

A Real-time Realization of MPEG-4 Video Decoder

Lap-Pui CHAU*, Nam LING⁺, Gunnar HOVDEN⁺, Hui LAN*,
Hon-Cheong NG*, Keng-Pang LIM*

* Centre for Signal Processing, Nanyang Technological University, Singapore.

⁺ Department of Computer Engineering, Santa Clara University, California, USA.

Abstract

In this paper, several techniques for real-time realization of an MPEG-4 video decoder software is presented. The structure of the decoder is also discussed. Our methods improve the processing of three computationally intensive modules, context-based arithmetic decoding, padding, and variable length decoding. The decoding speed of our developed software is a few times faster than the existing MPEG-4 reference software.

1. Introduction

MPEG-4 is a new ISO/IEC standard currently being developed by the Moving Pictures Expert Group (MPEG). The standard provides a set of tools targeted for object-based coding and decoding of natural as well as synthetic video or audio objects. Applications of MPEG-4 include Internet communication, video conferencing, mobile and wireless multimedia, just to name a few. It attempts to support a wide range of bitrates from 2 kbps to several Mbps. Each application dictates a subset of tools, which may vary from one application to another.

The video portion of the MPEG-4 standard [1] aims to support the coding of arbitrarily shaped video objects, which enables new applications based on object manipulation and content-based accessing. The shape of a video object is generated by segmentation techniques, which are not covered by the MPEG-4 standard.

The committee draft (CD) of MPEG-4 version 1 [2] includes the following coding tools for natural video sequences: I-VOP (video object plane), P-VOP and B-VOP coding, arbitrary shape coding, temporal scalability, spatial scalability, error resilience, static sprites, and texture coding. Compared to existing compression standards, MPEG-4 requires much more computational power. Given the increased performance and improved architecture for general purpose CPUs, it is possible to implement many MPEG-4 decoding tools using a standard CPU. However, efficient software structures and algorithms are important to ensure that decoding meets real-time requirements. These are also

important so that faster speed, higher resolution, and more features can be supported.

In this paper, an efficient realization of a subset of MPEG-4 video decoding tools is presented. The focus of this paper is on real-time decoding. The content of this paper focuses on the analysis of MPEG-4 video decoding tools. The tools for variable length decoding, motion padding, and context-based arithmetic decoding [3] are analyzed here. These are usually the most computationally intensive modules in MPEG-4 video decoders.

The paper is structured as follows. An overview of our software is discussed in Section 2. Sections 3 to 5 discuss the details of our techniques used to implement context-based arithmetic decoding, motion padding, and variable length decoding. The last section summarizes the major notions introduced in this paper.

2. Decoder Structure

Our decoder software is designed as a converter between the bitstream and the video display device. Our decoder consists of three main decoding engines, the shape decoder, the texture decoder, and the motion decoder. They decode the shape, the texture, and the motion of the object respectively. These engines exchange information among themselves to accomplish the task. Our decoder decodes each object on a macroblock (16x16 pixels) by macroblock basis, until a VOP is formed. Our strategies used to enhance the speed of decoding performance include

- The use of efficient and simple data structures with minimum pointers
- The reduction and improvement in input/output operations and memory allocations
- The use of fixed point arithmetic
- The use of efficient algorithms for computationally intensive modules (e.g. padding, context-based arithmetic decoding, table searching, and inverse discrete cosine transform)

Figure 1 shows the general structure of our software (excluding the shaded modules). The main modules are:

- yuvshow.c, which converts YUV signals to RGB format for the display device
- visual.c, which is used to integrate all the visual signals in natural image decoding
- video.c, which decodes natural image data and parses the bitstream to each sub-module (macroblock level decoding and VOP padding are also implemented in this module)
- vlc.c, which performs fast binary tree search for variable length decoding table
- vop.c, which defines data structure and initializes VOP parameters
- padding.c and mot_padding.c, which perform rectangular shape and arbitrary shape padding
- block.c, idct.c and motion.c, which perform block level decoding, inverse DCT and motion vector decoding
- cad.c, which decodes shape information by using context-based arithmetic decoding

