Low-Complexity MPEG-4 Shape Encoding towards Realtime Object-Based Applications

Euee Seon Jang

Although frame-based MPEG-4 video services have been successfully deployed since 2000, MPEG-4 video coding is now facing great competition in becoming a dominant player in the market. Object-based coding is one of the key functionalities of MPEG-4 video coding. Real-time object-based video encoding is also important for multimedia broadcasting for the near future. Object-based video services using MPEG-4 have not yet made a successful debut due to several reasons. One of the critical problems is the coding complexity of object-based video coding over frame-based video coding. Since a video object is described with an arbitrary shape, the bitstream contains not only motion and texture data but also shape data. This has introduced additional complexity to the decoder side as well as to the encoder side. In this paper, we have analyzed the current MPEG-4 video encoding tools and proposed efficient coding technologies that reduce the complexity of the encoder. Using the proposed coding schemes, we have obtained a 56 percent reduction in shape-coding complexity over the MPEG-4 video reference software (Microsoft version, 2000 edition).

Keywords: MPEG-4, encoding complexity, real-time video, shape coding.

Manuscript received Apr. 9, 2003; revised Oct. 30, 2003.

This work was supported by the Hanyang University Research Fund.

Euee Seon Jang (phone: +82 2 2290 1086, email: esjang@hanyang.ac.kr) is with the Software Division, College of Information and Communications, Hanyang University, Seoul, Korea.

I. Introduction

Started in 1993, the first edition of MPEG-4, entitled *Coding of Audiovisual Objects*, was completed as the international standard in 2000. One of the main features of MPEG-4 is object-based coding, which allows the user to become the content creator, ultimately, by reusing and editing the audiovisual objects inside the MPEG-4 bitstream.

The visual part of MPEG-4 includes not only conventional video coding, but also the coding of graphical entities such as 2D/3D mesh coding and face and body animation [1]. Hence, the first edition of MPEG-4 visual had nineteen profiles! However, the first commercial service using MPEG-4 was not object-based video coding (OVC), but frame-based video coding (FVC). In fact, the only profile that is widespread for mobile and Internet applications is the Simple profile based on FVC [2]. This is not a surprise because the Simple profile was designed to fulfill the need of low bit-rate applications with low complexity. Also, the immediate need of MPEG-4 from the industry was for low bit-rate applications.

In Internet and mobile applications, MPEG-4 video coding is now facing a hard time due to its competitors (such as Microsoft's Windows Media Technologies and Realnetworks' Real)[3]-[5]. Moreover, the MPEG committee (ISO/IEC JTC1/SC29/WG11) with its counterpart in ITU is now making a new video coding standard called MPEG-4 (Part 10) advanced video coding (AVC) as a more efficient video coding tool. MPEG-4 AVC compression is expected to be twice as efficient as MPEG-4 video coding [6].

Nevertheless, the competitive edge of MPEG-4 video coding remains intact: object-based coding. With the objects inside a scene, the user can now edit and/or reuse the contents in the scene. Although content-based editing is not new in other

media, introducing the reusability of video objects will enable interactive multimedia services in the near future.

The number of commercial applications of object-based MPEG-4 video are not many. There are a few issues that have blocked the wide diffusion of object-based MPEG-4 video:

- **Limited real-time, arbitrarily-shaped object creation**: The boundary of each video object has to be captured to every frame. Even though the blue screen technique using chroma keying is a major on-line segmentation method, it is used for limited environments (i.e., weather forecasts, real-time broadcasting). The maturity of most on-line segmentation technologies is distant from high-fidelity segmentation.
- **Increased complexity due to shape coding**: The conventional frame-based video coding consists of two parts: motion and texture coding. Motion estimation and motion compensation (MEMC) are the most expensive parts in the encoding process. The decoding process of frame-based video is relatively less complex than the encoding process. Due to this fact, real-time encoding with MEMC is often not possible. Object-based video coding adds shape-coding complexity on top of the frame-based video coding. Hence, object-based video is considered more complex than frame-based video.
- **No existing market and support**: For images or vector graphics objects (i.e., FLASH), the object-based representation is already in use. This notion, however, was never tried successfully in video. Hence, there has been no immediate market need for object-based video, even though the technical maturity is approaching the state of the art. But the emerging market is from digital multimedia broadcasting, which demands object-based video services. In such services, the user will be able to customize the video services on her own.

In this paper, we address the complexity issues of object-based video coding toward real-time video services. The complexity of MPEG-4 OVC has been researched in both software and hardware [7]-[9]. One common observation from the previous research on either object-based or frame-based coding is that MEMC takes up the largest part in the encoding process. Even with shape coding in MPEG-4 OVC, MEMC is consuming more than 75 percent of the encoding run time [7], [8].

Regarding MEMC, highly efficient technologies are being documented as a part of the informative standard [10], [11], promising a substantial reduction of computational complexity. This means that the optimization of non-MEMC parts is becoming more important in OVC.

The portion of shape decoding over the MPEG-4 OVC decoder is relatively larger than that of the MPEG-4 OVC encoder. It should be noted, however, that the decoding process in MPEG-4 is much faster than the encoding process, since most time consuming processes are in the encoding stage. Even the optimized encoders are reported to be three times more complex than the optimized decoders [7]. In order to achieve real-time object-based video services, both the encoding and decoding processes have to be optimized. So, more emphasis has to be given on the encoding side for further optimization.

Most previous research was conducted in such a way that the reduced complexity is from efficient implementation while keeping the original framework of the encoding and decoding processes intact. To support real-time services, however, further complexity reduction on the encoder side is imperative, since digital broadcast studios, mobile handsets, and Internet PCs will need a faster encoder, and not just a faster decoder.

We addressed the design-level optimization of the non-MEMC parts of an MPEG-4 OVC. Shape coding and padding processes are the typical, expensive, non-MEMC parts. We provided further complexity analysis and proposed efficient encoding techniques thereof. The organization of the rest of this paper is as follows. In section II, the basic elements of MPEG-4 video coding tools are explained. Efficient coding algorithms of the MPEG-4 video coding tools, as well as their simulation results, are given in section III. Finally, we summarize the paper in section IV.

II. MPEG-4 Object-Based Video Coding

1. Concept of Object-Based Video

According to the MPEG-4 requirements [12], object-based video coding shall provide the following features:

- Object-based representation
- Object-based bitstream manipulation and editing
- Object-based random access
- Object quality and fidelity
- Object-based coding flexibility
- Object-based scalability

The main theme in MPEG-4 OVC is to allow the user to conveniently play, edit, and reuse MPEG-4 video objects. Figure 1 shows the concept of object-based video. In conventional frame-based coding, Fig. 1(d) shows one single frame to encode. In MPEG-4 OVC, a scene may have multiple video objects as shown in Fig. 1(a) and 1(b). If a video object is not rectangular shaped, additional information called shape (or alpha map) has to be encoded in the bitstream. The shape information is used to determine the transparency (or opaqueness) of the given pixel, as shown in Fig. 1(c). The shape information is often called a binary shape.[1] Once

1) In order to change the level of transparency of the given pixel, the user can choose to use 'grayscale' shape data. The grayscale shape is then composed of a grayscale value and its binary shape. The coding of a grayscale shape is called grayscale shape coding in MPEG-4.

represented by MPEG-4 OVC, the user can edit and reuse the video objects with other contents, as shown in Fig. 1(e).

The coding of video objects is comprised of three parts: shape, motion, and texture. In the following sections, more details of the decoding and encoding procedures are explained.

