture problem when no reliable image information is available. The good trade-off between **R** and **UR** region division can provide better visual quality of sprite than simply averaging images. Suppose ***p*** and ***s*** are a pair of pixels located in warped image and sprite, respectively. The weighting variable relative to ***p*** and ***s*** is denoted by *w*. The processing of ***p*** and ***s*** can be described using pseudo C as follows,

```
If  (p is reliable) {
        If (s is not reliable) {
         Set s as reliable;
         w = 0;
        }
        s = (w * s + p) / (w + 1);
        w = w + 1;
}
Else if (p is unreliable) {
    If (s is not reliable) {
                s = (w * s + p) / (w + 1);
                w = w + 1;
                If (s is undefined)
            Set s as unreliable;

        }
}
```

4.3 Conclusions

The described algorithm for fast and robust sprite generation in this section can speed up the sprite generation up to 7 times compared with the method in MPEG-4 Video VM with slight improvement in coding efficiency. Moreover, the described algorithm can significantly improve the visual quality of sprite image and reconstructed background object.

CONTACT PERSON:

Prof. Wen Gao, Institute of Computing Technology, Chinese Academy of Science, Beijing 100080, China. Tel: +8610 82649208; Fax: +8610 82649298;

Prof Hanqing Lu, Institute of Computing Technology, Chinese Academy of Science, Beijing 100080, China, Email:hqlu@ict.ac.cn.

Mr. Yan Lu, Department of Computer Science, Harbin Institute of Technology, Harbin 150001, China, Email: ylu@ieee.org.

5 Optimised Reference Software For Simple Profile and Error Resilience Tools

5.1 Scope

This subclause specifies an optimised reference encoder and decoder for visual simple profile. The improvements include integration and optimization of program structure, addition of fast algorithms, and reduction of I/O access times. The added fast algorithms consist of fast motion estimation, integer forward discrete cosine transform (DCT), inverse integer discrete cosine transform (IDCT), fast variable length decoding, etc. In this Technical Report, there is also a second set to reference decoder that implements full error resilience capabilities. In the following text, we will describe in detail the optimization procedure of the work.

5.2 Integration and Optimization of the Reference Software

5.2.1 Introduction

This work is to optimise and integrate the reference software for the MPEG-4 14496-2 Simple Profile encoder and decoder. The encoder and decoder of the video reference software are as described in ISO/IEC 14496-5.

For both the encoder and decoder, the common optimization approaches can be divided into the following three categories.

(1) Remove the unused procedures, parameters, and data structures from the code bases for the reduction of code size.

For example, some of the code bases, which provide identical functionalities, can be merged and some of the allocated buffers are not used for the Simple Profile decoder. Consequently, we can just retain only necessary modules and code bases of the Simple Profile by the MPEG-4 visual specification.

(2) Rewrite the code bases for saving the execution time and code sizes.

For example, we remove the unnecessary data movement and conditional jumps at the reference code base. Additionally, we avoid using the arithmetic operations including division (/) and modulation (%) in the code bases.

(3) Use the existing fast algorithms for the computational expensive modules.

Evaluating the computational complexity from profiling, the DCT and IDCT modules are replaced with fast algorithms that have negligible degradation of picture quality. In addition, a novel fast variable length decoder is developed using a hierarchical table lookup technique.

In the subsequent sub-sections, we will describe the optimization techniques of each category for the encoder and decoder.

5.2.2 Removal of the unused procedures, parameters, and data structures

In this Section, we will describe the modifications of the program and data structures. From the MPEG-4 specification [1], the MPEG-4 Simple Profile syntax includes basic elements (BA) such as I-VOP, P-VOP, AC/DC prediction, 4-MV, unrestricted MV, three options for error resilience (ER) tools and short header (SH). This will reduce the code size to one third of the original code sizes for both encoder and decoder. For Simple Profile encoder and decoder, we remove the unused tools from the FCD video reference software where conditional statements can be removed and the required modules are improved. Consequently, we retain the necessary modules and code bases for the encoder and decoder that are complaint with MPEG-4 Simple Profile.

As for the input parameters used in the control file *.ctl and configuration file *.cfg, the unused coding parameters for Simple Profile encoder and decoder are removed along with the variables, data structures, and constants that are related to these unused parameters. For example, the parameters and the corresponding code bases for the scalability and Quarter-pixel motion estimation, shape coding, global motion estimation and compensation, and multiple object composition are deleted from the optimised code bases for the MPEG-4 Simple Profile. Due to the removal of unused parameters and their associated code bases, we can speed up the reading of parameter file and the actual coding rate.

