MPEG-4 Very Low Bit-rate Video Compression Using Sprite Coding

Kumi Jinzenji †*, *Hiroshi Watanabe**, *Shigeki Okada*†, *Naoki Kobayashi*†
†NTT Cyber Space Labs.
1-1 HikarinookaYokosuka-Shi, Kanagawa 238-0846, JAPAN
*Waseda University, Global Information and Telecommunication Institute
1-3-10 Nishi-waseda, Shinjuku-ku, Tokyo 169-0051, JAPAN

ABSTRACT

This paper focuses on the "sprite coding" that supports the MPEG-4 Version 1 Main profile in order to transfer "near VHS quality video " across narrow-band transmission links such as the Internet. Automatic VOP (Video Object Plane) generation technologies are being studied as one of the most important issues of MPEG-4 object coding. This paper proposes a two-layer VOP generation scheme with some core algorithms such as GME (Global Motion Estimation), foreground moving object extraction, and background sprite generation. This paper also describes a shape information reduction method for the foreground object. The foreground object is object-coded in the , while the background sprite is coded using sprite coding in MPEG-4. We call this coding scheme "sprite mode"; MPEG-4 simple profile coding is called "normal mode". Experiments are conducted on VOP generation and video coding with MPEG-4. We compare sprite mode to normal mode. The coding efficiency of sprite mode is several times higher than that of normal mode at the same objective image quality if the foreground ratio is within 10-15%. Given the target of very low bitrate (128kbps, 64kbp) rate coding, sprite mode achieved almost the same SNR but more than twice the frame rate compared to normal mode.

1. INTRODUCTION

We are researching and developing very low bitrate coding schemes applicable to Internet applications [1]. Conventional coding schemes fail to maximize the frame rate or image quality at very low bitrate (i.e. 64kbps). This reflects the weakness of MC+DCT and an alternative compression scheme was seen as necessary. This paper focuses on the "sprite coding" (the coding tool that supports the MPEG-4 Main profile) and proposes a way to automatically split a moving image into the sprite and the foreground object. Sprite coding expresses the area that occupies the same position across several frames as one plane (sprite) and can be described by parametric conversion (global motion: GM). The corresponding area in each frame can be regenerated from the sprite. If this area is large, most of the moving image can be expressed using one sprite (=static image), which leads to very high compression efficiency.

For the automatic generation of sprites, Irani [2] et al. use the top-down approach to calculate the global motion by aligning the entire image. Wang [3] et al. generates multiple sprites from multiple global motions determined by the clustering of local motion. Lee [4] et al. report that manual sprite generation and achieved dramatic compression efficiency while keeping the same subjective image quality. However, none of these papers discussed the area outside the sprite (= foreground area).

The answer to how to express and code the non-background area (= foreground area) is MPEG-4 object coding. If there are multiple objects within a frame, object coding is executed for each object. This means that each object must keep its correspondence across several frames. One of the conventional object extraction schemes (semi-automatic scheme) proposed by Choi [5] et al. clearly solves the correspondence problem of each object, but its processing cost is not desirable. Some thechnoloies [6][7] extract and trace single objects. However, none of them discuss the correspondence of multiple objects across several frames.

First of all, this paper defines the VOP (Video Object Plane) which consists of two layers, namely the foreground object and the background sprite. Each frame is automatically split into foreground VOP and background VOP. Because this two layer VOP generation algorithm regards the whole moving area outside the sprite as the one foreground, the problem of object correspondence is sidestepped. The two layer VOP generation algorithm consists of a GM calculation method specialized for sprite generation, a high-quality background sprite generation method, and a foreground object extraction method. The foreground object extraction method is based on the difference between the original image and background image extracted by GM, and is robust against GM displacement. There is a problem in that an automatically generated foreground object has complicated contours and includes much noise, so its shape is very difficult to predict accurately. As a result, intra coding increases which increases code size. To sidestep this problem, we propose a method that approximates object shape by using macro blocks. This method can reduce the shape code size by about 90%. In this paper, we apply the proposed algorithm to generate the foreground object and the background sprite, then compress by object coding and sprite coding of MPEG-4 Main profile, respectively. We compare them to the results of MPEG-4 simple profile without VOP structure.