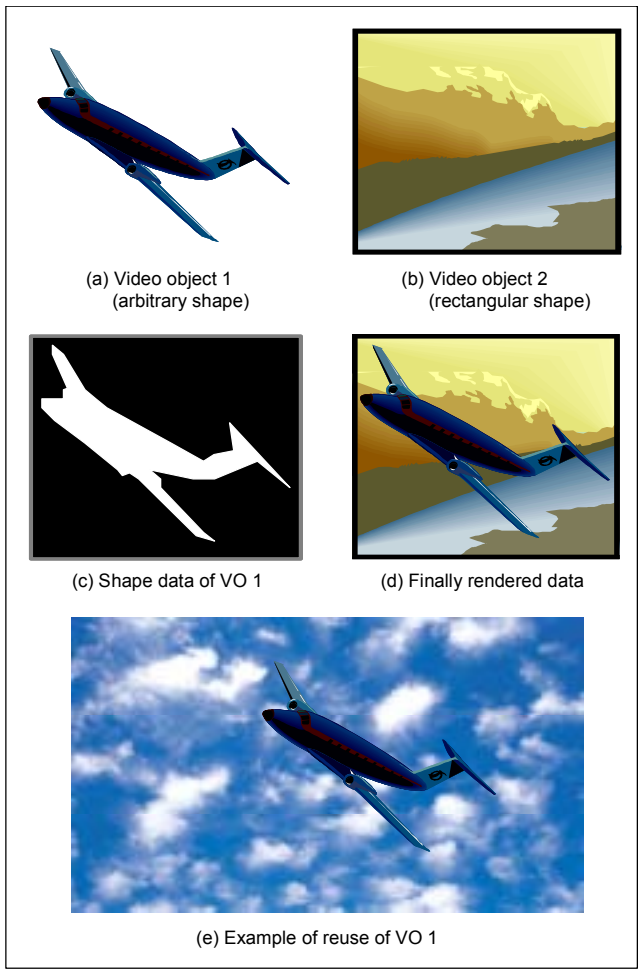

Fig. 1. Concept of object-based video.

2. MPEG-4 OVC Decoding Process

The overall complexity of the MPEG-4 OVC decoding process is much less than that of the encoding process. So, the discussion on the decoding process is given first. The normative parts of MPEG-4 Standards are in the bitstream and the decoding process. How to form the bitstream is decided by the encoder, not by the decoder. This makes research on the decoding process focus mainly on pure optimization (i.e., code optimization, data reuse, special architecture design, etc.)[13].

Fig. 2 shows the decoder structure of MPEG-4 OVC. It is not too different from that of an MPEG-2 video decoder, except in the area of shape decoding. If the decoded video object is not a rectangular shape, each macroblock (MB) will be tested if it is all transparent, all opaque, or has a boundary. For transparent MBs, only a padding process for motion compensation is applied. Padding is used to assign pixel values to the transparent pixels to support the motion compensation process. For opaque MBs, motion and texture decoding processes are applied. Finally, for boundary MBs, shape decoding and padding processes are applied in addition to motion and texture decoding.

The major focus of this paper is not on the decoding process, but on the encoding process. Those interested in the decoding process can find the details in [1], [2].

3. MPEG-4 OVC Encoding Process

There is no rigid rule on how to make an MPEG-4 OVC encoding process. This is due to the fact that the encoder does not need to be fixed in order to guarantee interoperability, and consequently, the encoder makes enough room for future efficient algorithms in the encoding process. Hence, it is difficult to say how complex the encoding process is. Here, we used MPEG-4 reference software for comparison [14]. The reference software contains both the encoder and the decoder. More

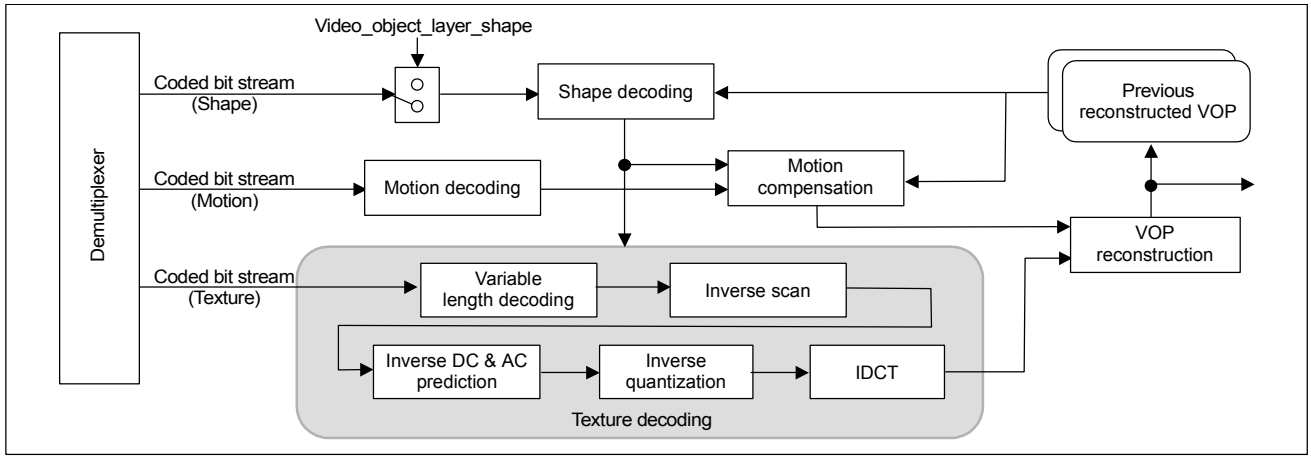

Fig. 2. MPEG-4 OVC decoder structure.

optimized algorithms can also be found in [10]. An extensive list of the encoding techniques that are used in the reference software can be found in [15].

Figure 3 shows the encoder structure of MPEG-4 OVC. Overall, the encoder is comprised of shape coding, motion estimation and compensation, and texture coding. It uses block-based coding with a macroblock of 16 × 16 pixels.

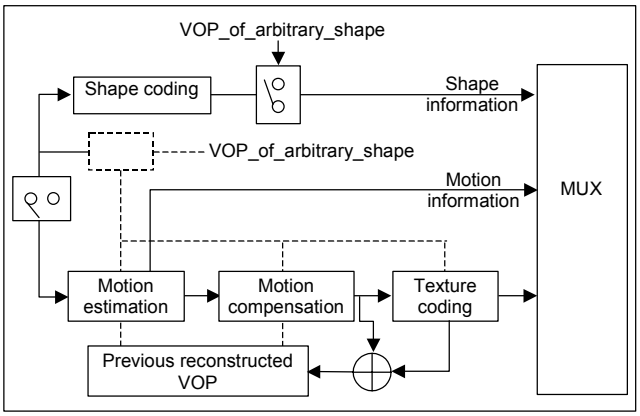

Fig. 3. MPEG-4 OVC encoder structure.

Table 1. MPEG-4 OVC encoding tools.

Category	Tool	Functionality	Cost
General	VOP formation	Compression	X
Shape	Mode decision	Compression	
	MEMC	Compression	X
	Subsampling	Rate Control	X
	Binary shape coding	Compression	X
	Grayscale shape coding	Compression	
Motion	Padding	Compression	X
	MEMC	Compression	X
Texture	Padding	Compression	X
	DCT	Compression	X
	Quantization	Compression	
	AC/DC prediction	Compression	
	VLC encoding	Compression	
Other	Error resilience	Error resilience	
	Scalable coding	Scalability	
	Sprite coding	Compression	

More specifically, the encoding process has many encoding tools, as shown in Table 1. Among the encoding tools, time-consuming ones are marked by an 'X', based on our own analysis and outside studies [7], [8], [14]. MEMC tools can be found in both shape and texture. Further, these MEMC tools for shape and texture are the most expensive tools in the encoding process. Many efficient techniques that improve the complexity of MEMC and texture coding have been researched and documented abundantly [10], [11], [15].