For the MPEG-4, there are many common modules like the motion compensation (MC), inverse discrete cosine transform (IDCT), etc. are used in the encoder and decoder. In the original program, they are assigned to the directory “common”, which requires the compiler to compile and link the unused code bases. The discrete cosine transform (DCT) for the decoder is a good example to illustrate such scenario. For the further reduction, we examine all the files in the “common” directory, and split the files for the encoder and decoder into two separate directories. Thus, the separation of the code bases for the encoder and decoder leads to streamlined code. Other improvements include removal of unnecessary VLD tables, argument lists, and logical execution control.

5.2.3 Revision of the code bases for saving the execution time and code sizes

In this section, we will describe how we rewrite the source code.

Improved encoding process

(1) Total number of the generated bitstreams

Referring to the number of the temporary and output bitstreams, we reduce to one output bitstream when all error resilient tools are disabled. When in the error resilient coding mode, we use one output bitstream and two temporary bitstreams, which are used for generating the video packet and supporting the data partition. The resultant bitstream usage is summarized as in Table 3.

As for the bit stream access, the original reference software writes each bit of the encoded data in binary representation into a 16-bit data bin for temporary storage. After the encoding of each VOP or when each video packet is encoded, the bits in the buffer are moved again to the output buffer bit-by-bit. Such an access approach not only takes a long time but also wastes memory. To address this issue, the coded bits are written directly into the bitstream 1 in Table 3 for the start codes and stuffing bits, which shall be byte-aligned. For the bits representing the motion information and texture information, the magnitude and bit-length of its binary format are stored into a 16-bit data bin of two temporary bitstreams: bitstream 2 and bitstream 3 as shown in Table 3, respectively. When the encoding of each video packet or each VOP is finished, the bits in the two temporary bitstreams are written to the output bitstream 1 in Table 3. Thus, the bitstream access time is reduced by storing the encoded data at 16-bit data bin in the temporary buffers and writing multiple bits simultaneously to the output bitstream.

Table 3 — Substitution of the bitstreams in the original reference software for efficiency

Bitstream Declaration	Appearance Module	Byte alignment	Replacement
Header_bitstream	BitstreamPutVols	NextStartCode	bitstream1
Header_bitstream	PutUserDataToBitstream	NextStartCode	bitstream1
Texture_bitstream	VopCode	NextStartCode	bitstream1
Aux_bitstream	BitstreamPutVopHeader	VOPheader:	bitstream1
mottext_bitstream	VopCode	NextStartCode	bitstream1
motion_comb_bitstream	VopCode	NextResyncMarker or NextStartCode	bitstream1
text_header_comb_bitstream (for packet header information)	VopCode	NextResyncMarker or NextStartCode	bitstream2
text_data_comb_bitstream (for packet texture data)	VopCode	NextResyncMarker or NextStartCode	bitstream3
comb_bitstream	VopCode	NextResyncMarker or NextStartCode	bitstream1
Aux_bitstream	BitstreamPutGOV	GOV header	bitstream1

(2) Raw data access

Memory access and data I/O often are bottleneck for speed enhancement. This issue can be resolved with decreased frequency of memory access and data I/O. Thus, a new approach is proposed to read and write the raw data from the files. The new code for data I/O can fetch the raw data of each color component into the corresponding frame buffer directly. To write back to the stored files, a temporary buffer is allocated to store the pixels of reconstructed VOP and the three components of each VOP are written back into the file simultaneously. Based on the proposed I/O access method, the execution time is reduced to one third of the time taken by the original methods.

(3) Memory allocation and access

Memory access is critical so we examine the whole program to reduce the unnecessary memory access. We found there is an inefficient implementation of memory allocation and release in the original software. To optimize the data flow, the variables and data structure are rearranged and updated. All the unnecessary memory accesses and data movement are removed to speed up the coding process. In the original reference software the memory is allocated as needed. The advantage for such algorithm is that memory can be used dynamically. However, for system with more memory capacity, we can allocate all the needed memory at the beginning and release them at the end which is more clear and efficient.

(4) Conditional jumps

The control flow is simplified by removing redundant conditional statements. For example, in the BlockDequantH263() modules, the condition checking at the end of the module

```
for (i=0;i<64;i++)

if (rcoeff[i]>(lim-1)) rcoeff[i]=(lim-1);
   else if (rcoeff[i]<(-lim)) rcoeff[i]=(-lim);
```

is redundant. The magnitude saturation for each reconstructed coefficient is done immediately after the inverse quantization with the statement

```
rcoeff[i] = MIN(2047, MAX(-2048, rcoeff[i] ))
```

With the removal of the redundant branches, a single statement replaces multiple statements and branch instructions. Each group of the conditional statement is reordered based on their frequency of occurrence for efficient logical execution control.