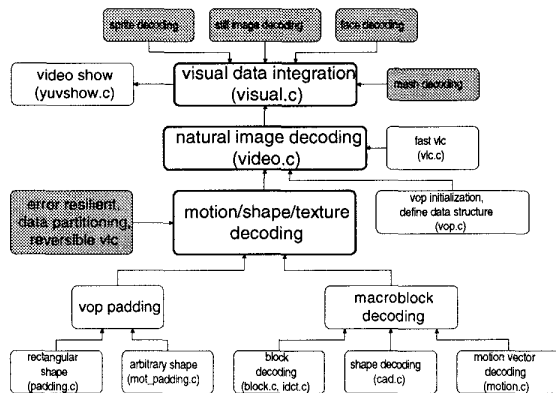


Figure 1: Block diagram of our software structure

The modules displayed as shaded blocks in Figure 1 are not covered by this version of our software. They can be added later.

Context-based arithmetic decoding, padding for motion compensation, and variable length decoding are three of the most computationally intensive algorithms in MPEG-4 video decoding [4]. In the next few sections, we discuss how these three modules are realized in our real-time decoder.

3. Context Based Arithmetic Decoding

MPEG-4 uses a block based method for

coding arbitrary shape information. Specifically, shapes are formed by primary data structures denoted as binary alpha blocks (BABs) and the decoding method used for each BAB is context based arithmetic decoding (CAD). Each BAB is a 16x16 block of binary pixels with values denoting opacity or transparency of the pixels - '1' for opaque and '0' for transparent. MPEG-4 defines a total of seven different BAB types (types 0 to 6), of which only types 4, 5 and 6 require decoding. Type 4 BAB is decoded using intra mode decoding (IntraCAD) and types 5 & 6 using inter mode decoding (InterCAD).

3.1 Context Computation

For each pixel in the BAB to be decoded, a context is computed based on the definition shown in Figures 2(a) (for the INTRA case) and 2(b) (for the INTER case). Figure 2(a) is the template to define a 10-bit context for the arithmetic decoder for INTRA mode decoding of the current shape pixel C. Figure 2(b) is the template to define a 9-bit context for INTER mode decoding of the current shape pixel C. The context so determined is used to index an Intra-Probability or Inter-Probability table, respectively [2]. These tables contain the probabilities of the current pixel being '0', given the context; and these probabilities are used to drive the binary arithmetic decoder.

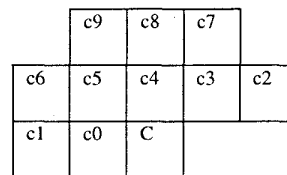


Fig. 2(a) Pixels in current BAB for INTRA mode

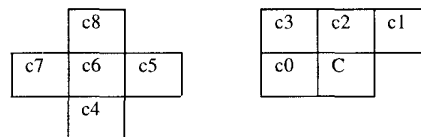


Fig. 2(b) Pixels in motion compensated BAB (left) and in current BAB (right)

3.2 Arithmetic Decoding

While performing decoding, the arithmetic decoder essentially maintains and updates three values - the current lower bound L, the current range R, and the current arithmetic codeword V, while it is being driven by probabilities obtained from the computed context as described in the previous section.

Before the start of decoding, the lower bound of the current interval L is initialized to 0, the range R is initialized to HALF - 1, where HALF = 0x80000000,

and V is initialized to the first 31 bits of the arithmetic codeword as obtained from the bitstream to be decoded. L, R and V are all 32-bit registers.

Decoding then begins. The context for the current pixel to be decoded is computed and used to index the corresponding probability table according to the INTRA or the INTER case. The least probable symbol LPS and its corresponding probability pLPS is then derived. The range of the least probable symbol rLPS is obtained by $rLPS = R * pLPS$. The current interval $[L, L+R)$ is then divided into two to give intervals $[L, L+R-rLPS)$ and $[L+R-rLPS, L+R)$.

