Abstract

Research is doing exciting progress in understanding images as more than just a set of pixels, videos as more than just an even greater set of pixels synchronized with time, or audio streams as more than just an array of samples. Current implementations of multimedia data formats, such as MPEG, however, still focus on the storage of pixels and samples. This article uses a straightforward example of a system where a pixel-based video stream is superimposed on a vector-based animation to show that the lack of capability to store higher-level semantic information leaves the developers only two choices: Either accept suboptimal quality or design a new non-standard format. The paper presents an in-depth discussion of a practical example and draws conclusions for the future directions of multimedia data formats.

1 Example Scenario

Consider the following real-life scenario [4]. In a classroom we replaced the traditional chalkboard by an electronic device that, in addition to drawing capabilities, provides additional multimedia features. Lectures produced by this setup shall be transmitted through the Internet both, live and archived. Of course, the content should be watchable by a maximum amount of users without them having to install specialized software.

In order to convey the lecture content optimally, three channels have to be transmitted and archived:

- The voice of the instructor,
- the creation of the board content,
- and the lecturer pointing to parts of interest on the board.

Of course, the appropriate data format for the voice of the instructor is a compressed audio stream. The board, which mostly contains drawings, should be transmitted and stored in a vector-based animation format. The instructor can be encoded pixelwise using a traditional video format.

In order to capture a lecture, the following approach was used: The instructor is filmed as he or she acts in front of the board by using a standard video camera and is then separated by a video segmentation approach, discussed in [3, 2]. Figure 1 shows a sketch of the setup. For replay, the image of the instructor is overlaid on the board, creating the impression that the lecturer is working directly on the screen of the remote student. Figure 2 shows an example. Facial expressions and gestures of the instructor appear in relation to the vector-encoded board content.

On might think that this is a straightforward setup that should be easily encodable and transmittable using state-of-
the-art multimedia data formats. However, this article will discuss many conceptual and technical problems that developers face when trying to build a system that does this. Current implementations of multimedia data formats are still not able to handle many tasks without the loss of either desired functionality or semantics. In the end, the example is used to derive general rules for the design of semantic-aware multimedia data formats.

2 Possible Multimedia Data Formats

The following section provides a brief overview of the main alternatives a developer can choose from in order to accomplish the transmission and storage task defined in Section 1.

2.1 Playback and Transmission as (traditional) Video

The solution that many similar systems are still in favor of is using traditional video formats. A general advantage of video-based replay is that many tools are available for conversion and processing. This is especially useful for the distribution of content in the Internet (such as YouTube) or for the replay on handheld devices. PDAs, mobile phones, and iPods are able to play back different types of video formats. Manufacturers often ship the appropriate conversion and processing tools for their device along with other accessories. The tools encode and scale any operating-system-supported video format down for playback on the small device. The quality of the final replay depends on the quality of the video scaling and on the properties of the device’s display. The same is true of the ability to randomly seek into a lecture.

However, most video formats provide only very bandwidth-inefficient storage. Video codecs use a frame-by-frame encoding. This results in the stroke data being converted from vector format to pixel format. Of course, vector format storage is not only more bandwidth efficient; it is also favorable because the stroke semantics is preserved. After a lecture has been converted to video, it is for example not possible to recognize handwriting or simply edit the drawings, without recalculating and rendering huge parts of the video again. Another disadvantage concerns the way most traditional video codecs work. Most often, lossy image-compression techniques are used that are based on a DCT or Wavelet transforms. The output coefficients representing higher-frequency regions are mostly quantized because higher-frequency parts of images are assumed to be perceptually less relevant than lower-frequency parts (see for example [5, 7]). These and similar techniques (for example vector quantization as in [1]) work for most images and videos showing natural scenes where a slight blurring is perceptually negligible. For vector drawings, such as electronic chalkboard strokes, however, blurred edges are clearly disturbing. Figure 3 shows the typical artifacts resulting from frequency quantization applied on an electronic chalkboard drawing. Specialized screen capture codecs (such as used in Section 3) are able to compress board strokes well with few artifacts.