The parts that are relatively extensive in complexity focus on VOP formation, shape coding, and the padding process, all of which are explained in the following section.

4. VOP Formation

Each frame of a video object is called a video object plane (VOP). If the video object is arbitrarily shaped (i.e., not a rectangle), each macroblock may contain all-opaque pixels, partially-opaque pixels, and all-transparent pixels. Instead of using the entire frame, the tightest rectangular boundary of a VOP is normally used as a reference to form the macroblocks.

The size of the tightest rectangular boundary is often not in multiples of 16. As a result, some macroblocks include unnecessary transparent pixels. The intelligent VOP formation (IVOPF) method is used to efficiently assign the starting point of the first macroblock to maximize coding efficiency [15], [16].

Figure 4 shows an example of how a VOP is positioned. From the figure, the tightest rectangle of the given object is not equal to the multiple of MBs. An easy way to position a VOP is to select the upper-left corner of the tightest rectangle to be included in the first MB. In an IVOPF, a control MB is chosen to include the upper-left corner of the tightest rectangle as the bottom-right corner of the control MB. In the control block, every other pixel is selected to analyze the VOP formation and the number of MBs, such that the number of tested pixels in the control block is 64 (or 8 × 8), including the upper-left corner point (of the tightest rectangle).[2)]

Each test pixel out of 64 pixels in the control block becomes the candidate position of the first MB. Then, we can calculate the total number of MBs, the total number of all transparent MBs, and the total number of non-transparent (either all-opaque or partially-transparent) MBs. After testing all 64 test pixels, the optimal test pixel is selected, such that the test pixel yields the minimum number of non-transparent MBs.

The performance of IVOPF combined with the shape-adaptive discrete cosine transform (SA-DCT) was reported to provide a 5 to 10 percent gain [16]. SA-DCT, with its good compression performance, is an optional mode in MPEG-4 video coding. The performance of IVOPF with DCT and padding, which was not available in the literature, is provided in the next section. A potential weakness in the IVOPF

2) The reason for selecting every-other pixel is that the video format is 4:2:0. A chrominance pixel value (U or V) is associated with 4 (2×2) Y pixel values. Hence, the upper-left corner point of the tightest rectangle should be represented by even coordinate values.

technique comes from its repetitive searching procedure. The worst case is to calculate all 64 cases and compare the number of non-transparent blocks. This is a clear burden on the encoder side. The feasibility of IVOPF for real-time object-video services can be verified by analyzing its coding efficiency over its computational complexity.

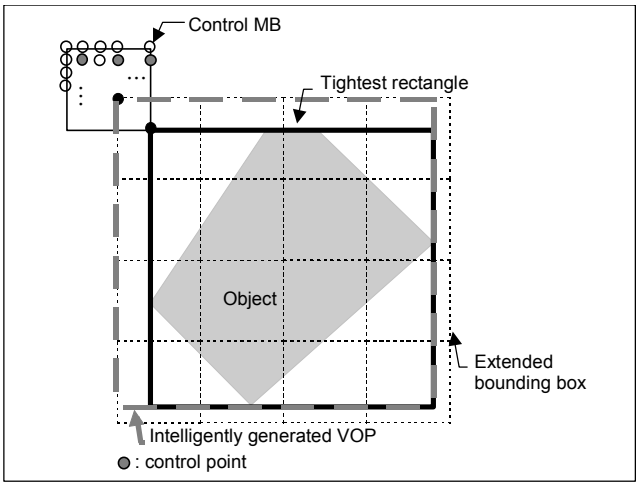

Fig. 4. Intelligent VOP formation.

5. Binary Shape Coding

The alpha map with 8-bit data is used to represent the transparency (or opaqueness) of the given image. If the opaqueness value is 0, the pixel is completely transparent. If the opaqueness value is 255, the pixel is completely opaque. If the opaqueness value is between 0 and 255, the pixel is to be proportionally blended with the background pixel. If the alpha map data contain any alpha value between 1 and 254, it is called a *grayscale* shape; otherwise, it is called a *binary* shape. So, binary shape data shall contain only 0 and 255 as its alpha values.

The relationship between a binary shape and its corresponding texture data is not as strong as the relationship between motion and texture, since the binary-shape data merely represent the opaqueness of the pixels, not the color intensity values. The current shape coding is macroblock-based to cope with motion and texture coding, although there are many competing technologies that are solely frame-based [17]. The transition from frame base to macroblock base resulted in a combined design of motion, shape, and texture. It is still worth noting that shape coding in nature is independent from the motion and texture coding process.

Details of binary shape coding can be found in [2], [15], [17], [18], and [19]. Overall, binary shape coding includes the following tools: mode decision, MEMC, subsampling, scan order decision, inter/intra context-based arithmetic encoding (CAE),[3] and scalable shape.

A. Mode Decision

Binary shape coding supports both lossy and lossless coding. If the shape is encoded as lossy, it affects the texture quality regardless of the number of bits of the texture coding. Usually, the lossless shape coding bits are very marginal to the texture coding bits. So, lossy shape-coding is useful when the shape-coding bits become significant, i.e., for low bit-rate applications (around 64 kbps). For lossy shape coding, accepted quality (ACQ) is a measure to test a binary alpha block (BAB)[4] if the number of opaque pixels are over a certain threshold. If the number is below the threshold, the BAB is treated as fully transparent.

Inter-frame coding is also applied to shape coding. Using an inter-frame scheme increases the encoder complexity due to motion search. It is, however, very efficient compared with intra-frame coding.

In summary, there are seven coding modes used in shape coding as illustrated in Table 2. Coding types 2, 3, and 4 are used for intra and inter-shape coding, whereas the other types are used for inter-shape coding only. It should be noted that coding type 4 (intraCAE) is used for inter and intra-shape coding. Coding of types 0 and 1 require no additional encoding, whereas coding of types 5 and 6 adopt interCAE coding.

Table 2. BAB coding modes.

Type number	Coding mode	Usage
0	MVDs==0 && no update	P- or B-VOPs
1	MVDs!=0 && no update	P- or B-VOPs
2	All transparent	All VOPs
3	All opaque	All VOPs
4	intraCAE	All VOPs
5	MVDs==0 && interCAE	P- or B-VOPs
6	MVDs!=0 && interCAE	P- or B-VOPs

B. MEMC

Motion estimation and motion compensation (MEMC) plays a very important part in shape encoding, especially when the video objects are static or moving slowly. The procedure for MEMC is as follows:

Step 1. Determine motion vector predictors for shape (MVPs).
Step 2. Estimate motion for shape.

3) CAE is the shape coding algorithm in MPEG-4.

4) A BAB is a 16×16 block corresponding to macroblock (MB) texture. The term 'MB' is used for texture, and BAB for shape.

Step 3. Compensate for motion.

In order to determine the MVPs, previous motion vectors for shape as well as previous motion vectors for texture can be utilized. In shape-only coding, only shape motion vectors are used for prediction, obviously. For the current BAB, the previous surrounding motion vectors are depicted in Fig. 5. Texture motion vectors are rounded to integers, even if the precision of the motion vectors is half-pel or quarter-pel. The MVPs is determined by the first valid motion vector that is encountered. If there is no valid motion vector, the MVPs is set to zero.

