Experiments on Using MPEG-4 for Broadcasting Electronic Chalkboard Lectures

Benjamin Jankovic, Gerald Friedland, Raúl Rojas
Institut für Informatik
Freie Universität Berlin
[jankovic|fland|rojas]@inf.fu-berlin.de

June 2006

Abstract

The MPEG-4 standard describes a multimedia format that allows to combine vector based data with audio and video. This makes it possible to store and transmit lectures created with the E-Chalk system without any loss of semantics. Utilizing the infrastructure already available for MPEG-4 promises the possibility of getting rid of a proprietary lecture format. This article presents the implementation of a converter tool that transforms E-Chalk lectures to MPEG-4. It shows the current possibilities for the replay of MPEG-4 based E-Chalk lectures and discusses technical problems and principal issues resulting from the use of this already aged standard.

1 Motivation

Lectures held in front of a blackboard can be captured and transmitted in two ways: Either as a video of lecturer and board, or as a set of strokes and images captured by an electronic whiteboard which are rendered with high quality in the remote computer. The downside of encoding board data into a video format is a bandwidth inefficient storage and the stroke data has to be converted from vector format to pixel format. After a lecture has been converted to video, it is for example not possible to delete individual strokes or to insert a scroll event, without rendering huge parts of the video again. Another disadvantage concerns the way most codecs work. Mostly lossy image compression techniques are used that are based on a DCT or Wavelet transformation. The output coefficients representing the higher frequency regions are mostly quantized because higher frequency parts of images are assumed to be perceptually less relevant than lower frequency parts. This assumption holds 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.

Pen tracking devices, on the other hand, capture strokes that can be transmitted and rendered as a crisp image: The strokes can be further processed, for example, using handwriting recognition software. Replay, however, requires the use of proprietary

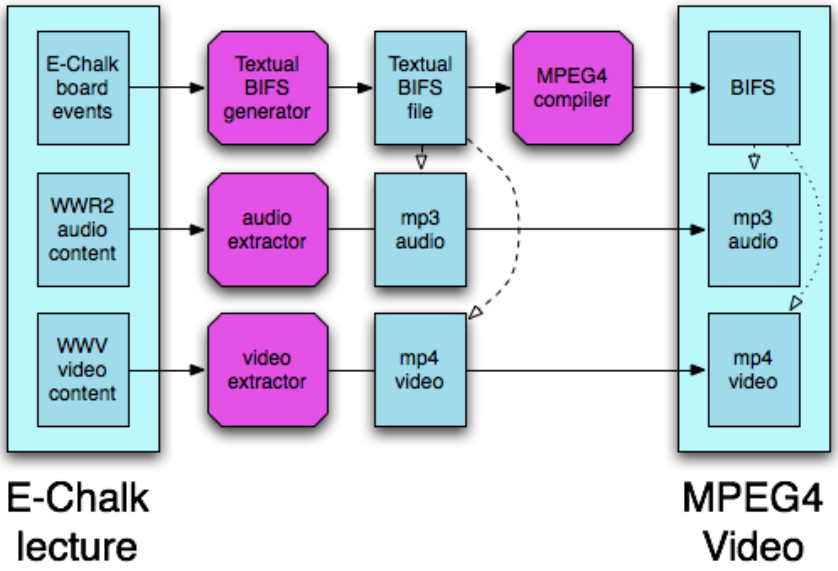

Figure 1: Conversion from E-Chalk format to MPEG-4. First, the audio and video data have to be converted from the E-Chalk proprietary formats to standard formats. After this, the E-Chalk board events are converted to a textual description of a BIFS scene, which is then compiled by an MPEG-4 content authoring tool. The resulting MP4-file contains the converted audio and video files and a binary description of the scene.

client software. To cope with this issue, the E-Chalk system [8] uses Java-based client software [3]. This advantage of this method is that it does not require a download or an install process at the remote side. However, maintaining a proprietary client solution for different operating systems requires a lot of development efforts. The advent of more and more players that support the playback of MPEG-4 promises a change of the situation. Because MPEG-4 supports the encoding of vector based data, E-Chalk lectures encoded as MPEG-4 could not only by watched on any device that supports MPEG-4 (e.g., Apple's Video iPod), the existing infrastructure (like servers, converters, or editors) could also be used to handle E-Chalk lectures. This would leverage the requirement to maintain a proprietary E-Chalk infrastructure for editing and converting archived lectures as well as for live transmissions.

2 Creating MPEG-4 Content

E-Chalk lectures consist of three parts: A pre-segmented video containing the lecturer, the lecturer's comments as audio, and the vector-based electronic chalkboard content. The reason why we would choose MPEG-4 to broadcast E-Chalk lectures is that the format is a world wide standard and defines a way of representing vector-based graphic objects in combination with regular audio and video.