2.2 Player Plugins, Java Applets, or Flash

In order to work around the disadvantages of inefficient compression, disturbing artifacts, and the loss of board semantics, we have developed a converter and a plug-in for the Windows Media Player that allows the replay of board data simultaneously with audio. The converter encapsulates the board events into ASF files. An advantage of this is that a specialized plug-in can reuse the rendering engine of the server in the client, thus making it easier to guarantee that the replay looks exactly like the server presentation. The generated file can be played back with Windows Media Player using the plug-in. The board server and client use a human-readable, uncompressed textual format to encode events, the bandwidth consumed depends on the sampling rate of the drawing device. In practice it varies between 2.5 kbit/s and 6 kbit/s. Random seek to a specific time position is implemented using a fast redraw of the events from time position 0 to the desired position. The main disadvantage of this solution is that the user has to download the proprietary plug-in.
Figure 3. Apart from the loss of board stroke semantics and bandwidth inefficient encoding, traditional video codecs also introduce unwanted artifacts when encoding vector graphics. Original rendered chalkboard picture (left) and image showing the typical artifacts resulting from quantizing the higher frequency coefficients of a DCT-transformed image (right).

Another idea is to create a proprietary replay client in the form of a Java Applet. The advantage is that most web browsers support Java without the requirement of downloading a plugin. The main disadvantage of any self-developed client is that it has to be maintained and kept compatible with any future browsers and Java versions. The advantage, on the other hand, is that the underlying formats can be kept simple and the operational requirements for the user can be kept low. The remote viewer can turn off individual streams, and the minimal bandwidth requirements can be fulfilled by analog modems. The Java client does not require any explicit download or installation, and random seek is efficiently supported (using httpd partial get).

A third alternative is to use Macromedia Flash. Flash has become very popular because of the increasing interest in the platform-independent Internet distribution of self-made videos (see for example YouTube). Automatic conversion tools usually convert any video format into the flv-format (also called Flash Video) which is a traditional video format. The flv-format is a subset of the swf-format (“swiff”). The swf-format supports vector graphics and animations as well as interactive buttons and is compatible with a large amount of players and browsers. Unfortunately, at the time of writing this article, the swf-format does not support the live transmission of vector graphics.

2.3 MPEG-4, 7, or 21

When new techniques for video storage and compression are discussed, the video standards that are most often mentioned are MPEG-4, 7, and 21. MPEG 21 is a multimedia framework for applications and thus not suitable for coding. MPEG-7 is a multimedia description standard that is very interesting for the semantic computing community. However, it is mostly concerned with individual metadata definitions that are not yet integrated into a coherent, replayable format. Its current development status makes it unusable for the tasks described here, because the available infrastructure is not yet sufficient for a wide audience.

For the purpose of encoding vector-data, MPEG-4 contains an interesting part called Binary Format for Scenes (BIFS). BIFS includes support for the vector-based storage of 2D and 3D scenes, as well as some interactivity. As its name implies, BIFS is a binary representation that has to be compiled from a user-editable source format, called Extensible MPEG-4 Textual (XMT). Although XMT is specified as BIFS source format by the ISO standard, its biggest downside is that it is an XML-based format. XML files can only be parsed entirely, since the document opening tag has to be closed by the document ending tag at the end of the file. This makes it impossible to compile XMT files incrementally for live streaming, although BIFS is by itself a streamable format. A solution has been provided by the authors of the GPAC framework [9], developed at the Ecole Nationale Supérieure des Télécommunications (ENST) in Paris. The format is called BIFS Text (bt) and is a non-XML-based exact transcription of the BIFS stream. Some users also prefer the format for better readability as the bt document architecture is very similar to XMT-A and the syntax is close to VRML[6].

In theory, the MPEG-4 format seems to provide a very good representation at least for the storage of lectures produced within our scenario. In our experiments, however, we experienced several disadvantages. Although many pro-
grams are available that are described as capable of playing back MPEG-4 content, most of them, for example the QuickTime Player, RealPlayer, Windows Media Player, or Apple’s Video iPod, only support movie profiles and are not able to play back BIFS content. We identified three players that are capable of playing back BIFS content in combination with audio and video, namely the Osmo-Player that is part of the GPAC Framework, a Java-based player that is part of the IBM MPEG-4 Toolkit [10] called M4Play, and a plug-in for Windows by Envivio [8] that adds this functionality to the Windows Media Player and the QuickTime Player.