Once MVPs is determined, the motion estimation begins by calculating the motion compensation error of 16 × 16 pixels in the range of +/– 16 pixels around the MVPs along both the horizontal and vertical directions. The current reference software employs a spiral search in the same manner as texture motion estimation.

One notable thing, the shape motion estimation is almost equally as complex as the one used for texture. Hence, the optimization of texture-MEMC is solving only half the problem. How we can improve shape-MEMC is discussed in the next section.

Once the motion vector for shape is found, the motion compensation process prepares the reference BAB for interCAE coding, which will be addressed later in this section.

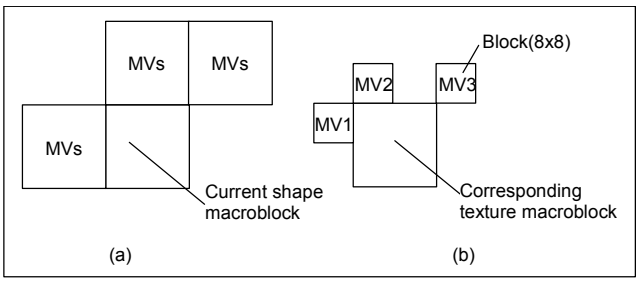

Fig. 5. Candidates for MVPs: (a) MV for shape, (b) MV for texture.

C. Subsampling

Subsampling, also called size conversion, was introduced to enable rate control for lossy shape coding. The down-sampled shape is often very similar to the original shape. So, subsampling has been used as a way to efficiently compress the shape by only encoding one-fourth, or one-sixteenth, the size of the shape. Subsampling, inherently, is better suited to lossy shape coding.

Subsampling has two distinct processes: down-sampling and up-sampling. However, it is incorrect to state that down-sampling is used for the encoder and up-sampling for the decoder. Both processes are required by the encoder and the decoder, since MEMC and inter/intra CAE coding require the original-sized BABs for reference. Accordingly, subsampling can be quite expensive in both the encoder and decoder.

D. Intra/Inter CAE

CAE encoding stems from a JBIG (Joint Binary Image Group) algorithm, which is basically an arithmetic coding with 10-pel contexts [18]. In fact, CAE extends JBIG by having inter-frame coding with 9-pel contexts.

IntraCAE uses the 10-pel contexts as shown in Fig. 6(a). The context information will generate a number between 0 and 1023 ($2^{10} - 1$), which will be used to calculate the probability used to encode the given symbol. InterCAE uses the 9-pel contexts as shown in Fig. 6(b). The inter contexts use 5 pels from the reference BAB and 4 from the current BAB. In regards to computational complexity, the difference between inter and intraCAE doesn't seem great. IntraCAE uses one more context than interCAE, whereas interCAE consumes more memory bandwidth to keep the reference BAB for coding.

Motion and texture coding use variable length coding (VLC), which is a variant of Huffman coding. The use of arithmetic coding in CAE may raise a doubt if it is more complex than the simple VLC. It is true that VLC has many advantages, but binary arithmetic coding doesn't seem too complex for software implementation. Through our runtime analysis of MPEG-4 reference software, we determined that arithmetic coding is not a critical issue.

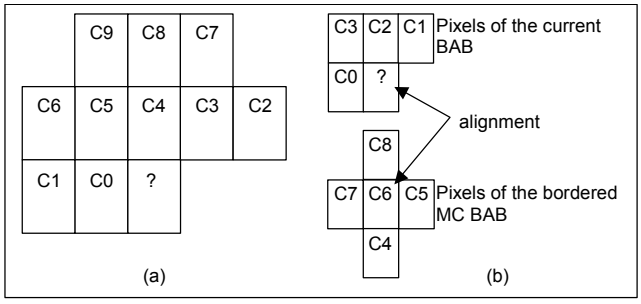

Fig. 6. CAE context information: (a) intra context, (b) inter context.

E. BAB Encoding Decisions

To maximize the coding efficiency, the scan order should be selected, i.e., horizontal scan or vertical scan. The encoder tries both and selects the one minimally coded with a selection flag (scan type). This technique is very useful in coding efficiency, but it increases the CAE complexity twofold. For PVOP coding, in particular, both interCAE and intraCAE are tried. Each CAE has two (horizontal and vertical) scan modes, which increases the CAE complexity by four times. So, this should be addressed clearly if added complexity is important for enhanced coding efficiency.

F. Scalable Shape

The scalability of the shape of an object may be independent

Table 3. Inter/Intra lossless binary shape coding results.

Test sequence	Intra-only total bits	Intra/Inter total bits	Inter bit reduction (%)
akiyo	371,151	170,109	45.8
coastguard	610,910	219,236	35.9
container	749,045	177,061	23.6
dancer1	277,110	220,968	79.7
dancer2	465,133	374,411	80.5
foreman	304,369	264,536	86.9
news	463,794	162,360	35.0
saxophone	590,051	481,935	81.7
singer	333,577	182,279	54.6
stefan	244,307	206,196	84.4
bream	408,103	297,156	72.8
child	612,851	401,639	65.5
cyclamen	1,210,536	608,446	50.3
fishandlogo	1,236,803	820,976	66.4
weather	297,731	158,882	53.4

from the scalability of its texture. This functionality is added in the second edition of MPEG-4. Since this functionality requires a subsampling process for scalability, the complexity of shape coding is likely to further increase. Although this functionality is important for some applications, we have left this out of our discussion. More details can be found in [19].

Before closing the discussion on binary-shape coding, we have provided the binary-shape coding performance comparison in Table 3. We have used 15 test sequences that are also used in MPEG-4 shape coding core experiments. The results shown in the table are the comparison of lossless inter/intra coders. It is clearly shown that a 23 to 86% bit reduction was achieved by using inter-shape coding. Using inter-frame coding on the shape of an object may imply that the total complexity added is more than that of intra-frame-only shape coding. Inter-frame shape coding is worse than intra-frame shape coding for error resilience. Yet, the substantial increase of coding efficiency is a strong point of inter-frame shape coding over intra-frame-only shape coding.

Tables 4 and 5 show the coding results of MPEG-4 OVC over two test sequences, while the first frame images are shown in Fig. 7. From the tables, we see that the shape coding used is the lossless inter-coding without subsampling and ACQ. It is very clear to see that the portion of shape coding bits is generally less than 10 percent. The lossy shape coding was worth trying when the quantization parameter (QP) of the texture was very high (more than 21 in some cases). But, for the other areas where medium or high-quality object-based video services are needed, lossless shape coding was still marginal to the total bit-stream size. An argument can be made that the smaller shaped bits would help to enhance the picture quality by adding more bits to the texture. This is clearly a dangerous argument if we consider the impact of lossy shape on texture, and also its impact on segmented video objects, since these will open 'holes' in the composed scene. If a part of the shape is distorted by lossy coding, the texture portion of the missing parts is removed from the encoding stage. Therefore, our conclusion is to not use lossy shape coding in order to maintain the quality of the texture pixels.

Table 4. MPEG-4 OVC results on akiyo sequence.

QP	Total bits	Shape bits	Shape portion (%)
1	16,189,711	170,109	1.1
5	14,561,999	170,109	1.2
9	12,078,647	170,109	1.4
13	9,979,983	170,109	1.7
17	8,299,727	170,109	2.0
21	6,934,015	170,109	2.5
29	5,450,959	170,109	3.1

Table 5. MPEG-4 OVC results on stefan sequence.

