Streaming of DivX* AVI Movies†

Roger Zimmermann
Department of Computer Science
University of Southern California
Los Angeles, California 90089
rzimmerm@usc.edu

ABSTRACT

In recent years the MPEG-4 ISO compression standard has gained much attention. With its high compression ratio it promises to make broadband streaming more feasible. However, currently there are only a limited number of fully ISO standard compliant MPEG-4 systems widely available (Apple's Quicktime 6 is one of the first). Some partial implementations, such as the MPEG-4 style codec "DivX" are popular for encoding video content into AVI (Audio Video Interleave) files. The AVI file format was originally not intended for streaming. In this report we document our efforts to enable streaming of DivX AVI files across broadband IP based networks. This will allow the large number of existing AVI files to be made available in streaming applications.

Categories and Subject Descriptors

E.4 [**Data Files**]: Organization/Structure; H.3.2 [**Information Storage**]: File organization; H.5.1 [**Multimedia Information Systems**]: Video

Keywords

Streaming media, DivX, MPEG-4, file formats, AVI, buffer management

1. INTRODUCTION

Several developments over the last few years have increased interest in streaming media technologies. Among them are the following:

- Broadband networking technologies such as ADSL and Cable modems became available in many metropolitan areas. End-users for the first time had reasonably priced access to approximately 256 to 768 kbps, or sometimes up to 1.5 Mbps, downstream bandwidth.

- Advances in compression technologies have emerged that allow compressing full-screen NTSC video into a stream requiring less than 1 Mbps of bandwidth. The MPEG-4 standard [1] was one of the widely discussed and publicized techniques.

One of the first implementations of an MPEG-4 style codec was available from Microsoft. The implementation did not fully comply with the MPEG-4 standard, however it demonstrated the very high compression ratio achievable (more than 300:1). Therefore, it became feasible, for example, to store the contents of a DVD (4.7 GB capacity and compressed with the older, less effective MPEG-2 standard) on a CD-ROM (650 MB capacity). Originally Microsoft's software only allowed decoding of media. However, after it had been slightly altered, both encoding and decoding were possible. This altered version became widely known as "DivX;-)" [2]. The early versions of the DivX codec were not fully MPEG-4 ISO standard compliant. For example, originally it did not use the MPEG-4 ISO standard file format. More recently, that file format has become an option, but is not yet very popular. Instead the media data is commonly stored in the Microsoft AVI (Audio Video Interleave) file format [3], which is a special case of the Resource Interchange File Format (RIFF). Furthermore, most of the DivX files available on the Internet use MPEG-1 layer 3 audio encoding (popularly known as MP3) and not the MPEG-4 audio codec.

Several tools became available on the Internet to easily transcode MPEG-2 encoded media files into DivX AVI files. One of the more popular tools was FlaskMPEG (go.to/flaskmpeg) which has since been superseded by XMPEG (www.mp3guest.com). These tools allow to easily convert MPEG-2 content into DivX files. For example, one of our 10-minute demonstration videos with an original file size of 423 MB in MPEG-2 format resulted in a 63.8 MB-sized DivX file. Hence, the required playback bandwidth dropped from 5.6 Mbps to 870 kbps (a 6.5:1 reduction).

One of the disadvantages of the AVI file format is that it was not designed for streaming. AVI and Video for Windows were developed for playback of audio and video from hard disks and CD-ROMs on personal computers. They are also adequate for downloading a video file from a remote site on the Internet for subsequent playback from the computer's hard drive. They are not well suited for real-time or streaming video playback over networks.

†This research has been funded in part by NSF grant IIS-0082826 unrestricted cash/equipment gifts from Intel and Metromedia Fiber Networks and by the Integrated Media Systems Center, a National Science Foundation Engineering Research Center, Cooperative Agreement No. EEC-9529152.
*DivX is a trademark of DivXNetworks, Inc.

SAC 2003, Melbourne, Florida, USA

Microsoft has since defined a new format in the form of the Active/Advanced Streaming Format (ASF) with improved support for video over networks. However, the most widely used version called ASF 1.0 has not been officially documented. Therefore, not many open source tools and media players can work with ASF files and its popularity on the Internet is limited.

In this report we document our efforts in experimenting with streaming DivX AVI files across broadband IP based networks. Specifically, we elaborate on the client playback software that is based on open source tools. The rest of this report is organized as follows. Section 2 details our design. Results are presented in Section 3 and we conclude with Section 4.

2. APPROACH

We have been designing and implementing scalable streaming servers for the past several years [4, 5]. Initially we did most of our system testing with either MPEG-1 or MPEG-2 streams. When the DivX codec became available we were interested to try MPEG-4 streaming at lower bit rates that would be suitable for ADSL or Cable modem connections. At first, version 3 of the DivX codec was only available in binary code form for the Windows platform. However, we had already implemented a Linux MPEG-2 client and wanted to re-use as much as possible of that software.