(5) Arithmetic operations

We avoid using the arithmetic operations including division (/) and modulation (%) in the code bases for speedup. Time-consuming operations are replaced with low-cost operations as follows.

- The division (/) or multiplication (*) by an integer of value in the form of 2^k, k>0, is replaced with left shift operation (>>) and right shift (<<) operation, respectively. For the values not in the form of 2^k, the division is replaced with multiplication of its reciprocal in floating point precision or double precision.
- The modulo operation by an integer of value in the form of 2^k, k>0, is replaced by the Logical AND operation (&) with an integer of value $(2^k - 1)$.

(6) Un-restricted Motion Estimation and Motion Compensation

For un-restricted ME/MC, the source frame is padded on the boundaries. Since only the reconstructed frame is used for ME and MC, so the VOP padding of the current frame is removed.

(7) Interpolation for the half-pixel motion estimation and motion compensation

For the half-pixel ME and MC, the half-pixel motion vectors are generated with bilinear interpolation and stored in the padded VOP in the reference frame buffer. In the original reference software, the frame interpolation is done twice for the half-pixel ME and the half-pixel MC. Thus, we allocate a frame buffer prior to the frame interpolation and store it into this buffer before the half-pixel ME. The interpolated frame in the buffer is reused for the half-pixel MC. Consequently, we can save the pre-processing time of the half-pixel motion compensation.

(8) Motion Compensation

Since Simple Profile coding scheme supports only rectangular shape block coding, boundary checking is removed for the center part of the picture.

(9) Calculation of the DC scale

Some calculations are replaced with table lookup. For example, the calculation of the DC scale is substituted with table lookup as described in the MPEG document [4]. The first parameter indicates the type of DC_scale and the second parameter indicates the quantization parameter (QP) used.

```
Int cal_dc_scaler[2][32]=
/*QP=0  1  2  3  4  5  6  7  8  9  10  11 12  13  14  15  16 17  18  19 20  21  22  23  24  25  26 27
28  29  30  31 */
{   {0, 8, 8, 8, 8, 10, 12 , 14, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 34, 36, 38,
40, 42, 44 ,46},
    {0, 8, 8, 8, 8, 9,  9, 10, 10, 11, 11, 12, 12, 13, 13, 14, 14, 15, 15, 16, 16, 17, 17, 18, 18, 19, 20, 21,
22, 23, 24 ,25},
};
```

(10) Module integration

In the original reference software, deep calling path of 12 levels is used for both the encoder and decoder. Several modules are merged to decrease the number of function calls and the calling depth.

In addition, procedures with only a few statements are also re-worded as Macros to avoid frequent calls. Similarly, the procedures used just for call transition is removed and the caller can directly interact with the underlying called subroutine. For example, AddImage() has two subroutines including AddImageI() for short integer values and AddImageF() for floating point values. To speed up the coding process, we call directly the AddImageI() and AddImageF() instead of AddImage().

As shown in Figure 9, we demonstrate how the original reference software reconstructs a P-VOP via motion compensation, quantization, dequantization, and magnitude saturation. The reconstructed ME and MC residuals of the reference VOP are saved in the buffer indicated as “Rec_cur”. Prior to motion compensation (MC), the reconstructed frame Rec_cur is cloned in another buffer indicated as “comp” by the procedure cloneVop(), which duplicates all of the texture data, quantization parameters, motion vectors and header information from Rec_cur to comp. The frame in the buffer comp is motion-compensated and stored in the reconstructed frame. The prediction residuals of the current frame are derived through the procedure named as SubImage() and saved back to the buffer “error vop”. The residuals of the current frame are quantized and dequantized into reconstructed residuals and saved in another buffer “error_vop1”. The reconstructed residuals in error_vop1 are added back to the reconstructed reference frame in comp by the procedure AddImageI() to reconstruct the decoded current frame. Every pixel within the reconstructed frame of the current VOP is saturated into a range of [0, 255] by the procedure ClipImage() as the bit depth equals to 8.

Observing the reconstruction process, we can find that the code bases in the gray area can be merged to reduce the unnecessary data migration and function calls. Since the buffers Rec_cur and comp has no conflict usage, the module cloneVop() can be removed and all data and information in the buffer Rec_cur can be used for the reconstruction. Since the reconstruction and saturation of pixels are performed sequentially, the modules AddImageI() and ClipImage() can be merged into single module. By merging the non-conflicting sequential modules, we can save not only the overhead of the function calls but also the code sizes.