Both Osmo-Player and IBM’s Java-based player do not support random seek for BIFS content. Fast-forwarding and rewinding can only be implemented manually using XMT-commands. Video codecs generally do not support α-transparency. For this reason, encoding transparency in the instructor video itself is currently impossible. However, MPEG-4 supports tagging colors as transparent. Shades of transparency needed for sub-pixel-accurate segmentation cannot be used. Osmo-Player, however, does not yet support the transparency tag. The player uses a different strategy: The video is overlaid onto the board by pixelwise mixing the colors of the two layers. This results in a darkening of the board strokes (mixture with black) in the areas not occluded by the instructor and other undesired effects. M4Play supports transparency; however, as it is Java-based, the IBM player drops many frames when playing back such a video.

Streaming of MPEG-4 content over HTTP requires the conversion from plain MPEG-4 to a format called m4x. These files contain so-called “hints” that enable partial playback of MPEG-4 files. Of course, the conversion itself does not work incrementally. In other words, streaming of MPEG-4 files is only possible after lecture recording has been completed. Although the GPAC Framework allows incremental compilation of BIFS content using BIFS text, it does not yet allow for incremental creation of the audio and video track. Live streaming is usually performed using the Realtime Transport Protocol (RTP), and MPEG-4 supports the streaming of BIFS content using so-called BIFS commands. Although the computational needs for the generation and conversion of MPEG-4 BIFS would easily allow it, there is no program or framework available yet that supports live encoding and streaming of content that consists of BIFS, audio, and video. One reason for this might be the license policy [12] connected to the MPEG-4 standard which requires every application generating MPEG-4 content to pay a royalty fee. The policy has often been criticised as being the primary reason for the slow adaptation of the standard, see for example [11].

<table>
<thead>
<tr>
<th>Format</th>
<th>Board only</th>
<th>Video overlay</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPEG-4 BIFS</td>
<td>1,662 kB</td>
<td>57,839 kB</td>
</tr>
<tr>
<td>WM ScreenCapture</td>
<td>2,873 kB</td>
<td>inapplicable</td>
</tr>
<tr>
<td>MPEG-4 Movie</td>
<td>44,251 kB</td>
<td>147,367 kB</td>
</tr>
</tbody>
</table>

Table 1. Comparison of file sizes resulting when representing electronic chalkboard content alone and with overlaid instructor in different formats. Please refer to the text for a description of the experiment.

3 A Note on Bandwidth Requirements

The following short experiment is to support the discussion of the possible approaches. A sample lecture was converted into different formats. The experiment was conducted in two runs, the first run containing no video and the second run containing an overlaid instructor video. In both experiments, the audio track was omitted. The resolution of the board is 1024 × 768, and the total length of the lecture is 1 hour, 37 minutes. The lecture was encoded using MPEG-4 BIFS, with Windows Media Video using frame-by-frame screen-capture. The video files were created with the lowest possible bandwidth that did not result in any visible artifacts during replay. Of course, conclusions drawn from the presented figures take into account that “no visible artifacts” is sometimes a subjective measure. The presented figures provide an indication for the amount of data that is generated when encoding the same content in different formats, in terms of orders of magnitudes.

Table 1 shows the results of the two runs. When only the electronic chalkboard is encoded one can observe that BIFS brings the storage requirements of a 97-minute lecture down to less than two megabytes. The output of screen capture codecs is only a bit larger for this lecture. In general, the quality does not match the other formats described here. Using state-of-the-art movie codecs results in file sizes that are several orders of magnitude higher. It is evident that for big parts of the video file the perceptually relevant information between two frames actually only consists of a change in several pixels. However, even if the representation is sampled down to ten frames per second, the techniques used by traditional video encoders do not yield acceptable compression results.