While investigating possible solutions we discovered a project called *avifile* for Linux (avifile.sourceforge.net). Its goal was to provide a runtime environment on Linux that would allow the execution of Windows codecs which are generally provided as Dynamic Link Libraries (DLL). To that end, it uses a specific subset of the libraries from the *Wine* project (www.winehq.com). Wine is an implementation of the Windows 3.x and Win32 application programming interface on top of X11 and Unix. Because Windows codecs have a very narrow and well-defined interface, only a small subset of the Wine code — a library to provide a runtime environment for DLLs — is necessary to successfully execute Windows codecs.

We found the avifile project intriguing and decided to use it as the decoding engine within our playback application. The avifile library included a rudimentary player that was capable of reading AVI files, decoding their content and rendering the video on the screen and the audio via the soundcard. We decided to use this player as a starting point for our own implementation. After successfully testing the playback of several AVI files with this application we were presented with the following challenges for our own player:

- The media data from our server would arrive in a memory buffer while the avifile library was designed to read data from a file. Hence we had to redirect all file system calls to our own buffer manager.
- The AVI file structure is such that a codec not just reads a file sequentially. On the contrary, it generally will seek to various locations within the file to gather information about the media structure before and during the playback. For example, an index table labelled 'idx1' usually provides the starting point of each frame within the file. This table is often located at the very end of an AVI file (see Figure 3). Furthermore, additional seeking takes place to multiplex between the video and audio data that is interleaved within a file.

	Movie Length (Size)	'idx1' Table Size
"Test1"	2 hours (625 MB)	4800 KB
"Test2"	10 mins (63.8 MB)	349 KB

Table 1: Frame table size and the overall movie size for two test movies (720×480 resolution at 30 fps).

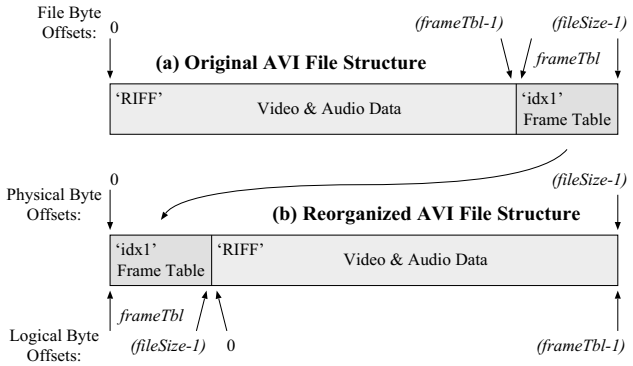

Figure 1: File data reorganization from (a) the original to (b) a new layout to confine seek operations to a bounded area.

One of the advantages of streaming as opposed to downloading media is that playback will start after a short initial latency. Therefore, the amount of buffered data is severely limited and seek operations that span the complete file cannot be readily supported. A simple solution where every seek is propagated back to the server side to retrieve the correct data is not generally feasible because the client-server latencies are not short enough to guarantee a smooth playback. Hence, we decided to address these challenges in two ways. First, a simple reorganization of the AVI file prior to placing it onto the server would ensure that seeks would not exceed the buffered data. And second, the implementation of a sophisticated buffer manager would allow us bounded seeking capabilities within the playout buffer.

2.1 Data Reorganization

Figure 1 shows the data reorganization that we performed prior to streaming a movie. The 'idx1' frame index table, which is generally located at the end of an AVI file is moved to the very beginning. The video and audio data that starts with a 'RIFF' four character signature is appended to the table. No other changes are made to the contents of the file. Specifically, no internal pointers are changed to match the new frame locations. The buffer manager logic is designed to correctly and transparently translate all access locations. Its operation is described in the next sections. For streaming, one disadvantage of moving the frame table to the beginning of the file is that playback can only start after the complete table has been received at the client side. However, Table 1 shows that generally the frame table size is less than 1% of the complete size of a movie and does therefore not introduce significant startup latencies.

2.2 Buffer Management

The data reorganization described in the previous paragraph allows the server to stream the AVI file sequentially, from the beginning to the end. On the client side, sub-