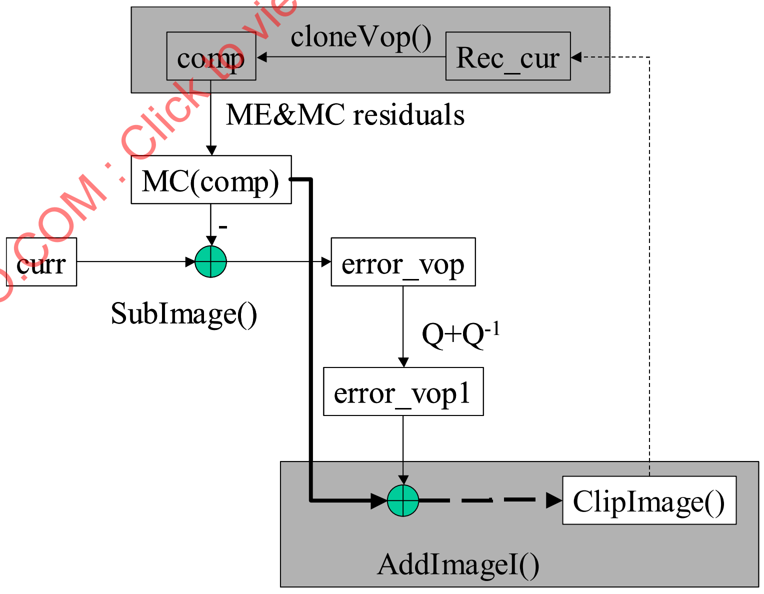


Figure 9 — Illustration of module mergence for reducing the data movement

Improved decoding process

Most of the enhanced methods used in the encoding process, which covers the improvement for the raw data writing, conditional jumps, arithmetic operations, calculation of the DC scale, and module integration, can be applied to the decoding process. The speedup methods particular for the decoding process are described as in the following.

(1) Bitstream reading

To minimize the frequency of the memory/file accesses and data I/O for handling the input bitstream, the bit-wise operations including loading, showing, reading, and flushing are modified. The original code bases employ a three 8-bit register and a buffer with size of 2048 bytes between the bit-wise operation and the bitstream file, which leads to triple data movement including fetching from the file to the memory buffer, copying from the memory buffer to the register for caching, and computations of the value from the bits in the cache. Such data movement takes significant computational cycles to access the bits for decoding. Thus, we propose an approach that adopts the buffer with size 2048 bytes as a data cache. The bit-wise operations are done in the cache for decoding. Therefore, two data movements are required by skipping the step to cache the data from the memory buffer to the register. Furthermore, the bit-wise operations can be simplified based on the proposed structure of bitstream access.

(2) Memory allocation and access

Referring to the memory usage, as in the encoder, we calculate the required buffer size and allocate the memory once. In addition to the approaches at the encoder, we switch the buffer pointers instead of moving the data from one buffer to another buffer to reduce data movements such as copying reconstructed frame for image output, and for removing unnecessary data processing in generating padding frame and in interpolating half-pixel reference frame.

(3) Motion Compensation

In the decoder, the motion compensation for each handling block covers the following four combinations including single motion vector with full-pel offsets, single motion vector with half-pel offsets, 4 motion vectors with full-pel offsets, and 4 motion vectors with half-pel offsets. The MC with half-pel offsets is the most computational expensive part. Since the half-pel MC needs more overhead bits in the bitstream, thus this mode is less frequently used, which is consistent with the observations that less than 20% of the coded blocks in the test bitstream need half-pel MC. Thus, to reduce the computational load of these parts, the following three strategies are adopted.

A. Perform the frame interpolation when demanded

When the coded block needs half-pel MC, we apply the bilinear interpolation to the reference pixels in a way specified in the block header. Thus, we not only reduce the time in generating the whole interpolated frame, but also remove a huge buffer to store the interpolated frame.

B. Load the processing pixels from the memory into the registers only once

Since the interpolation is coupled with the MC module, the bilinear interpolation of the two successive pixels is required. In the video specification, the bilinear interpolation can be done in the horizontal, vertical, and diagonal directions, where the horizontal interpolation is used to generate the sub-pel data in x-axis and the vertical interpolation is used to derive the sub-pel data in y-axis, and the diagonal interpolation is used to synthesize the sub-pel data in both x and y axes. For the horizontal and diagonal interpolation, the interpolation window is moved right to the next location by one step in integer coordinate. Thus, the neighboring interpolation window will overlap at the sub-pixels. To save the load time, we put the overlapped pixels of the successive two filtering window in a register and reuse it to save the memory load overhead.