In the second run of the experiment, the same lecture was encoded again but with a semi-transparent, overlaid instructor. The instructor is recorded using video resolution, that is 640 × 480 pixels and then extracted. MPEG-4 players that are able to play back BIFS content, receive the segmented video in the original resolution and scale it up to fit the board replay. For replay using Windows Media Screen-Capture codec or regular MPEG-4 movie players, the seg-
Figure 4. Traditionally, media is captured, prepared for transmission using an encoder, transmitted, and then decoded and replayed by a viewer. The main focus is the reproduction of the original content given a noisy channel.

Figure 5. Today, multimedia data formats still focus on archival and transmission of video or audio data. Any extracted information must be transmitted and archived separately in the form of metadata. Ideally, multimedia data formats should encode the entire semantics of the content so that a viewer can present the stream in an integrated fashion to the user.

Because of their broad availability, pixel-based video encoders are still predominant for almost any application, even when vector-format storage would be better for various reasons; the most important one being the preservation of semantics. In the presented example, a separated transmission of the three streams allows to switch off individual streams for connections that do not provide the required data-rates and other features, for example dimming the transparency of the lecturer or scrolling the board independently of the replay. MPEG-4 BIFS players support scaling and MPEG-4 BIFS editing tools would even allow for post-processing of lectures. In practice, however, there are too few implementations of the standard and they still have too many technical problems. Especially the lack of random seek and not being able to stream BIFS content live make this simple task of transmitting a lecture without losing semantics unachievable. At the time of writing this, a completely self-developed encoder, server, and replay-client seems to be the only viable solution in order to transmit and archive the lecture as proposed in Section 1.

4 Summary

5 Multimedia Data Formats and Semantic Computing
In the past, multimedia data formats were mainly a compressed representation of the captured input and players only acted as decoders and viewers (compare Figure 4). Users have are accustomed to have the following functionality:

- Archived replay and live transmission
- Random seek (or at least VCR-like operations)
- Broad compatibility with different formats (i.e. one player is able to play many types of media)

Of course, users expect players to continue to provide these features “even” when rich content is provided. In fact, users expect that players and editors get more functionality. The example in this text, however, indicates that currently the opposite is true. The more content is encoded, the harder it gets for developers to guarantee even the traditional features. The reason is that current video players are not using any annotation standard to let user access and browse video content. Figure 5 illustrates this thought.

One of the reason for this is that there are still many unsolved problems that research will have to address in the future. With multimedia becoming more than the reproduction of captured sensory input, both multimedia data formats and their infrastructure will have to undergo radical changes.

Many established AI algorithms, for example, require more time than real-time processing (i.e. the processing time exceeds the presentation length of the processed file). Furthermore, many information extraction technologies are not streamable (i.e. the algorithm needs to process the entire file at once). Live transmissions therefore require the development of many new supervised and unsupervised learning techniques. Most algorithms used to extract information from multimedia data are not perfectly accurate. In the past, multimedia formats and their infrastructure were tuned to cope with channel errors (e.g. stalled networks) but will we also be able to cope with source errors in a way acceptable to the user?

When object information is encoded in a video stream, the user will want to be able to operate with these objects, in other words VCR operations are not sufficient anymore. But how will a generic user interface for this look like? How can we guarantee broad compatibility between players and formats? Can one specific player really be providing different views on the objects, depending on the use case or will video players rather be executing programs (in this sense become Turing complete)? In the end, will there still be a distinction between multimedia data and the programs handling them?

Currently, the only solution to many of these problems is to use self-developed applications and non-standard metadata, even for very straightforward tasks. Many areas of research will have to find solutions to problems that range from coding and efficiency issues to architectural and user interface questions. In the end, however, all the solutions will have to work together so that widely accepted standards enable developers to create applications that allow an easy utilization of rich content by the end user on different devices.

Acknowledgements

The experiences of the author reported herein were made during the development of the E-Chalk system. The project was started in 2001 after an idea by Raul Rojas. Many others have also contributed to the development of the system, among them Lars Knipping, Kristian Jantz, Benjamin Jankovic, Ernesto Tapia, Christian Zick, Mary-Ann Brennan, Margarita Esponda, and Wolf-Ulrich Raffel.

References