2. AUTOMATIC TWO-LAYER VOP GENERATION

2.1 Two-layer Video Object Model

Fig.1 overviews the coding/decoding model proposed in this paper. On coding side, the image is first split into two layer video objects, namely the foreground object and the background sprite. The background sprite is the background reflecting the camera motion. The foreground object is any moving area not belonging to the background, and all these areas are treated as one foreground. For example in "soccer" image, the players, the referees and the ball are treated as one foreground object. The foreground object and the background sprite are independent video objects, and the

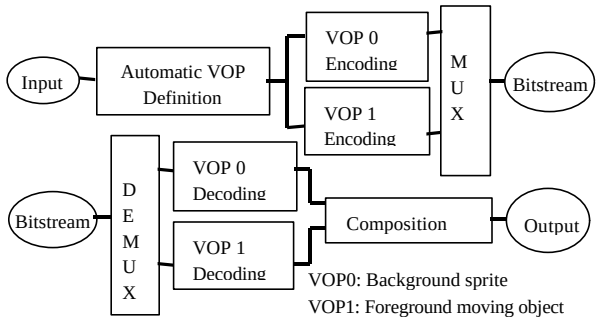

Fig.1 Two layer VOP Model.

former is converted to free shape code; the latter is subjected to sprite coding. These isolated bit streams are multiplexed and sent as one. At the receiving side, the bit stream is demultiplexed and each video object is decoded, superimposed and displayed.

Automatic two layer VOP generation algorithm consists of 3 main parts: GM calculation, background sprite generation, and foreground object extraction. At first, we calculate the motion vectors between adjacent frames, then using these motion vectors, calculate GM from the base frame. Using this GM, we deform the original image and map it to generate a temporary sprite without moving objects. We then calculate the difference between the temporary sprite and the original image, and use the result to generate the candidate foreground image and the background image. The candidate foreground image is approximated using macro blocks to yield the final foreground image. The background image is used to generate the background sprite. Details are given in the following chapter and reference [1].

2.2 GM Calculation

For sprite generation, GM should reflect only camera operation. This is the most significant difference from the typical GME algorithm which is tries to minimize the error between the original image and the predicted image with GM. To avoid non-camera movement (outlier), two techniques are proposed for GME: use of motion vector distribution in the feature space and cluster candidate selection.

Camera motion can be described using the Hermart transform (four parameters affine) as:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} a' & b \\ -b & a' \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} c \\ d \end{pmatrix} \qquad (1)$$

$$a = a' + 1$$

where *(u,v)* is the motion vector calculated in each macro-block, *(x,y)* is the position of the pixel, and *{a,a',b,c,d}* is the set of GM parameters to be calculated. *a* and *a'* are scaling parameters, *b* denotes rotation, *c* and *d* denote translation.

First, the motion vector for each macro-block is calculated using the block-matching algorithm. Partial derivatives (see equation (2) and (3)) of the motion vectors are calculated for each macro-block.

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \equiv a' \qquad (2)$$

$$\frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x} \equiv b \qquad (3)$$

Each partial derivative creates a significant cluster on a line written by equations (2) and (3) in each feature space. Here, all blocks with smooth intensity gradation are removed from the target blocks in the GME process, because such motion vectors are not accurate and often concentrate around zero. The centroid of each cluster yields scaling and rotation parameters. In this way, *a'* and *b* are detected.

Equation (1) can be transformed into

$$\begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix} - \begin{pmatrix} a' & b \\ -b & a' \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}. \qquad (4)$$

Translation vectors in each macro-block are calculated using Equation (4). The translation vectors create several clusters in the feature space. These centroids of the clusters are the candidates of translation parameters *c* and *d*. Here, suppose that the translation vector $(c_{outlier}, d_{outlier})$ reflects outlier motion, while (c_{camera}, d_{camera}) is camera motion. The absolute difference at each pixel between the original and predicted images using a certain translation parameter candidate is calculated. The number of pixels whose value exceeds a certain threshold Th is calculated. If p_{ideal} is the pixel number for ideal camera motion, while $p_{outlier}$ is for outlier motion. p_{ideal} is always larger than any $p_{outlier}$. This principle can be used for the cluster candidate selection needed for translation parameter detection. First, an absolute difference image between the original image and the predicted image is calculated for each candidate to examine. Then, in each candidate, the number of pixels in the absolute difference image less than a certain threshold Th is counted. The centroid having the largest number of pixels is selected as the translation parameter. Here, threshold Th is experimentally determined. These detected parameters *{a,a',b,c,d}* are then transformed into GM from a preset base frame.