C. Reuse the overlapping pixels of the neighboring locations

For the diagonal interpolation, which has a 2x2 filtering window, we apply bilinear operation in a separable manner. In other words, the 2x2 filtering is substituted with the following steps:

1. For the filtering window located at (x, y), sum the values of the neighboring pixels located at (x, y) and (x, y+1) and save the result into a register A.
2. Sum the values of the neighboring pixels located at (x+1, y) and (x+1, y+1) and save the result into another register B.
3. The pixel value found by diagonal interpolation is identical to the averaged magnitude of the values in the registers A and B and the rounding control amplitude.

4. For the next location (x+1, y), the content in register B is moved to register A. The register B is refreshed with value equal to the summation of the pixels at (x+2, y) and (x+2, y+1).
5. The same step is applied for each location of a block until all locations are traversed for diagonal interpolation.

Consequently, we can save one addition operation and two load operations for the diagonal interpolation.

5.2.4 Use of the existing fast algorithms for the computational burden modules

In order to accelerate the program operation, many fast algorithms including fast ME, and fast DCT/IDCT have been proposed. Most of them sacrifice the memory capacity or quality to get speed improvement. For the OM software, we adopt the algorithms that are widely used.

Integer Forward Discrete Cosine Transform (DCT)

This Integer DCT is a common algorithm that combines an efficient DCT algorithm and integer operation. The efficient DCT algorithm is to use butterfly structure to reduce the complexity of exhaustive computation. Since the fixed-point operation (integer operation) is much more efficient than floating-point algorithm, we can enlarge the coefficients to compensate the operation errors, and then shift down to the original scale. By properly enlarging the operation coefficients, we can get a gain of more than 10 times improvement for the single module with negligible quality loss in Peak-Signal-to-Noise-Ratio (PSNR).

Integer Inverse Discrete Cosine Transform (IDCT)

The Integer IDCT is a common algorithm that combines an efficient IDCT algorithm and integer operation. The efficient algorithm is to use butterfly structure to reduce the complexity of the exhaustive computation. Such algorithm can be found in MPEG-2 codec to speed up the operation without too much quality loss.We applied integer IDCT in the OM program, and get a great reduction from 13% to 1% in an encoder with MVFAST, and from 28% to 4% at the decoder.

Fast Motion Estimation (ME)

The MVFAST detailed in Section 2 offers high performance both in quality and speed and does not require memory to store the searched points and motion vectors. We turn on this algorithm and get a gain of 4.5 times improvement.

Improved Half-Pel Motion Estimation

As the calculation of matching criterion using Sum of Absolute Difference, the half-pixel motion estimation is improved through the following two approaches.

(1) Remove the recalculation of the SAD value for the central location

(2) Adopt an early rejection approach in calculating the SAD of the remaining locations.

Fast Variable Length Decoder Using Hierarchical Table Lookup

A. Introduction

A hierarchical table lookup technique is used to improve the variable length decoding (VLD) speed in the MPEG-4 OM software. In a typical VLD as shown in Figure 10, a memory fetch in the VLD retrieves a fixed number of data bits each time and move them into the register. For a bit stream containing consecutive short code symbols, a single fetch allows the decoding of multiple short symbols while decoding a long symbol may need multiple fetches. To reduce the number of memory fetch, a hierarchical VLC table structure is designed by exploring a fully expanded binary tree that contains all possible combinations of 13-bit long bit patterns. The bit number 13 is chosen to be the maximum length of legal symbols defined in the MPEG-4 video specification. Having analyzed all legal symbols, we split the binary tree into two layers. The first-layer table started from the root of the binary tree is used for both decoding multiple short symbols and for indexing the second-layer tables, which use the remaining bits as the prefix to form another legal symbol. The second-layer tables store the postfix bits for the symbols having more than 6 bits.

We examine the decoding performance using different hierarchical structures of VLC tables. Using the aforementioned VLC table structure, the experimental results show that we achieve about 12%~107% speedup on the VLD module execution and 6-13% improvement on the overall execution time over the previous version MPEG-4 Optimization Model (OM) software [3].

B. MPEG-4 Simple Profile Video Decoder

An MPEG-4 Simple Profile video decoder consists of modules for variable length decoding (VLD), inverse zigzag scan, inverse quantization, IDCT, and MC. Taking in the input bitstream, the VLD module decodes it into a sequence of the triplets {*run, level, last*} defined by the VLC tables, in which the symbol *run* is the run length of zero-magnitude DCT coefficients, the symbol *level* represents the amplitude of the quantized DCT value, and the symbol *last* indicates the end of a block. Together, a series of triples {*run, level, last*} contain sufficient information to reconstruct the DCT coefficients.