The current pixel, C, is decoded as follows:

```

If (L ≤ V < L+R-rLPS),
{
  C = opposite of LPS /* decode pixel */
  L = L /* update lower bound */
  R = R-rLPS /* update range */
}
else
{
  C = LPS /* decode pixel */
  L = L+R-rLPS /* update lower bound */
  R = rLPS /* update range */
}

```

If the updated range $R < 0x40000000$, a re-normalisation process is performed on L, R, and V, and a new bit is shifted into V. This process is repeated until the condition $R < 0x40000000$ is no longer true. Decoding of next pixel then continues. Another complication that the decoder needs to take care of is to properly skip inserted 1's in the bitstream while decoding so as to un-do bit stuffing used by the encoder to avoid start code emulation.

4. Padding

According to [4], padding is the most computationally intensive portion in MPEG-4 decoding. It takes more than 30% of the decoder computational power. A fast padding algorithm is therefore very desirable. In our decoding, binary alpha planes are used for shape. Our padding algorithm is briefly discussed below.

4.1 Padding for rectangular shape

Figure 3(a) shows the padding technique used in MPEG-4 MoMuSys software [5], in which both the horizontal and the vertical paddings are done as the extension of horizontal and vertical borders of the object. The residual areas (the areas with cross lines) are padded with the average of two corresponding border pixels. This procedure is slower than extending

the borders, in which only memory accesses happen. Observing this padding procedure closely, we find that the averaging part is unnecessary. For example, both pixels C1 and C2 are obtained by duplicating C, so $C1 = C2 = C$ and $A1 = C$. This invokes us to use a simpler way to do padding for this area, which is shown in Figure 3(b). In our method, padding is first done horizontally along the object border and then vertically along the horizontal-padded object. In this way we eliminate the time for averaging and checking for whether the pixels belong to the area with cross lines.

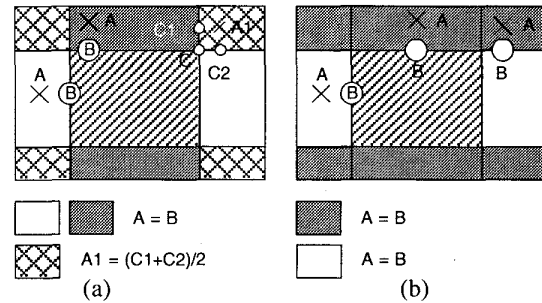


Figure 3: Padding schemes used in (a) MoMuSys software and (b) our software

4.2 Padding for arbitrary shape

According to the Committee Draft [2], there are three padding schemes for different types of macroblock (MB). Here we present our method for boundary MBs. Boundary MB padding is done block by block. Each pixel in a block (8x8 pixels) can be one of the three types: pixel of the object (type 1), pixel of the background to be padded with duplication (type 2), and pixel of the background to be padded with averaging (type 3). In MPEG-4 MoMuSys software [5], three 64-byte memory segments are used for shape, texture, and indicators for three possible types for each pixel. Since we consider only binary shapes, the shape information is either 1 or 0. This invokes us to represent shape information with 1 bit/pixel instead of 8 bits/pixel, as defined in [5]. Hence, in our scheme, shape information for a block is represented by 8 bytes, with 1 byte (8 bits) for each row. With this representation, we reduce memory and cut down manipulation time for shape information.

5. Variable Length Code (VLC) Decoding

In MPEG-4, a lot of data are coded via VLCs in the VOP layer. Some VLCs represent single data, while others, such as DCT coefficient codes, represent triplets {run,level,last}. In general, the aim of VLC decoding is to retrieve this data each VLC codeword represents. We develop a binary tree search technique

to improve the speed for VLC decoding, as discussed below.

