Design and Implementation of an Efficient MPEG-4 Interactive Terminal on Embedded Devices

Yi-Chin Huang, Tu-Chun Yin, Kou-Shin Yang, Yan-Jun Chang, Meng-Jyi Shieh, Wen-Chin Chen
Communication and Multimedia Lab.,
Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan

E-Mail: {yichin, birding, voyager, ckuan, feitian, wcchen}@cmlab.csie.ntu.edu.tw

ABSTRACT

We present an efficient MPEG-4-based interactive player for PDA-like embedded devices in this paper. Our system can receive media from various sources, then decompress and compose them in an object-oriented manner. Embedded devices such as PDA usually have limited computational resources, memory size, power budgets, and multimedia capabilities. To overcome these constraints, two novel mechanisms were introduced, namely *adaptive frame rate (AFR)* and *scene cache graph management (SCGM)*. Furthermore, a *media decoding framework* was proposed such that multimedia objects of different formats can be retrieved from different sources and then be decoded. This framework can be extended by add-on components. A *semi-pull model* was also designed for the synchronization of heterogeneous media objects. Finally, all the key modules were efficiently implemented and optimized. These modules include MPEG-4 video decoder, 2D/3D graphic engine, buffer management, script engine, and streaming service.

1. INTRODUCTION

With rapid hardware advance and the growth of broadband networks, MPEG-4 is becoming one of the most important international standards. Unlike the traditional frame-based multimedia standard, MPEG-4 adopts the object-oriented methodology, which integrates the existing multimedia technologies, such as still images, 2D/3D graphics, animations, video, audio, and virtual reality into one single architecture. MPEG-4 specifies a mechanism for describing the spatial and temporal relations among the different multimedia components of the application. This mechanism is called the "MPEG-4 Scene Description" in the standard, which is termed "BInary Format for Scenes" (BIFS). Furthermore, users can define the interactive behaviors and logical relationships among these media. Being object-based and programmatic, MPEG-4 will bring on a variety of new multimedia applications.

One of the essences of MPEG-4 is the interoperability of different platforms. With the popularity of embedded devices such as PDA, mobile devices will surely be the platforms for the emerging mobile multimedia applications. However, embedded devices are usually under the constraints of lower computing power and limited hardware supports. Therefore, there are many technical issues need to be tackled in designing and implementing an efficient MPEG-4 system on embedded devices. These issues are: 1) Lower computing power; 2) Less hardware acceleration; 3) Constrained memory size; and 4) Lack of built-in multimedia APIs. For the first three issues, some optimization efforts are needed to reduce the time and space complexity. Unacceptable responding time and heavy power consumption may occur if cares are not taken for these system resource constraints. For the last one issue, on embedded devices, there is usually no high-level multimedia APIs, e.g. DirectX, available to help programmers to access the advanced features of high-performance hardware such as 3D graphics acceleration chips and sound cards. Moreover, APIs for playing back multimedia streams, e.g. DirectShow, are too complex for embedded devices. Thus, we should design a framework for audio-visual streams decoding and synchronization.

This paper consists of six parts. Part 1 gives an introduction to MPEG-4 and the problems while designing MPEG-4 system on embedded devices. Part 2 describes the system architecture. Part 3 presents the synchronization mechanism for heterogeneous media. Part 4 presents some implementation and optimization details. Part 5 shows the experimental results of the system and the performance of the novel mechanisms we proposed. Finally, Part 6 gives future plans and concludes the paper.

2. SYSTEM ARCHITECTURE

2.1 The MPEG-4 Media Process Flow

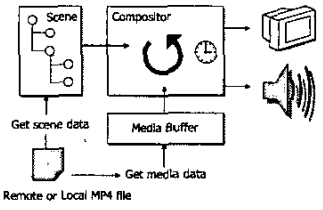

Figure 1. The processing flow of a MPEG-4 presentation

Figure 1 gives the general processing flow of an MPEG-4 media presentation. At first, the MPEG-4 media data are retrieved from local files or remote server. These data are decoded to constitute the scene description of the content. A scene graph is then built. Based on the scene graph, the media data are retrieved from different sources accordingly. These media streams are decoded into media buffer and are ready to be used by the compositor to present the whole MPEG-4 content.