Figure 10 shows the VLD process for each 8x8 block.

Step 1: Initialize the 8x8 block DCT with zero values.

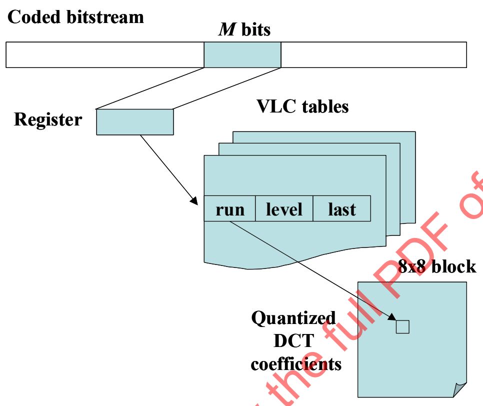


Figure 10 — Flow chart of the conventional MPEG-4 variable length decoder

Step 2: Read in *M* bits from the coded bitstream and move them into the register. In this case, *M*=12.

Step 3: Use the value (bits) in the register to select the VLC table entry to decode the register content. If the register contains all the bits needed to represent a symbol, its triplet {run, level, last} can be retrieved by table lookup and such a case happens mostly. As for a long symbol, it requires a second or third fetch to obtain all the necessary bits for decoding. Move the register read pointer to the beginning of the next symbol.

Step 4: Repeat Steps 2 and 3 until the “last” field in the corresponding VLC table entry is true, which indicates the end of coding this block.

The above VLC table needs 4096 entries to represent all possible symbols or their prefixes. Typically, short symbols occur more often than long symbols. Thus, to improve the VLD, we propose a fast algorithm that can decode multiple symbols for a single fetch and make good use of the remaining bits to conduct the decoding of next symbol.

C. A Fast VLD Decoder

Based on the discussions in Section 5.2.4 B, a hierarchical VLD table lookup method is used to improve the VLD efficiency. There are two stages in this approach: the hierarchical table construction stage and the VLD decoding stage.

C.1. Hierarchical Tables Construction

The hierarchical VLC tables are constructed using a fully expanded binary tree, which has the depth identical to the maximum length of legal symbols defined in the MPEG-4 14496-2 video specification. We split this binary tree into two layers. The first-layer (Layer-1) table has two types of entries: some contain multiple short symbols and the other are pointers (prefixes) pointing to the second-layer (Layer-2) tables for decoding a long symbol. The second-layer tables store the postfix bits for the long symbols.

Based on this hierarchy, we construct a group of hierarchical VLC (HVLC) tables including one Layer-1 table and eight smaller Layer-2 tables. The Layer-1 HVLC table is indexed by 12 bits and the Layer-2 HVLC tables are indexed by 3 to 6 bits. As shown in Figure 3, the Layer-1 HVLC table has 6 columns (fields) and every Layer 2 HVLC table has 4 columns (fields). The 'Run' and 'Level' fields contain the values of the triplet and the sign of each symbol is separatly stored and added back to the amplitude of 'Level' for the DCT coefficient reconstruction. The 'flag' is used to indicate the recent VLD status including the end of block (EoB), the escape code, and the next table to be used. The final field 'Flushbits' means how many bits of the fetched bits have been used in decoding and thus can be discarded. As shown, we assume the first M_1 bits out of the M bits have been used and the remaining bits belong to the next symbol. To save memory, we use 8 bits for each column of the HVLC tables, since the run value is between 0 and 63 and the level values except those in Escape mode are in the range of [-27, 27].

A summary of HVLC tables is shown in Table 4. Since the percentage of the Huffman codes longer than 12 bits is small (16/102=15.67%) in both MPEG-4 Intra and Inter TCoeff VLC tables, the Layer-1 HVLC table uses a 12-bit index.

C.2 VLD using Hierarchical Table Lookup

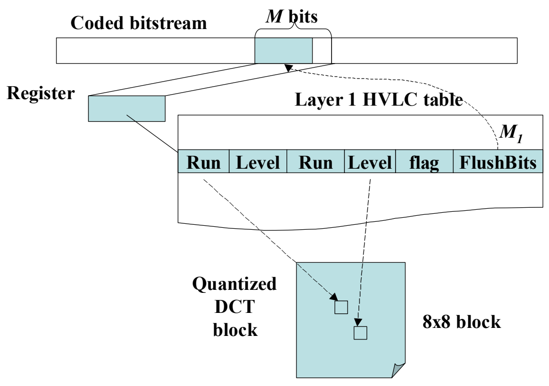
Depending on the combinations of the Huffman symbols, the fetched bit patterns fall into one of the following three categories as shown in Figure 11. The classification is based on the number of symbols that can be decoded in one fetch and the short symbols with length shorter than 7 bits. The detailed description for the three categories of patterns is as follows.