QP	Total bits	Shape bits	Shape portion (%)
1	7,345,871	206,196	2.8
5	5,703,231	206,196	3.6
9	4,002,543	206,196	5.2
13	2,812,615	206,196	7.3
17	2,081,287	206,196	9.9
21	1,658,607	206,196	12.4
29	1,263,823	206,196	16.3

(a) Akiyo sequence (352x288) (b) Stefan sequence (352x288)

Fig. 7. The first frame image of two test sequences.

6. Padding

The padding process is also a newly introduced concept with shape coding in OVC. The major reason for having this process is to efficiently handle boundary MBs and transparent MBs for motion and texture encoding and decoding. Hence, the padding is applied to motion and texture, but the actual padding technique for motion is different from that for texture. This is due to the fact that padding techniques for motion and texture have different purposes.

Figure 8 shows the padding process for motion. Horizontal and vertical padding are applied on the boundary MBs. Horizontal padding is used to put the transparent pixels with the horizontally closest pixel(s). If there are still transparent pixels left after the horizontal padding, vertical padding is applied in the same manner in the vertical direction.

The remaining transparent MBs are padded in two ways. If a transparent MB is adjacent to at least one boundary or opaque MB, the transparent MB is filled with the border pixels of the (padded) boundary or opaque MB. For the remaining transparent MBs, the value 128 is filled. By doing so, there is no transparent pixel without any padded value inside a VOP. Then, this padded VOP is used for MEMC.

Padding for texture is only applied to the boundary 8×8 blocks, and is different from padding for motion. MPEG-4 OVC is a DCT-based algorithm, which requires the transparent pixels to be filled before DCT. Without padding, the decoder may create unneeded edges between the transparent pixels and border pixels. Hence, the extrapolation algorithm based on non-transparent pixels is often used for padding. In the verification model (VM), the low pass extrapolation technique is used [15], [20]. Extrapolation padding is not necessary at the decoder side, which makes the padding for texture a non-normative part. On the contrary, padding for motion is normative; the decoder needs to use the same padding technique to produce accurate motion compensation.

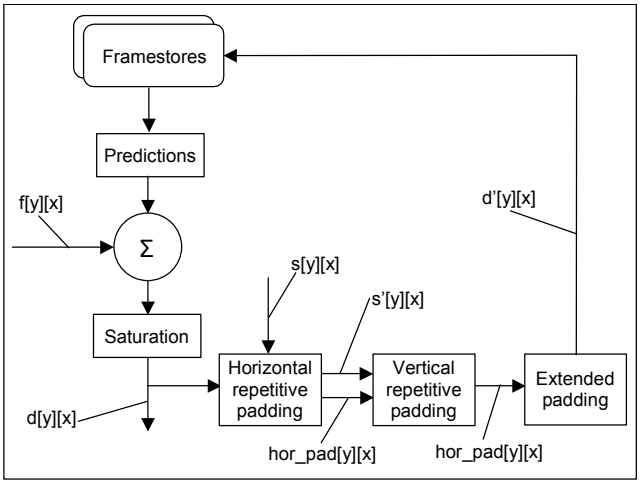

Fig. 8. Padding process for motion.

III. Efficient Encoding Conditions and Simulation Results

The overview of an MPEG-4 OVC encoder and decoder was given in the preceding section. Moreover, the detailed descriptions of IVOPF, binary-shape coding, and padding were each provided for further understanding of their structure and complexity. We also analyzed IVOPF and binary shape coding for further optimization at the encoder side, and proposed our choice of encoding conditions. Simulation results were also provided along with the description of efficient algorithms.

Padding for motion is not a light process, but the optimization on padding can be done only through pure optimization without hurting the original design concept. On the contrary, many different techniques using new design concepts may be applied to padding for texture. Yet, the impact may not be substantial due to its limited use (only for boundary 8×8 blocks). For these reasons, we leave aside the optimization issues on padding in this paper.

1. Test Conditions

The encoder and decoder software used throughout the paper is the MPEG-4 reference software 2001 edition (Microsoft version). The reference encoder and decoder contain the tools described in the VM [15]. The reference software, however, is not an optimized code. The reference software was built and tested on a Pentium 4 1.8 GHz PC with 512 MB RAM.

The complexity was measured using two methods, the run time and the run-time instructions of the reference and proposed encoders. The run time is obtained from the built-in run-time module in the reference software. The run time is averaged using three or four run times on the test PC. For the run-time instruction analysis, Intel's Vtune Performance Analyzer 6.1 was used.

The test sequences are listed in Table 6. The encoding conditions are as follows. The frame rate is 10 Hz. For simplicity, the encoding VOP pattern should be an 'IPPP…' type. For the QP of intra-VOP and predicted-VOP the value used should be 10, if not otherwise explicitly specified. No rate control is used. No additional optimization is applied when building the encoder or decoder.

2. IVOPF

The question that we raised on IVOPF in the previous section is whether the technique is really more efficient for compression. If so, the next question is how expensive is it. The

Table 6. Test sequences.

Test sequence	No. of frames	Format
akiyo	300	CIF
coastguard	300	CIF
container	300	CIF
dancer1	250	CIF
dancer2	250	CIF
foreman	300	CIF
news	300	CIF
saxophone	250	CIF
singer	250	CIF
stefan	300	CIF
bream	300	SIF
child	300	SIF
cyclamen	300	SIF
fishandlogo	300	SIF
weather	300	SIF

Table 7. New VOP formation modes.

Selection mode	Semantics
IVOPF	Find the minimum number of non-transparent MBs
8×8 Bnd (8×8)	Find the minimum number of 8x8 boundary blocks
Worst	Find the maximum number of non-transparent MBs
Trans/Bnd Min. (TBM)	Find the minimum number of transparent and boundary MBs
All Min (AM)	Find the minimum number of MBs (or try to find the minimal size of VOP)
Tightest rectangle (TR)	Do not use IVOPF at all

core idea of IVOPF is to find the VOP position which yields the minimal number of nontransparent MBs. The minimal number of nontransparent MBs is presumed to minimize the motion and texture coding, which will lead to further compression.

We extend this basic concept further to minimize the complexity while keeping the coding efficiency as high as IVOPF. In order to minimize the complexity, MEMC and padding are the most expensive processes to be minimized. So, we have designed the following additional conditions shown in Table 7 to choose the VOP position out of 64 candidate pixels in the control MB.

Table 8. Run time results of VOP formation modes (in ms).

Test sequence	IVOPF	8x8	Worst	TBM	AM	TR
akiyo	105.3	142.9	111.1	144.4	105.3	93.7
coastguard	169.3	170.7	169.9	170.6	169.3	168.8
container	122.5	123.1	122.7	123.4	122.7	121.8
dancer1	73.4	88.0	80.9	86.8	74.9	74.4
dancer2	120.4	142.3	127.9	141.5	121.8	118.1
foreman	118.4	127.9	120.1	128.0	118.5	116.3
news	141.2	144.8	141.9	145.5	137.2	138.2
saxophone	200.6	243.0	207.8	246.0	203.9	191.3
singer	62.0	79.1	69.4	78.2	63.8	62.1
stefan	65.5	77.1	72.7	76.9	67.1	67.3
bream	120.8	144.9	126.7	143.2	120.7	116.3
child	103.6	135.3	114.4	136.2	106.8	99.3
cyclamen	201.8	211.1	203.1	206.1	207.5	207.3
fishandlogo	127.2	155.3	138.9	155.8	131.7	125.7
weather	76.4	97.6	81.8	98.9	78.1	71.9
Average	120.6	138.9	126.0	138.8	121.9	118.2

In addition to the IVOPF from the table, we have added five different variations in choosing the VOP position. The 8×8 mode is to extend IVOPF from the 16×16 MB decision to 8×8 blocks, considering that DCT coding is done in 8×8 blocks. In order to avoid boundary texture coding, the best policy is to minimize the number of 8×8 boundary blocks. The Worst mode is a simple test mode to find out how bad it is when we choose the maximum number of nontransparent blocks, which is the inverse of IVOPF. In order to minimize MEMC and shape coding complexity, the minimal number of non-opaque blocks is presumed to be important, which becomes the TBM mode. If IVOPF is trying to minimize the number of nontransparent MBs, it may be worthwhile to minimize the number of MBs (all macroblock minimization mode). Finally, we decided not to do IVOPF at all (tightest rectangle mode), but to use the tightest rectangle (TR) (with multiples of MBs).