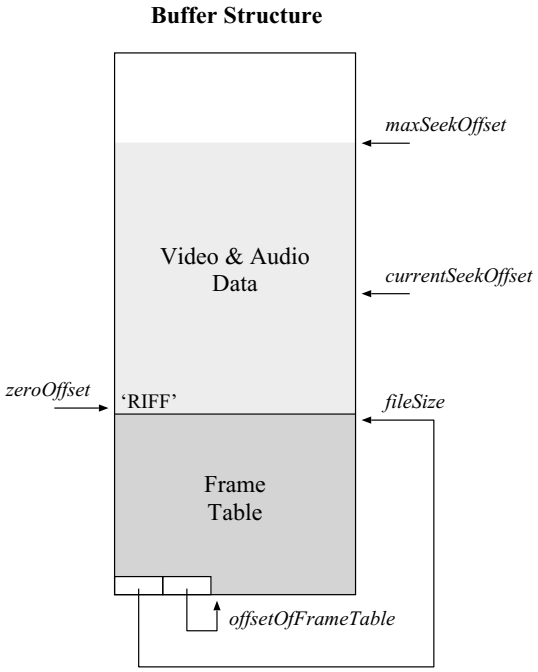

Figure 2: The client buffer organization after the start of streaming. The frame table and part of the frame data has arrived. Two additionally transmitted pointers allow the buffer manager to know every byte offset in the original file.

sequently care must be taken that the codec accesses the correct data. Recall that we did not alter any index values within the AVI file. Therefore, the buffer manager must correctly translate any file system calls (`read` and `seek`) attempted by the codec to the relevant locations in the client playout buffer.

Figure 2 shows the initial organization of the client buffer when data starts to arrive. Two number values are transmitted at the very beginning of the data stream: (1) the file size: *fileSize*, and (2) the byte offset where the frame table starts: *offsetOfFrameTable*. From these two values, the buffer manager is able to compute additional important information:

- The frame table size is *(fileSize - offsetOfFrameTable)*.
- The beginning of the frame data (the 'RIFF' signature) starts *(fileSize - offsetOfFrameTable) + 2 × sizeof(int)* from the beginning of the buffer.
- Data with file offsets between *offsetOfFrameTable . . . fileSize* and *0 . . . maxSeekOffset* are available for consumption by the codec.

Based on this information we implemented four functions to replace the file system calls `open()`, `read()`, `seek()`, and `close()` within the avifile library. Each replacement function has the same set of parameters as its original counterpart.

- `QOpen()`: For compatibility reasons this function will accept a file name. However, the filename is discarded because data is moved directly from the network into the memory buffer. The buffer manager is initialized and the *zeroOffset* location is set as soon as data starts to arrive, which is generally before this function is called.
- `QRead()`: This function retrieves a specific amount of data from the current seek position in the buffer. The *currentSeekOffset* is then adjusted to reflect the next unread byte. Error checks ensure that data is not read beyond the current buffer end. If this condition were to occur, the function would wait for more data to arrive and then return. In that case, the buffer sizes and/or the streaming rate are misconfigured and a display hiccup would result. Similarly, if data arrives too fast, i.e., above the decoder consumption rate, then either the server must be slowed down via a flow control mechanism, or the system is misconfigured and the streaming needs to be manually adjusted.
- `QSeek()`: The seek pointer *currentSeekOffset* is repositioned according to the passed parameters. The seek position is carefully checked against the minimum and maximum seek values maintained by the buffer manager. Any out-of-range seeking will result in display disruptions.
- `QClose()`: Releases the allocated resources and reinitializes the buffer manager.

2.3 Freeing Data

A significant challenge that arises if the buffer manager allows seek operations is the following: When can we be certain that a data item will not be accessed anymore in the future? With a purely sequential stream format — e.g., an MPEG-2 program stream — the read pointer in the data buffer will generally be monotonically advanced and data behind it can be freed. However, with the AVI file format seek operations do not guarantee a monotonically increasing read position. Obviously, if read locations are randomly distributed across the complete file then having a limited playout buffer will be impossible. Luckily, seek and read operations exhibit a localized behaviour. Figure 3 shows the trace data we obtained from a 10 minute long DivX AVI movie. After initially reading the frame table at the very end of the file, all operations take place within a "sliding window" of a few megabytes of data. To accommodate this behaviour we implemented a sliding release pointer in the following way. A *maxReadOffset* pointer keeps track of the furthest seek/read position. This pointer only moves forward, i.e., a seek backwards will leave it unchanged. A *freeOffset* pointer is then made to follow at a fixed distance behind the *maxReadOffset*, for example by 4 megabytes. Hence, whenever the *maxReadOffset* advances, some data can be freed behind *freeOffset*. If seek or read operations take place that do not move the *maxReadOffset* pointer, no data will be freed. We experimented with the release distance and found that most reads and seeks take place within a 2 MB sliding window. To have a safety margin we doubled the release distance to 4 MB and this worked well with all our test movies.