Case 1: 2 symbols

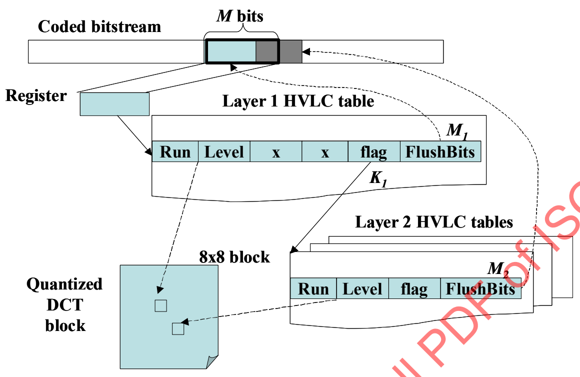
Since most frequent symbols are of short length, we can retrieve 2 symbols at the same time, which leads to a fast VLD. In this case, only the Layer-1 HVLC table is searched for the VLD. With the fetched M-bit data, we can find the matched binary pattern and decode the run and level values of these two symbols. Assume only the first M_1 bits out of the M bits have been used, the remaining bits belong to the next symbol.

Table 4 — Summary of the Hierarchical VLC tables

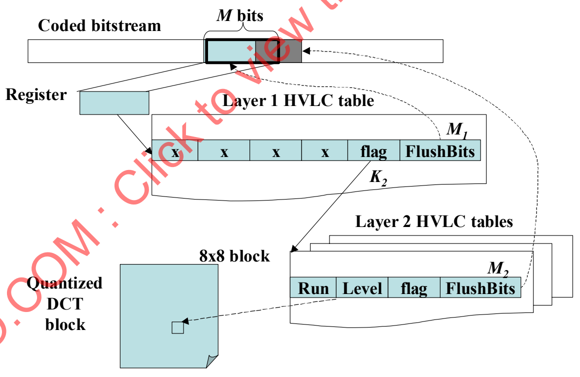
Table index	Prefix code pattern	# of index	# of columns	Actual size (bytes)
0		12	6	24576
1	0011	3	4	32
2	010	4	4	64
3	000000	6	4	256
4	0010	4	4	64
5	0001	6	4	256
6	00001	6	4	256
7	00000100	4	4	64
8	00000101	5	4	128
Total memory sizes				25696



a) Case 1: 2 symbols



(b) Case 2: 1 or 2 symbols plus 1 link



(c) Case 3: 1 link

Figure 11 — Flow chart of the proposed MPEG-4 variable length decoder

Case 2: 1 or 2 symbols plus 1 link

In this case, one or two symbols can be decoded based on both Layer-1 and one link (pointer) is used to find the postfix bits of the next legal symbol in Layer-2 HVLC tables. With the fetched *M*-bit data in the register, we can find the matched binary pattern and decode the run and level value of the first one or two symbols. The 'flag' points to the chosen Layer-2 HVLC table with label K_1 and then we can retrieve the run and level values of the next symbol through one additional fetch and searching over the K_1-th Layer-2 HVLC table. Next, assuming the first M_1 bits out of the *M* bits have been used, the remaining bits are belonging to the next symbol.

Case 3: 1 link

Now assume the incoming symbol has a length longer than the maximum index value of the Layer-1 HVLC table, which is *M*. To decode this symbol, the Layer-1 table points to a proper Layer-2 table through the use of 'flag'; say, the K_2-th Layer-2 table has been selected. We can retrieve the run and level values of this symbol by one additional fetch and search over the K_2-th Layer-2 table.

To further reduce the memory fetch frequency, we can read in more bits at a time. To this end, the minimum number of bits per fetch is equal to M=12 (index of the Layer-1 table) plus 6 (the maximum index length of the Layer-2 tables). Another advantage of fetching more bits at a time is that we can flush only once (instead of twice) for decoding a symbol in case 3 to save memory access. This technique can also be applied to case 2 to save memory access.