We have to confess at this point that all five new modes were not thought of from the beginning of the simulation. Rather, all the modes evolved within our simulation. The run time simulation results are shown in Table 8. Overall, none of the new selection modes except the TR mode was better than the IVOPF. This verified that the current selection algorithm of IVOPF is competitive in terms of complexity. However, the TR mode results turn things upside down. If the TR mode is used, then we do not use IVOPF from the beginning; hence no

additional complexity is added because of IVOPF. The simulation results in Table 8 show that the TR mode is slightly faster than IVOPF in most cases.

Before concluding that IVOPF is slightly more complex than the encoding without it, we provide the rate distortion performance of the MPEG-4 OVC reference software with and without IVOPF in Figs. 9 and 10. Only the results on two test sequences are shown, but the trend of the curve is the same for the rest of the sequences. From the figures, we can hardly find any proof that the encoder using IVOPF is coding more efficiently than the one without it. Not only that, the computational complexity increases!

Our simulation results are not contrary to the reported results in [16], since IVOPF's coding efficiency was mainly from SA-DCT. And SA-DCT is an optional mode, which is not cheaper than padding and DCT. Therefore, our finding is not to use IVOPF where the encoder complexity matters as much as the coding efficiency.

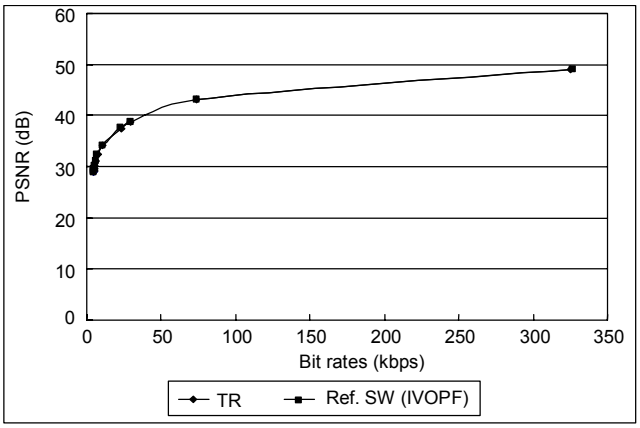

Fig. 9. Rate-distortion curve on akyio sequence.

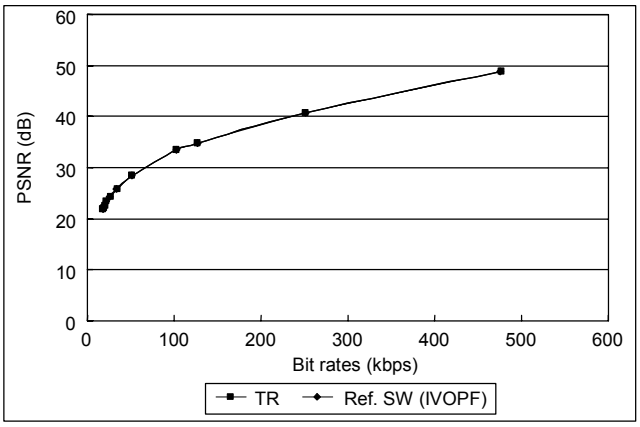

Fig. 10. Rate-distortion curve on fishandlogo sequence.

3. Binary Shape Coding

Among the tools that are explained about shape coding in the previous section, we have chosen three tools to test further optimization: subsampling, BAB encoding decisions, and MEMC for shape.

The reference software does not use subsampling by default. It can only be used by enabling the feature. We have tested the subsampling in lossless shape coding mode and obtained the simulation results shown in Table 9 and Table 10.

Table 9 is the run time results of the encoder and Table 10 is the run time results of the decoder. It should be noted that the decoding complexity is always much lower than the encoder complexity in the presence of MEMC. From the tables, the encoder with subsampling was slightly faster than the encoder without subsampling. On the contrary, the decoder with subsampling was slightly slower than the decoder without subsampling. Overall, the run time differences are very marginal. This is mainly because there are few cases where subsampling is really used in lossless shape coding. We did not test subsampling with lossy coding for the reasons mentioned in the preceding section. Conclusively, there is no added value by using subsampling.

As mentioned in the previous section on BAB encoding conditions, the inter-frame shape encoding tries four coding modes: intraCAE (horizontal scan), intraCAE (vertical scan), interCAE (horizontal scan), and interCAE (vertical scan), and chooses the best one to produce the minimal number of needed bits.

One way to reduce the complexity by half is to disable either the horizontal scan or vertical scan. In return, this would increase the number of coding bits. Our test results with disabled vertical scan are shown in Table 11. The results show that the reduced number of CAE coding trials does not help much. The complexity regarding CAE becomes lower, but this fact is not explicitly reflected in the run-time results. A complexity reduction was not apparent, which differed from our expectations. We believe this is due to the fact that the texture and motion encoding parts contribute heavily to the encoding time

Table 9. Encoding run time with/without subsampling (in ms).

	Subsampling ON	Subsampling OFF
akiyo	104	105
fishandlogo	127	134
stefan	65	68

Table 10. Decoding run time with/without subsampling (in ms).

	Subsampling ON	Subsampling OFF
akiyo	34	30
fishandlogo	88	88
stefan	37	36

Table 11. Encoding run time with vertical scan disabled (in ms).

	Ref. SW	Horizontal scan only
akiyo	105.3	103.8
fishandlogo	127.2	127.5
stefan	65.5	65.5

MEMC in MPEG-4 OVC is performed twice: once for shape and then again for texture. MEMC for shape is reasonably less complex than for texture since the block matching with shape is done on binary values. To calculate the similarity between the current BAB and reference BAB, we need only to count the matched pels, rather than the sum of absolute differences in texture.

There is a common process that is shared by MEMC in shape and texture: a basic block-matching algorithm. The basic block-matching algorithm that is currently under use in the reference software is the spiral search. From the predicted motion vector, the following candidate MBs are ordered and searched in a spiral way. It is easy to choose the closest pixel from the central point in X and Y directions. However, this spiral search is the core of the encoder complexity because of its many loops (+/– 16 pels wide).

There are many new and efficient techniques that substantially reduce the complexity of MEMC [10], [11]. These new techniques may be tried for shape coding. However, it should be remembered that the target of MEMC on shape is different from that of texture.

Our approach to optimizing the MEMC part in shape was not to do any kind of block matching. In other words, MVPs (from the texture motion) becomes the motion vector of the current BAB. Based on that, the encoder will decide the best coding mode among the seven coding modes. This implies that further coding gain from shape coding by the spiral search may be lost. The test results in Figs. 11 and 12 show that there is no noticeable difference in peak signal-to-noise ratio. In fact, it is very hard to notice any difference, whatsoever.