Since the VLCs are unique prefix codes, they can be represented by a binary tree with the paths from the root node to the leaves representing the VLCs and the corresponding data stored at the leaf nodes. In the binary tree implementation, the right branch of a node represents bit '1' and the left branch represents bit '0'. Therefore, by starting from the root node, decoding the VLC simply involves traversing through the binary tree according to the bit scanned from the incoming bit-stream until the leaf node is reached. The corresponding data item that is represented by the valid VLC is then retrieved at the leaf node. These are presented in Figure 4.

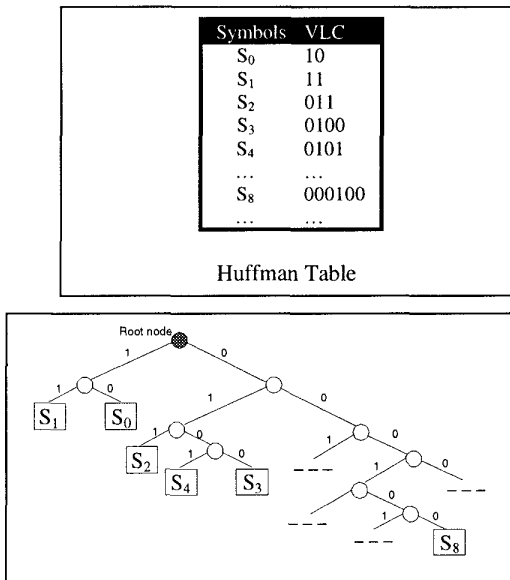


Figure 4: A Huffman table represented by a binary tree

Fast decoding is achieved using binary tree search and the implementation requires only small memory. There are two main steps in VLC decoding. The first is to create a binary tree for each Huffman code tables and the second is the binary tree search for the respective Huffman table driven by scanning the incoming bit-stream. In our implementation, each node in the binary tree consists of three components: The right node pointer, the left node pointer and the symbol data item. To build a binary tree for the corresponding Huffman table, the following pseudo codes are used:

```

CREATE ROOT node
FOR each VLC_CODE and VLC_DATA in the Huffman table
  PTR = ROOT
  FOR each BIT read starting from the MSB of the VLC_CODE
    IF BIT = 1
      IF PTR's right node is empty
        PTR's right node points to new node
      ENDIF
      PTR = PTR's right node
    ELSE
      IF PTR's left node is empty
        PTR's left node points to new node
      ENDIF
      PTR = PTR's left node
    ENDIF
  ENDFOR
  STORE VLC_DATA into the node pointed by PTR
ENDFOR

```

Once the binary tree is created, decoding the bitstream is simply done by traversing the binary tree from the root node using the following pseudo codes:

```

PTR = ROOT
WHILE PTR not the leaf node
  BIT = bit from incoming bit-stream
  IF BIT=1
    PTR = PTR's right node
  ELSE
    PTR = PTR's left node
  ENDIF
ENDWHILE
RETRIEVE VLC_DATA from the node pointed by PTR

```

6. Conclusion

In this paper, we briefly describe some of our methods in developing our MPEG-4 decoder to speed up decoding. The performance of this preliminary version of our decoder is satisfactory. The software can decode video sequences over 18 frames per second, which is on the average, five times faster than that of the MPEG-4 simulation software [5].

References

- [1] ISO/IEC JTC1/SC29/WG11, MPEG98/M3100, MPEG-4 Video Verification Model Version 9.1, February 1998.
- [2] ISO/IEC JTC1/SC29/WG11, N2202, Information Technology-Coding of Audio-Visual Objects: Visual ISO/IEC 14496-2, Committee Draft, Tokyo, March 1998.
- [3] N. Brady, F. Bossen and N. Murphy, "Context-based Arithmetic Encoding of 2D Shape Sequences", International Conference on Image Processing ICIP'97, pp.29-32, 1997.
- [4] ISO/IEC JTC1/SC29/WG11, MPEG96/M1257, Complexity Analysis of the MPEG-4 Video Verification Model Decoder, September 1996.
- [5] ISO/IEC JTC1/SC29/WG 11, N2205, Text of 14496-5 (MPEG-4 simulation software) final committee draft, March 1998.