0-7803-8603-5/04/$20.00 ©2004 IEEE

There are some challenging issues for implementing a MPEG-4 scene player: First, a BIFS scene graph must be well-managed with related resources. Second, the system should support various media formats. Furthermore, media may come from different sources. The system must access not only local storages but also remote services in streaming fashion. Finally, the whole scene with all associated media must be properly presented in a synchronized manner. From the above discussion, our system is designed as shown in Figure 2. The system consists of several modules: Scene Graph Manager, Media Decoding Framework, Composition Buffer, Audio Engine, and 2D/3D Graphic Engine.

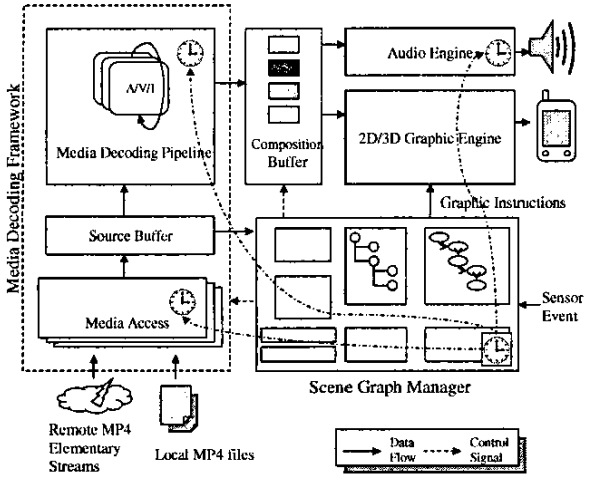

Figure 2. System Architecture

The function of each module is as follows. Scene Graph Manager, which is the kernel of the system, manages the operations of the scene graph. Media Decoding Framework is in charge of retrieving various media data and then decoding them. Composition Buffer stores the decoded data. Under the operation of semi-pull model, Scene Graph Manager synchronizes the presentation of heterogeneous media objects. In order to compose the whole scene, Scene Graph Manager also generates graphic instructions according to the information and 2D/3D spatial relationship of media objects defined by the scene description. According to these instructions, Graphic Engine then efficiently renders 2D/3D graphic primitives as well as textures. At the same time, Audio Engine renders decoded audio streams.

2.2 Media Decoding Framework

MPEG-4 supports media of various formats, from different sources and conveyed by different file formats or network protocols. To fulfill these requirements, a generic framework for decoding comprehensive media streams is proposed. In this framework, a **Media Decoding Stream** (MDS) is established by connecting the **Media Access Module** (MAM) and a **Media Decoding Pipeline** (MDP), as shown in Figure 2. Three generic input/output interfaces and Source Buffer, as the bridge of the connection, make MAM and MDP work independently. These three interfaces are the output interface of MAM and the input/output interfaces of MDP. Whenever a new component implementing these interfaces is plugged-in the system, the system is able to process new media data. This framework provides a feasible and extensible mechanism to control various supported media streams.

2.3 Scene Graph Manager

2.3.1 Scene (Cache) Graph Management

ISO/IEC 14496-1 specifies numerous kinds of nodes and data types. In order to manage these nodes with a graph data structure, namely **Scene Graph**, a general form is introduced to represent each node. To accelerate the rendering of the scene graph, each node is attached with a corresponding cache render resource, in which pre-computed information is cached. These cached render resources constitute another graph, namely **Scene Cache Graph**. Only when the values of some fields are changed, a node needs to recompute related information and update its corresponding cached resource. In this way, considerable amount of computational resource are saved. The performance of SCGM will be shown in Part 5.

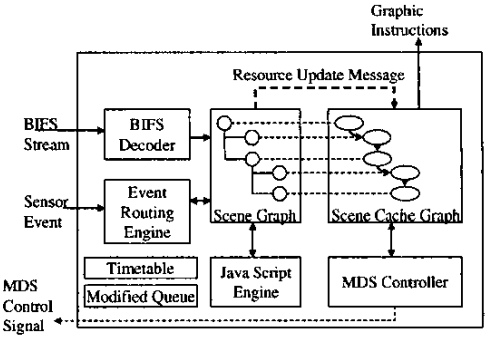

Figure 3. Architecture of Scene Graph Manger

2.3.2 MDS Controller

The MDS Controller, which is the controller of the Media Decoding Framework, constructs necessary MDSs for decoding media to the composition buffer. A MDS is constructed by connecting an appropriate media access module and a decoding pipeline. The MDS Controller also drives the data retrieving thread and decoding thread in Media Access and Media Decoding Pipeline modules, respectively.