Table 12 shows the encoding run time with no additional shape-motion searches on 15 test sequences. The decoding run-time analysis was not presented because there is no noticeable difference in the decoder. For comparison, the results with reference software (the same as IVOPF in previous tables) and the TR mode are provided on the side. Non-motion search outperformed the reference software in all of the sequences. The encoder with non-motion search used IVOPF. Still, the run time results are better than using the TR mode. On average, the non-motion search encoder is 10.9 percent faster than the reference software. A maximum speedup of 23 percent was observed in the fishandlogo sequence. The overall speedup is quite substantial if we consider that no optimization was performed on the most expensive encoding part, MEMC on texture. The simulation results also prove that MEMC on shape is as expensive a process as MEMC on texture.

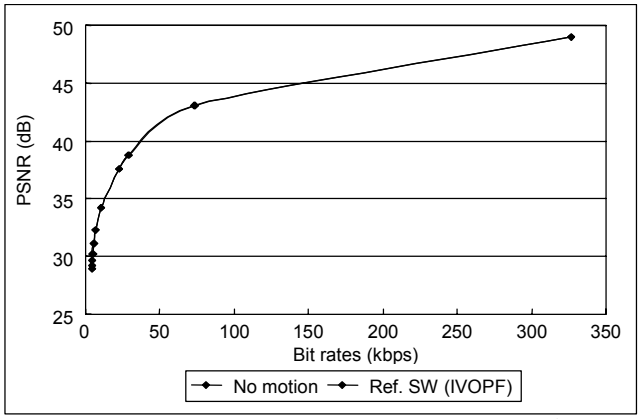

Fig. 11. Rate-distortion curve on akiyo sequence.[5)]

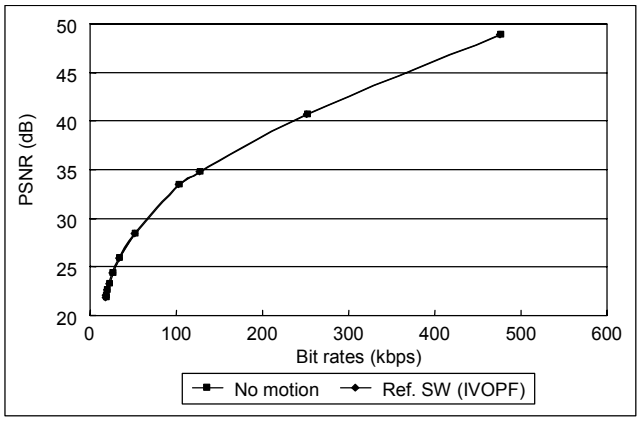

Fig. 12. Rate-distortion curve on fishandlogo sequence.

4. Proposed Encoding Conditions

We have reviewed and tested new ideas to further optimize the use of individual tools regarding VOP formation and shape coding. The summary of the proposed optimization ideas are listed in Table 13. The final choices of the proposed encoding conditions (PEC) from the table are either not to use IVOPF or not to use motion search in shape.

Figures 13 and 14 show the rate distortion performances of the encoder with PEC. Again, the difference between the reference encoder and the PEC encoder is hardly noticeable.

5) The term 'No Motion' in Figs 11 and 12 is used when no additional shape motion estimation and compensation is used. This would include a case when texture motion compensation may be in use.

Table 12. Run time results of no motion search (NMS) (in ms).

Test sequence	Ref. SW	TR	NMS
akiyo	105.3	93.7	98.1
coastguard	169.3	168.8	161.0
container	122.5	121.8	118.8
dancer1	73.4	74.4	62.4
dancer2	120.4	118.1	101.9
foreman	118.4	116.3	107.6
news	141.2	138.2	137.1
saxophone	200.6	191.3	177.5
singer	62.0	62.1	54.5
stefan	65.5	67.3	55.4
bream	120.8	116.3	104.8
child	103.6	99.3	86.8
cyclamen	201.8	207.3	178.4
fishandlogo	127.2	125.7	97.9
weather	76.4	71.9	69.4
Average	120.6	118.2	107.4

Table 13. Summary of proposed efficient algorithms.

Tool	Proposed optimization	Noticeable speedup
Subsampling	No subsampling	NO
BAB encoding decisions	Vertical scan disabled	NO
IVOPF	Do not use IVOPF	YES
MEMC for shape	No motion search	YES

This implies that the combination of the proposed methods in PEC does not affect the overall compression performance of the encoder.

Run time results on PEC are given in Table 14. The results with PEC are the best of all the compared tools. The average speedup is higher than the NMS results: 13.2 percent. The maximum speedup is 25.7 percent in fishandlogo. Table 15 shows other interesting run-time results using the shape-only coding mode. In the shape-only coding mode, only the shape information is encoded and decoded. Hence, it is possible to estimate how much complexity reduction was made on only the shape coding part. From the table, the average speedup is more than half. This means that the MEMC part in shape coding consumes more than 50 percent of shape encoding, and that the PEC encoder completely removed the complexity without losing coding efficiency.

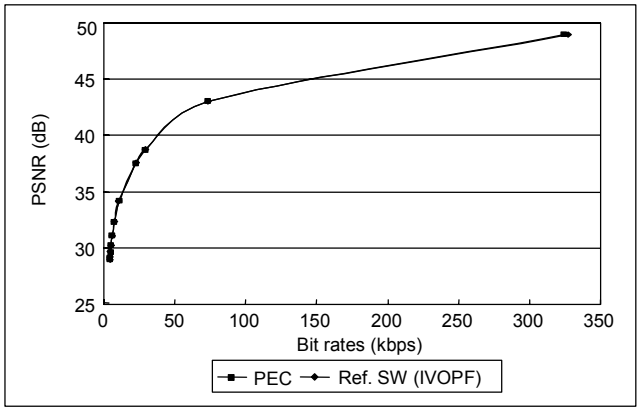

Fig. 13. Rate-distortion curve on akiyo sequence.

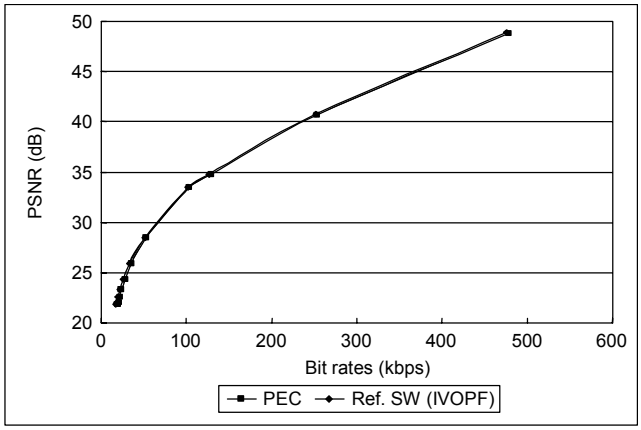

Fig. 14. Rate-distortion curve on fishandlogo sequence.

Table 14. Run time results of PEC (in ms).

Test sequence	Ref. SW	TR	NMS	PEC
akiyo	105.3	93.7	98.1	88.7
coastguard	169.3	168.8	161.0	161.6
container	122.5	121.8	118.8	119.0
dancer1	73.4	74.4	62.4	62.1
dancer2	120.4	118.1	101.9	99.0
foreman	118.4	116.3	107.6	105.9
news	141.2	138.2	137.1	134.6
saxophone	200.6	191.3	177.5	167.4
singer	62.0	62.1	54.5	53.7
stefan	65.5	67.3	55.4	55.9
bream	120.8	116.3	104.8	101.2
child	103.6	99.3	86.8	81.0
cyclamen	201.8	207.3	178.4	179.6
fishandlogo	127.2	125.7	97.9	94.4
weather	76.4	71.9	69.4	64.5
Average	120.6	118.2	107.4	104.6

Table 15. Run time results of PEC on shape-only mode (in ms).