2.3 BackGround Sprite Generation

To generate a high quality sprite, this paper takes advantage of a conventional temporal median method, the overwriting method. Two kinds of sprite are generated: provisional sprite and final sprite.

The sprite is the panorama image generated by aligning images so that their patterns link to each other. Using the GM calculated from the base frame in the above section, we transform the shape of each image and map them on the lattice points of the base coordinate system. We can see that multiple pixels overlap each other. We calculate their median and take it as the value of that coordinate. If the pixels of the moving object is less than half of pixels overlapping on that coordinate the pixels reflecting the moving object will be excluded by calculating the median. This generates the sprite without moving area. However, the sprite generated by the temporal median method tends to be a little fuzzy. We therefore, use this median-value-based sprite as a provisional sprite to extract foreground. The overwrite method pastes the pixels as they are on the base frame and so generates a high quality sprite. However, it has a problem in that the foreground of the top frame and the foreground object at the edge of each frame remain. So after extracting the moving object, we treat the remaining part of the image as the background image and overwrite the pixels on the base coordinates one by one to generate the final high quality sprite.

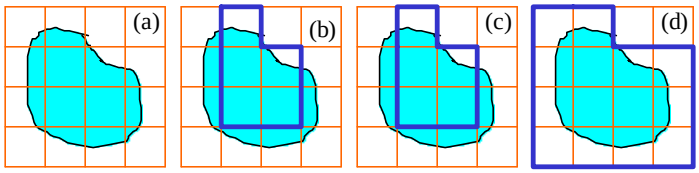

Fig.2 Macro-block approximation. (a)original contours; (b)most lossy mode; (c) first macro-block approximation; (d)second macro-block approximation and final contours.

2.4 Foreground Object Extraction

At first, we calculate the difference image between the original image and the image extracted using the temporary sprite. We binarize this difference image by thresholding and split it into a foreground candidate image and a background image. The foreground candidate image is processed by the macro block approximation method (described in the next section), to yield the final foreground image.

Because the automatically generated foreground object has complicated contours, many acnodes, its shape is very difficult to predict accurately. This triggers intra coding which increases coding volume. To sidestep this problem, this paper proposes a method that approximates object shape by using macro blocks.

MPEG-4 object shape coding has two modes: lossless and lossy. The most rough shape expression in lossy mode is approximated by macro blocks (Fig.2(b)). To be more specific, if more than half of the macro block pixels include foreground information, the whole macro block is regarded as foreground. Conversely, if less than half of the macro block pixels include foreground information, the whole macro block is regarded as background. One problem is that the foreground object is eroded and obstructs the view. Therefore, this paper proposes two phase macro block approximation. At first, the macro block (contains more than threshold Th1 foreground pixels) is regarded as foreground (Fig.2(c)). All other macro blocks are regarded as background. We then focus on the background macro block adjacent to the foreground macro block from the first macro block approximation phase. If more than threshold Th2 (Th2<Th1) of foreground pixels are included in that macro block, we regard it as foreground (Fig.2(d)).

The optimum threshold values, Th1 and Th2, depend on the image. For example, an image with a complicated background is easily effected by false GM, and foreground candidate areas calculated by the difference naturally increase. To automatically specify Th1 and Th2, we propose the algorithm shown in Fig. 3. As it is assumed that sprite coding does not yield a code size reduction if the foreground exceeds some ratio, we use the foreground ratio as the control key. Th1 and Th2 are initially set at the percentage corresponding to 1 pixel, and then incremented until the foreground ratio falls under the heuristic threshold Rmax.