3. SYNCHRONIZATION OF HETEROGENEOUS MEDIA

In this section, we introduce semi-pull model to solve the issues of intra-media synchronization for media of each type and inter-media synchronization between different media streams.

3.1 Categories of Media

Media objects supported by MPEG-4 can be classified into three categories according to their sampling time length: (1) **Instant Time Media** are the media data that can be computed immediately. They are usually processed in EMP (will be discussed in 4.1). For example, the event generated by a *TimeSensor* is such kind of media. (2) **Discrete Time Media** are the media whose sampling lengths are long enough so that computers can accomplish the processing. Most of these media are visually related, such as video and animation. For example, a video with frame rate 29.97 fps has media sampling time about 33 ms, which is a long time for the devices. (3) **Continual Time Media** are those media whose sampling time is very short. These kinds of media are usually auditory. For example, the sampling rate of an audio sound is 44100, its sample time is 0.000023 sec.

3.2 Semi-Pull Model

We propose semi-pull model to solve both intra-media and inter-media synchronization problems in a uniform way. In pull model, which is a conventional model to decode and render a media

stream, the render thread accomplishes all the tasks required to present a media stream, such as getting source data, decoding, and rendering. Therefore, the timing to render a decoded frame is controlled by the render thread. Pull model is an appropriate model for the simple case that only one video and audio stream is presented concurrently. However, for the more complicated case when more continual media objects in an MPEG-4 scene should be presented in a synchronized way, pull model does not apply.

In our semi-pull model, a new thread, decoding thread, is created for decoding. The decoding threads deliver decoded data through composition buffers, as shown in Figure 2. Each decoded data sample is associated with a composition timestamp, which indicates the time at which this sample should be rendered. A composition timestamp is created with respect to a global reference clock generated by scene graph manager. We will discuss how the semi-pull model achieves the heterogeneous media synchronization.

3.3 Intra-media Synchronization

The goal of intra-media synchronization is to maintain the on-time presentation of the data samples so that acceptable user perception at playback is ensured. Without intra-media synchronization, the presentation of the media may be interrupted by pauses or jitters. For instant time media, synchronization can be achieved if the EMP execution rate is kept high enough. For other media types, semi-pull model ensures that the presented data sample is the most updated and rendered on time. Thus, intra-media synchronization of each kind of media is accomplished.

3.4 Inter-media Synchronization

Inter-media synchronization can be categorized into two cases: one is for the synchronization between time-variant (discrete and continual) media and the other is for the synchronization between instant and time-variant media. A typical example of the first case is the lip-synchronization between video and audio streams. The first case can be resolved by forcing all media streams synchronized to the global clock. In this way, the synchronization can be accomplished, except for media streams that have their own reference clocks. Currently in MPEG-4, only audio streams have this problem. Audio streams are rendered by a sound card, which has a local clock. Inevitably, skew between the local and global clocks may be intolerable after a long-time playback. For this problem we have to synchronize the audio stream to the global clock by adjusting the amount of waveform data that is to be fed into frame buffer on the sound card. The algorithm adopted in our implementation is:

```
Initialize temp buffer
While more decoded waveform data {
    Copy decoded data to temp;
    TimeOffset =
      SystemAbsoluteTime - BufferPlayingTime;
    If( abs(TimeOffset) < Threshold )
       Smoothly drop or insert sample;
    Else {
       Reset audio device;
       Drop a segment of samples;
    }
    Output temp data to audio device buffer;
}
```

For the second case, the data of instant time media are processed when current frame is about to be rendered. Because of the high execution frequency of EMP, the delay between the time at which the data is created and processed is restricted within a tolerable time interval. Therefore instant time media can be synchronized to the global clock, which is also referenced by time-variant media. The inter-media synchronization is achieved.