Test sequence	Ref. SW	PEC	Speedup (%)
akiyo	28.6	9.6	66.3
coastguard	19.3	11.6	40.0
container	13.0	9.5	27.3
dancer1	24.5	9.9	59.7
dancer2	38.2	13.5	64.7
foreman	28.2	13.4	52.5
news	15.9	10.3	34.8
saxophone	53.5	16.2	69.7
singer	22.8	9.8	56.9
stefan	22.2	9.1	59.1
bream	33.0	11.6	64.8
child	39.5	12.9	67.3
cyclamen	44.7	20.2	54.7
fishandlogo	54.2	16.6	69.5
weather	22.7	8.4	62.8
Average	30.7	12.2	56.7

Table 16. Comparison of the number of instructions (in ms).

Test sequence	Ref. SW	PEC	Reduction (%)
akiyo	18753	12976	30.8
fishandlogo	19215	15701	18.3

We have presented only the run-time analysis thus far. For PEC, the analysis of the number of run-time instructions is given in Table 16. By disabling IVOPF and motion search for shape, the total number of instructions is reduced quite substantially. The reduction of instructions is not directly comparable to the run time analysis, but the instructional analysis confirms that the PEC encoder is substantially less complex than the reference encoder.

IV. Conclusion

We presented an efficient encoding method that reduces the shape-coding complexity by more than half while preserving the overall coding efficiency of motion, shape, and texture coding. Notable findings conclude that intelligent VOP formation really does not help in low complexity, that the MEMC part in shape encoding is the most computationally expensive, and that the improvement of MEMC on shape alone can contribute 10 to 20 percent in speedup of the entire encoder.

The combination of the proposed encoding conditions and the efficient MEMC tools for texture seems to be the next step after this research in the near future.

Acknowledgments

I would like to thank Sunyoung Lee, Seokju Moon, and Minhun Kim for their help in software simulation. I would also like to thank Mr. Se Hoon Son for providing some of the test sequences with shape data. And, I would like to express my sincere gratitude to the reviewers of this paper. Thanks to the reviewers, I was able to improve the quality of this paper.

References

[1] F. Pereira and T. Ebrahimi, eds., *The MPEG-4 Book*, Prentice Hall, Upper Saddle River, NJ, 2002.

[2] ISO/IEC 14496-2:2000, *Coding of Audio-Visual Objects-Part2: Visual*, 2000.

[3] Y. Ozer, "Will MPEG-4 Fly?" *PC Magazine*, Apr. 3, 2001.

[4] Microsoft Windows Media Technologies homepage, http://www.microsoft.com/korea/windows/windowsmedia/software/Playerv7.asp.

[5] Realnetworks homepage, http://www.realnetworks.com/.

[6] MPEG homepage, http://mpeg.telecomitalialab.com/.

[7] W. Zheng, I. Ahmad, and M.L. Liou, "Benchmark the Software Based MPEG-4 Video Codec," *8th Int'l IEEE Conf. Electronics, Circuits Syst.*, vol. 1, 2001, pp. 289-292.

[8] H.-C. Chang, Y.-C. Wang, M.-Y. Hsu, and L.-G. Chen, "Efficient Algorithms and Architectures for MPEG-4 Object-Based Video Coding," *IEEE Workshop Signal Processing Syst.*, 2000, pp. 13-22.

[9] S.-M. Kim, J.-H. Park, S.-M. Park, B.-T. Koo, K.-S. Shin, K.-B. Suh, I.-K. Kim, N.-W. Eum, and K.-S. Kim, "Hardware-Software Implementation of MPEG-4 Video Codec," *ETRI J.*, vol. 25, no. 6, Dec. 2003, pp. 489-502.

[10] MPEG-4 Optimization Model 2.0, ISO/IEC JTC1/SC29/WG11 N3675, La Boule, Oct. 2000.

[11] F. Dufaux and F. Moscheni, "Motion Estimation Techniques for Digital TV: A Review and a New Contribution," *Proc. IEEE*, vol. 83, June 1995, pp. 858-876.

[12] MPEG-4 Requirements version 17, ISO/IEC JTC1/SC29/WG11 N4319, Sydney, July 2001.

[13] F. Casalino, G.D. Cagno, and R. Luca, "MPEG-4 Video Decoder Optimization," *IEEE Int'l Conf. Multimedia Computing and Syst.*, vol. 1, July 1999, pp. 363-368.

[14] ISO/IEC 14496-5:2000, *Coding of Audio-Visual Objects-Part5: Reference Software*, 2000.

[15] MPEG-4 Video Verification Model 15.0, ISO/IEC JTC1/SC29/WG11 N3093, Maui, Dec. 1999.

[16] J.-H. Moon, G.-H. Park, S.-M. Chun, and S.-R. Choi, "Shape-Adaptive Region Partitioning Method for Shape-Assisted Block-Based Texture Coding," *IEEE Trans. Circuits Syst. Video Technol.*,

vol. 7, Feb. 1997, pp. 240-246.
[17] J. Ostermann, E.S. Jang, J.-S. Shin, and T. Chen, "Coding of Arbitrarily Shaped Video Objects in MPEG-4," *IEEE Int'l Conf. Image Processing*, Santa Barbara, CA, 1997.
[18] N. Brady, "MPEG-4 Standardized Methods for the Compression of Arbitrarily Shaped Video Objects," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, no. 8, Dec. 1999, pp. 1170-1189.
[19] S.-H. Son, E.S. Jang, S.-H. Lee, D.-S. Cho, J.-S. Shin, and Y.-S. Seo, "Scan Interleaving Based Scalable Binary Shape Coding," *Signal Processing: Image Communication* 15, 2000, pp. 619-629.
[20] A. Kaup, "Object-Based Texture Coding of Moving Video in MPEG-4," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, Feb. 1999, pp. 5-15.

Euee Seon Jang was born in Chonju, Korea in 1968. He obtained the BS in computer engineering at Chonbuk National University, Chonju, Korea in 1991. Then, he completed the MSEE and PhD in electrical and computer engineering at the State University of New York in Buffalo, NY, in 1994 and in 1996. His PhD topic was robust image communications. He worked as a Research Associate in the US Army Research Lab., Adelphi, MD, in 1995. After receiving the PhD, he joined Samsung Advanced Institute of Technology, where he spent six years in research and development of various MPEG-4 visual technologies. He was Project Editor of MPEG-4 visual standard in the MPEG committee (ISO/IEC JTC1/SC29/WG11) from 1997 to 2000. He also served as Chair of Synthetic Natural Hybrid Coding (SNHC) Subgroup in MPEG from 1999 to 2002. He has co-invented many MPEG-4 technologies: shape coding, 3D mesh compression, and interpolator compression. He is also a forefather of MPEG-4 Animation Framework Extension (AFX) standardization. Since 2002, he has served as Assistant Professor at the College of Information and Communications at Hanyang University, Seoul, Korea. He has authored more than 70 MPEG contribution papers, 12 journal or conference papers, 35 pending or patented patents, and 2 book chapters. His current research interests include computer graphics, animation, image and video coding, and lossless multimedia data compression. Dr. Jang is also a member of ACM. He has served as technical committee member in ICME. He has received two ISO/IEC Certificates of Appreciation for the contribution in MPEG-4 development in 1999 and 2000.