We executed the macro block approximation for the shape information contained in the standard image “Stefan” using a segmentation mask, and found that the shape code size was reduced by about 90%. Another merit of macro block approximation is the lower processing cost in software decoder implementation. As shape information encoding and padding are not needed, this method is suitable for applications that require real time encoding/decoding.

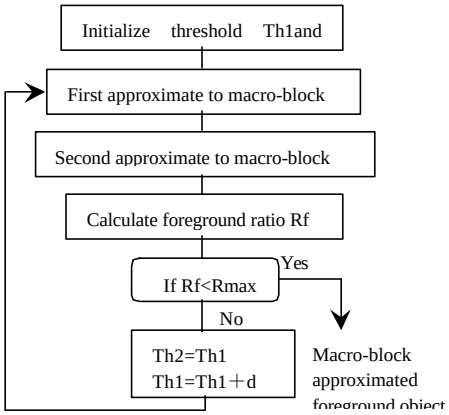

Fig.3 Flow chart of macro-block approximation.

3. MPEG-4 Coding Experiments

In the previous section, we applied the two layer VOP generation algorithm to moving images and generated video objects. We compressed the foreground objects by object coding and compress edthe background sprites by sprite coding. We then compress the original moving image using the MPEG-4 Simple profile. For convenience, the former (latter) is referred to as sprite (normal) mode. The code size with sprite mode is the sum of the foreground object code and the background sprite code.

3.1 Relationship between Foreground Ratio and Code size

It is obvious that the superiority of the sprite mode over the normal mode depends, in part, on the ratio of foreground to background object.

We examined 5 SIF images (150 frames each) that included camera operation as shown below, and compared the coding efficiency provided by both modes by fixing the quantization parameter value (QP=12) and changing the foreground ratio from 5 to 40 %. Applying the same quantization parameter to both modes should yield the same image quality.

1. Horserace: pan & several horses
2. Athlete: pan & several athletes
3. Soccer : pan, zoom & several players, a ball
4. Stefan: pan, zoom & a player, a ball
5. Board: pan & active skater border

Fig. 4 shows the foreground ratio and code size ratio with both coding modes. If the foreground ratio is within 10 - 15%, the coding efficiency of sprite mode is more than 2 times higher than that of normal mode. When the foreground ratio is about 40%, the coding efficiency of both modes is basically the same. At foreground ratios beyond 40%, normal mode achieves higher coding efficiency than sprite mode.

As the proposed two layer VOP generation algorithm selects foreground according to the foreground ratio, a part of the foreground may be excluded from this range and not extracted. In this case, the foreground is coded as it is, and the information volume decreases proportionally. This is noticeable in “Stefan”, which includes a large audience area. The other images have more even backgrounds. Even if a part of a moving area is not extracted

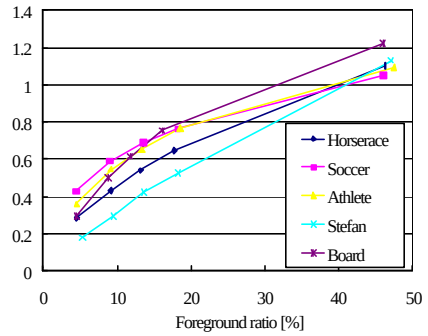

Fig.4 Foreground ratio and coding efficiency of sprite mode compared to normal mode.

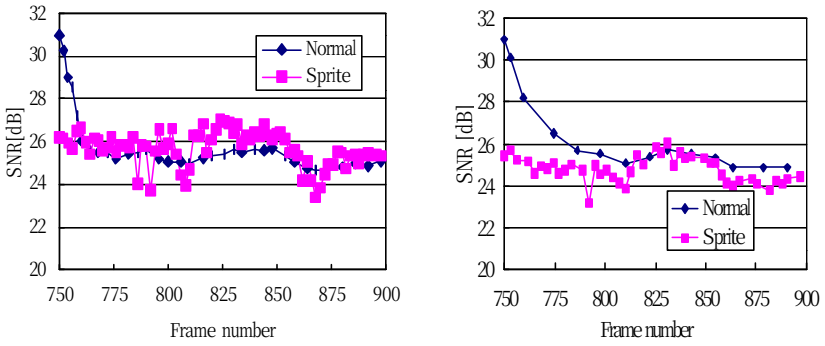

Fig.5 Objective coded image quality under fixed bit-rate. Left: 128kbps and right: 64kbps.

as foreground, its loss does not impact image quality. We found that if the background is perfectly static and the foreground ratio is within 10 - 15 %, we can more than double the coding efficiency by using sprite mode.