4. IMPLEMENTATION AND OPTIMIZATION

4.1 Adaptive Frame Rate Mechanism

There are two render threads in our system. One is for rendering visual frames; the other is for rendering audio streams. The visual render thread is responsible for driving scene graph manager and graphic engine. It takes a large part of the limited computational resources of the system. Thus, an optimization mechanism is proposed to reduce the workload of this thread.

	Procedure Type	Overhead	Execution Freq. (times / s)	Exec. Freq.
Visual render thread	EMP	Low	30	Fixed
	SRP	High	0~30	Adaptive

Table 1. The 2 procedures of visual render thread

The job of visual render thread are divided into two procedures (see Table 1.), Event Monitoring Procedure (EMP) and Scene Render Procedure (SRP). The first one detects, routes, and handles instant time media, such as events. The second one traverse scene cache graph and drives graphic engine to render 2D/3D graphics. The overhead of EMP is quite low and the execution frequency determines the response time for incoming events of the system. Hence, the execution frequency of EMP is fixed. SRP has much higher overhead and its execution frequency determines frame rate of the scene. Thus, we have to make the execution frequency of SRP as low as possible, while preserving the visual quality. To solve this problem, the **Adaptive Frame Rate** (AFR) mechanism is proposed. In this mechanism, a scene runs in either *active state* or *inert state*. The scene runs in active state when at least one MovieTexture or TimeSensor is active. Otherwise it is in inert state. When the scene runs in inert state, render thread adopts lazy-render strategy, with which the only periodic routine task is to execute EMP. In other words, render thread executes SRP only when some events occur and somewhere in the scene has been modified. When the scene runs in active state, to continually render the active movie textures, render thread executes SRP in every specified period $\mathbf{P_a}$, in addition to what it does in inert state. We propose an algorithm to adaptively estimate the appropriate period $\mathbf{P_a}$.

Definition: $\mathbf{F} = \{f_1, f_2 \ldots f_n \mid f_i$: the frame rate of MovieTexture i$\}$

Input: **F** Output: $\mathbf{P_a}$

```
If "Fa not init" or "F changed"
  Fa = Min( Median(F), MAX-FPS );
If (Average-CPU-Utilization > threshold)
  Fa *= FPS-DECREASE-FACTOR;
Else
  Fa += FPS-INCREASE-FACTOR;
Pa = 1/Fa;
```

More computational resource can even be saved. If the determined scene frame rate 1/Pa is lower than video frame rate, some frames will be dropped by the frame-dropping filter embedded in the video decoder.

4.2 2D/3D Graphic Engine

The engine supports primitive 2D/3D rendering functionalities comprising transformation, frustum clipping, viewport, lights, and optimization of texture processing. Two essential hardware supports are not present on the target platform, including

hardware acceleration for 3D rendering and floating-point number processing unit. To overcome these problems, the engine applies fixed-point arithmetic when decimal arithmetic is required and performs all 3D rendering functionalities by software emulation.

4.3 MPEG-4 Video Decoder Optimization

4.3.1 Optimized Inverse DCT Algorithm

A large part of IDCT calculation needs floating-point operations. XScale CPU doesn't have a dedicated floating-point unit, and all the floating-point operations are implemented by software simulation, which is not efficient. Thus, we have to implement the IDCT module with only integer operations. Our implementation also adopts Feig.'s inverse DCT Algorithm [2], which significantly reduces the amount of multiplicative and additive operations. In addition, de-quantization operation is also combined with IDCT module. The most multiplications are further eliminated with this combination. Moreover, unnecessary memory accesses on temporary buffers that store dequantized data before IDCT are reduced.

4.3.2 Optimization of Motion Compensation

In the recommended decoding procedure, the predictive error for motion prediction is derived from IDCT and written in a temporary buffer. The error values are then read and added with the result of motion compensation in another buffer. In our implementation, motion compensation is directly done upon the result of IDCT in the same buffer. A temporary buffer is thus eliminated and the number of memory access is reduced.

4.3.3 Optimization of Color Conversion

The color conversion from YCbCr to RGB is optimized by table-lookup. Each floating-point operation is replaced by a single memory addressing.