3. RESULTS

We have implemented the buffer manager within the *Yima* client playback software [4]. Several DivX clips were used

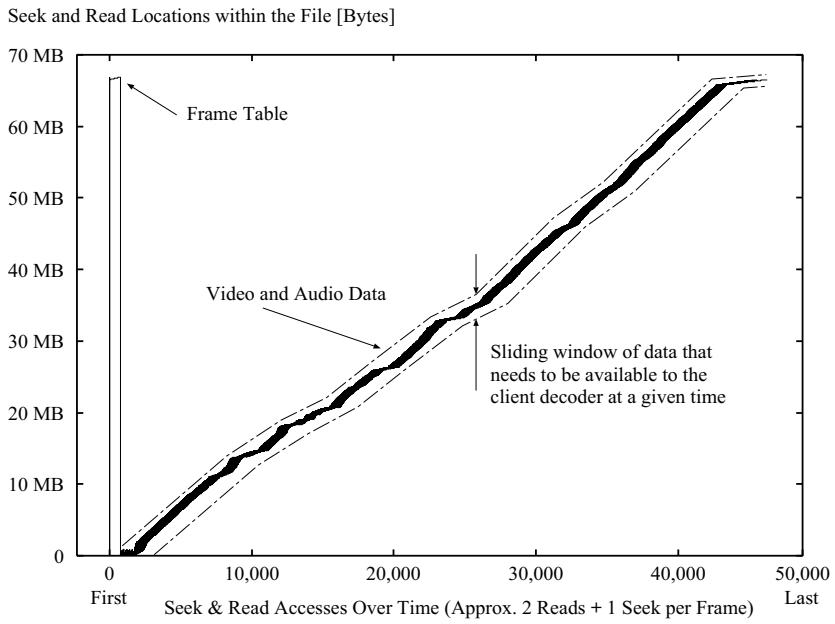

Figure 3: Typical seek and read access pattern for a 10 minute DivX movie encapsulated within an AVI file. The frame index table is physically located at the end of the file, but it must be read at the start of the playback. The x-axis enumerates all the seek and read accesses by the codec, while the y-axis shows the location (data offset) of each access in the file. As can be seen, all accesses target a narrow band of data (a "sliding window") while the file is being accessed from the start to the end. Therefore, only the data within that window needs to be available locally in the client buffer.

for testing. The parameters of two of the movies are listed in Table 1. Frame table sizes were 4.8 MB and 349 KB for movies "Test1" respectively "Test2." Our player was set to start the movie after receiving the frame table plus 2 MB of additional data. Recall that 2 MB is the amount of data that we empirically determined from traces of the movies' I/O activities to be sufficient for local seeking. Figure 3 illustrates the data that was accessed during the playback of the 10-minute movie "Test2." With the described configuration, an initial latency of 70 respectively 25 seconds resulted before the display started. With our buffer manager configured as described, it was possible to stream all movies successfully without data overflow or starvation and no seek or read access occurred outside of the buffered data.

4. CONCLUDING REMARKS & FUTURE RESEARCH DIRECTIONS

In this report we presented a combination of AVI file reorganization and sophisticated playout buffer management that allows commonly unstreamable AVI media files to be streamed. We have successfully implemented our techniques in a prototype system and demonstrated its operation across wide-area networks [5]. With the popularity of MPEG-4, we expect its official, streamable file format to eventually supersede AVI. For example, the MPEG4IP (www.mpeg4ip.net) project is building on the MPEG-4 file format as is Apple's Quicktime 6. However, our buffer management techniques are still applicable to new file formats, albeit for different applications. For example, we plan to investigate enabling pause, resume, and rewind operations with a diskless set-top box.

5. ACKNOWLEDGEMENTS

We thank Mehrdad Jahangiri and Hong Zhu for helping with the implementation and testing of the client software.

6. BIOGRAPHY

Roger Zimmermann received his Ph.D. degree in Computer Science from the University of Southern California (USC), Los Angeles. In 1998 he joined the Integrated Media Systems Center (IMSC) at USC and in 2000 he became a Research Assistant Professor with the Computer Science department of USC. His interests include novel database architectures, video server technology, and immersive environments. Some of the main projects that he has been involved in are Yima, a scalable real-time streaming architecture and the Remote Multichannel Immersion (RMI) platform (http://dmrl.usc.edu).

7. REFERENCES

[1] Moving Pictures Expert Group (MPEG), *MPEG-4 (ISO/IEC 14496)*, URL: http://www.iso.ch.

[2] J. Hibbard, "What the $%@# is DivX;-)?," *Red Herring Magazine*, January 2001.

[3] Microsoft Corporation, Microsoft Press, Redmond, WA, *Microsoft Windows Multimedia Programmer's Reference*.

[4] C. Shahabi, R. Zimmermann, K. Fu, and S.-Y. D. Yao, "Yima: A Second Generation of Continuous Media Servers," *IEEE Computer magazine*, vol. 35, no. 6, pp. 56–64, June 2002, URL: http://computer.org.

[5] R. Zimmermann, K. Fu, C. Shahabi, S.-Y. D. Yao, and H. Zhu, "Yima: Design and Evaluation of a Streaming Media System for Residential Broadband Services," in *VLDB 2001 Workshop on Databases in Telecommunications (DBTel 2001)*, Rome, Italy, September 2001, pp. 116–125.