3.2 Image Quality and Frame Rate Comparison at Fixed Bit Rate

We executed normal/sprite mode coding experiments on the "Horserace" image (bitrate=128 kbps with target frame rate=15 fps, and bitrate=64 kbps with target frame rate=10 fps). Normal mode coding control followed VM Ver.15 [8]. The foreground ratio of sprite mode was set at 15% to reflect the understanding gained in the previous section.

Fig. 5 compares image frame SN ratios at 128 kbps and 64 kbps. At 128 kbps, normal mode yielded the average SNR of 25.85 dB at the frame rate of 6.80 fps, while sprite mode achieved the average SNR of 25.69 dB at the frame rate of 15 fps. Similarly, at 64 kbps, normal mode achieved the average SNR of 26.31 dB at the frame rate of 2.8 fps, while sprite mode achieved the average SNR of 24.79 dB at the frame rate of 8.60 fps. At 128 kbps, the sprite mode offers more than double the frame rate. The tests showed that while the sprite mode yields slightly inferior SNR objective image quality, it sometimes yields superior subjective image quality (please note the result at 128 kbps).

4. Summary

We focused on "sprite coding" (the coding tool that supports MPEG-4 Main profile [8]) to develop a "near VHS on the Internet" coding method. We proposed an automatic video object generation algorithm. We also applied MPEG-4 coding (sprite mode) to the video objects generated by the proposed algorithm, and compared it performance to that of the MPEG-4 simple profile (normal mode). We considered the coding efficiency. Tests proved that at the low bitrates of 128 kbps and 64 kbps, the sprite mode yields almost the same SNR as the normal mode while achieving 2-3 times higher frame rates with the condition foreground ratio less than 10-15% of the whole image. However, video sequences do not always satisfy these conditions. How to judge which shot should undergo sprite coding, a seamless decoding method, and how to allocate the code to each video object are future tasks.

Acknowledgement

We are deeply grateful to Ms. Noriko Yonehara of NTT Software Corporation who was so helpful during the computer simulations.

REFERENCES

[1] K. Jinzenji, S. Okada, H. Watanabe, N. Kobayashi, "Automatic Two-layer Video Object Plane Generation Scheme And Its Application to MPEG-4 Video Coding," IEEE ISCAS2000, pp.606-609, May 2000.

[2] M. Irani, S. Hsu, and P. Anandan, "Video Compression Using Mosaic Representation," Signal Processing: Image Communication, Vol. 7, pp. 529-552, 1995.

[3] J. Wang and E. Adelsen, "Representing Moving Images with Layers," IEEE Trans. on IP, Vol. 3, No. 5, pp.v625-638, September 1994.

[4] M. Lee, W. Chen, C. Lin, C. Gu, T. Markoc, S. Zabinsky, R. Szeliski, "A Layered Video Object Coding System Using Sprite and Affine Motion Model," IEEE Trans. on CSVT, Vol. 7, No. 1, February 1997.

[5] J. G. Choi, S. Lee, S. Kim, "Spatio-Temporal Video Segmentation Using a Joint Similarity Measure," IEEE Trans. on CSVT, Vol. 7, No. 2, April 1997.

[6] Neri, S. Colonnese, G. Russo, "Automatic Moving Objects and Background Segmentation by Means of Higher Order Statistics," IEEE ISCAS'97, June 1997.

[7] T.Meier, K.N. Ngan, "Automatic Segmentation of Moving Objects for Video Object Plane Generation," IEEE Trans. on CSVT, Vol. 8, No. 5, September 1998.

[8] "MPEG-4 Video Verification Model Version 15.0," ISO/IECJTC1/SC29/WG11 MPEG98/N3093.