4.4 Buffer Management

This module maintains two buffers to hold decoded data. While a decoding thread writes data to one buffer, the render thread reads data from the other one. To avoid the race condition, both threads may lock the buffers. When the decoding thread finishes writing to the buffer, it locks and toggles the two buffers. When the render thread reads data from the buffer, it also issues a lock. Since it takes very short time for the toggling and reading operations, the potential waiting time can be minimized.

4.5 Script Engine

To enhance the interactivity of BIFS scene, the system supports JavaScript mechanism specified in MPEG-4. The script engine parses all the script codes in the scene and transforms them into a syntax tree for speeding up the script execution. When an event is routed to a Script node, the corresponding function is evoked.

4.6 Video Streaming

A media access module is implemented for MPEG-4 video streaming over RTP. The module maintains the RTP/RTCP session of an MPEG-4 stream, feedbacks the network status, manages the buffer, controls transmission errors and composes the fragmented data during transmission. The module adopts forward error correction (FEC) to recover data when a packet is lost. At the sender side, the XOR operator is used to generate a redundant packet among a group of packets. At the receiver side, if a single packet is lost, the lost packet can be recovered by applying XOR operation on those correctly received packets.

5. EXPERIMENTAL RESULTS

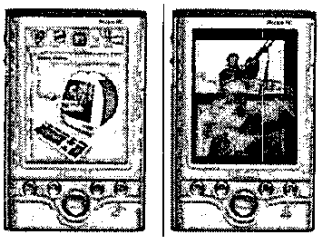

Figure 4. Snapshots of MPEG-4 content

We developed the system on Toshiba e740 Pocket PC with Intel XScale PXA 250 CPU and built-in 802.11b wireless module.

Figure 4 shows two snapshots of MPEG-4 content. In the left one, the screen of the "monitor" (the 3D model placed in the middle) binds an active movie texture. The experimental results of the performance of SCGM for this scene are described as follows. When the rendering frame rate is fixed at 12 fps, the average CPU load is 98.65% without SCGM. When SCGM is enabled, the CPU load is improved toward 6.37%. In other words, roughly 15 times of the computational resource is saved.

In the right of Figure 4, there are three objects, each of which is bound with a movie texture. The elliptical object in the middle of the scene is set with 0.3 transparency and randomly moving around in the scene. These videos are in QCIF and frame rate 15 fps. Two of them are streamed from remote server, while the third one is retrieved from local file. With AFR mechanism, the highest frame rate of the scene is about 14 fps.

To measure the performance of AFR, we consider the lifetime of the battery with power consumption varies from 100% to 20%. The experimental result shows that AFR gains 118% improvement of the lifetime in the first scene. Under the same condition, only 27% improvement of the lifetime is gained in the second scene. The reason that AFR performs better in the first scene is that most part of it is in inert state, while AFR works more efficiently in that state.

6. CONCLUSIONS

We have successfully developed a MPEG-4 interactive player on a resource constrained PDA. Our system adopts four novel mechanisms, namely adaptive frame rate, scene cache graph management, media decoding framework and semi-pull model for resolving the problems including computational resource reduction, low power consumption, comprehensive media format support and heterogeneous media synchronization. For further research we will improve the efficiency of our system and develop new functionalities such as MPEG-4 Multi User Worlds.

7. REFERENCES

[1] ISO/IEC 14496 AM 1, Part 1: Systems, Part 2: Visual, Part 3: Audio, Part 6: DMIF, International Organization for Standardization, 2000.

[2] E. Feig, S. Winograd, "Fast Algorithms for the Discrete Cosine Transform," IEEE Trans. on Signal processing, vol.40, no. 9, pp. 2174-2193, 1992.

[3] RFC 3016, Y. Kikuchi, T. Nomura, S. Fukunaga, Y. Matsui, and H. Kimata., "RTP Payload Format for MPEG-4 Audio/Visual Streams", November 2000

[4] Meng-Jyi Shieh, Chien-Feng Huang, Cha-Dong Duh, Wei-Teh Wang, Wen-Chin Chen, "Implementation of an efficient MPEG-4 2D/3D Mixed Renderer for Visual Editing and Interactive Applications", 3rd MPEG-4 Workshop and Exhibition.