D. Complexity Analysis

Based on the statistics of the Huffman codes, the expected number of symbols that can be decoded in one memory fetch is higher than 1.5 symbols. Thus, we expect a 50% or higher improvement on the VLD module. In addition, we fetch all the bits at one instance that are sufficient to index the Layer-1 and Layer-2 tables for the second case (in Figure 11). Consequently, the HVLD provides about 2 times speedup in execution timing profile. As for the memory size, as shown in Table 4, we use approximately 25.6k bytes for the HVLC tables for holding the Intra and Inter texture coefficients separately. The total memory size is thus about 51.2k bytes.

5.2.5 Optimised Simple Profile encoder and decoder

The preliminarily released code of the optimised reference software is placed in a new directory of the package MoMuSys-FPDAM1-1.0-021015_nctu.zip

.\OptSimple\vm_enc : for encoding routines

.\OptSimple\vm_dec : for decoding routines

.\OptSimple\vm_common : for commonly used routines

.\OptSimple\App : for encoding and decoding workspaces

.\resul\OptSimple : for control and configuration files

.\OptSimple\huffman: for HVLC Huffman tables

5.2.6 Experimental Results

Environment

- Source video sequences: Foreman
- Input/output frame number: 300/100
- Input/output frame rates: 30/10 fps
- Picture sizes: CIF (352x288) and QCIF (176x144)
- Format: YUV (4:2:0)
- Coding type: I-PPP
- GOV: used once prior to the first I-VOP
- Rate control: VM5+
- Platform: Intel PIII-600 MHz desktop PC, and Windows 98
- Optimization tools for complier and linker: the default tools including "Maximum Speed", "Blend *" processor, "__cdecl" calling convention, and "Only __inline" in Microsoft Visual Studio 6.0 are used for making release mode executable files.
- Fast Motion Search: MVFAST in N4554
- Integer DCT / IDCT

Simulation Results

Execution time in frames per second is increased from 39 % to 3 875 % as shown in Figure 12 through Figure 15. The profiling results for different combinations of tools are shown in Figure 16 and Figure 17.

Table 5 — Code size reduction of MPEG-4 Simple Profile encoder and decoder

	Encoder		Decoder	
	Size (byes)	Ratio	Size (byes)	Ratio
Momusys	806912	100.00%	536576	100.00%
N5550	217088	26.90%	163840	30.72%

Table 6 — Combinations of MPEG-4 Simple Profile visual tools

Number	Basic	GOV	Error resilient
1	X		
2	X	X	
3	X	X	X

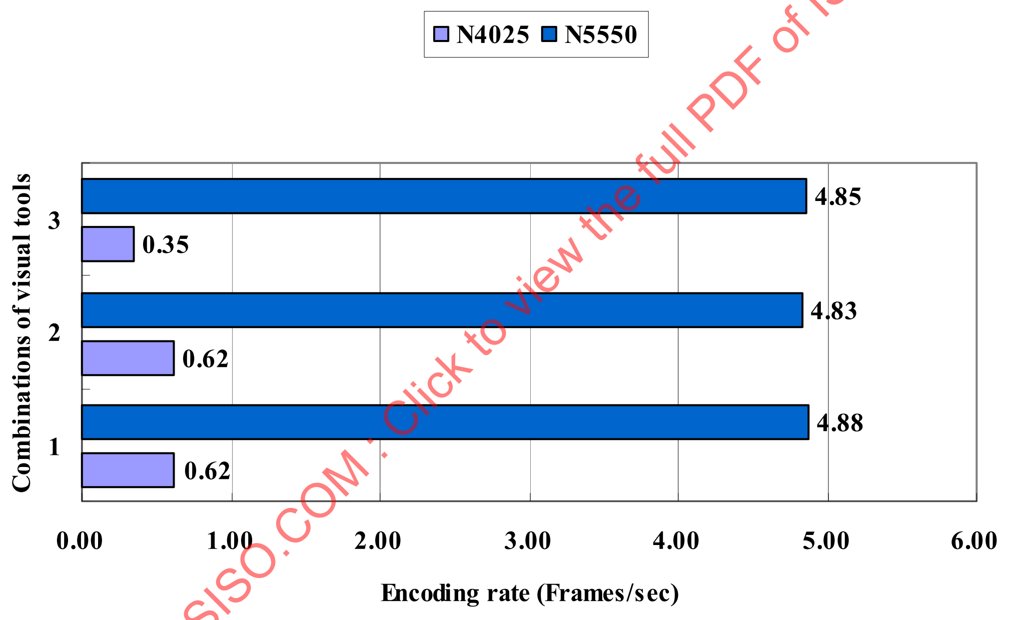


Figure 12 — The encoding rates (frames/sec) of the reference and the proposed software for Foreman sequence in CIF format