MPEG-4 specification part 11 (ISO/IEC-14496-11) defines a hierarchical and spatio-temporal description of a 2D or 3D scene derived from VRML where updates can be done by changing a node's attributes or by adding, removing or replacing a node at a certain time that is called BIFS (BInary Format for Scenes). BIFS provides nodes for geometric objects that are appropriate for representing strokes and other events appearing in an E-Chalk lecture. Audio and video are easily added using another set of nodes. A few types of nodes cover everything we need to represent an E-Chalk lecture in MPEG-4. So the entire conversion task reduces to translate the proprietary E-Chalk format to BIFS.

The MPEG-4 standard defines a container format called *MP4*. An MP4-file contains all media content like the audio and video stream, images, and a BIFS part that specifies when and where the content has to appear in the scene. As the names implies, BIFS is a binary representation and therefore not human readable. The MPEG-4 specification defines a user editable source format, called *Extensible MPEG-4 Textual (XMT)*. XMT has been defined in two levels: *XMT-A* format and XMT-Ω format. XMT-A is a low-level representation that can be very easily mapped to BIFS while XMT-Ω is a high level format that reuses as a base a subset of the tags defined by SMIL [10].

An MPEG-4 content creation tool gets as input a textual description of the scene, compiles it, and packages it together with all necessary media data to an MP4-file (compare Figure 1). Of course, available MPEG-4 compilers can't handle the E-Chalk proprietary audio (WWR2) and video (WWV) formats. We have written a tool called *echalk2video* that converts E-Chalk's audio and video format to MPEG-4 audio and video.

BIFS uses trees as basic structure. As a consequence, the basic structures are nodes and there are two types of them: Group nodes and leaf nodes. Group nodes can have many children (subtrees) whereas leaf nodes describe objects. Once a node has been defined, its properties can be changed at any time later or the entire node can be removed. Mapping the E-Chalk board events to BIFS is in most cases straightforward. Timestamps are directly supported by BIFS: Every node definition or command can be preceded by an `At <timestamp>` command which has the same semantics as E-Chalk's event timestamps. After an obligatory `InitialObjectDescriptor` which defines some basic parameters such as the video size (in this case the board size), the main scene is described by a group node of the type `OrderedGroup` that forms the root. All other nodes are added to this group node (see Figure 2). The next hierarchy level includes a node that defines the background color (`Background2D`), a `Sound2D` node that links to the audio data, a `Transform2D` node that represents the chalkboard and a `Bitmap` node with a `movieTexture`. The order of the children is important for the rendering, as it defines the layers. In particular, the lecturer's video should come after the board.

2.1 Converting E-Chalk lectures to BIFS

2.1.1 The E-Chalk Board Format

The E-Chalk board event format is thoroughly described in [6], Chapter 4.11. The events are described by a simple line-based, human editable ASCII format. After a few header

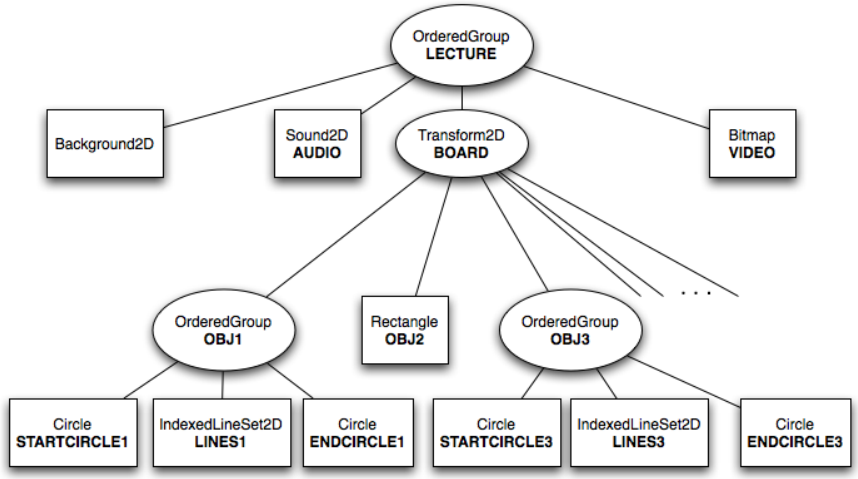

Figure 2: Scene hierarchy of an E-Chalk lecture in BIFS (simplified). A lecture consists of video, audio, and the board. Every object drawn on the board is a child of the `BOARD` node. In this example, `OBJ1` and `OBJ3` represent a stroke composed of a polyline, a start and an end circle. `OBJ2` is an image (a `Rectangle` with an `ImageTexture`).

lines, that provide version information, specify the resolution of the board, the lecture title, and the background color, every event is stored in one line of the following syntax:

```
<timestamp>"$"<event>["$"<arguments>*]
```

The `timestamp` is the hexadecimal coded amount of milliseconds that have gone by since starting the lecture. The character`$` serves as token delimiter. After the mandatory name of the event, a number of parameters can be passed to the event, again delimited by the dollar symbol. E-Chalk knows the following events that are relevant for MPEG-4 content authoring:

- `Form`
  This event marks that something is actually drawn on the board. The next parameter specifies what exactly:
  - `Line`
    This event takes the arguments $\$x_0\$x_1\$y_0\$y_1\$r\$c$ and triggers the drawing of a line segment from point (x_0, y_0) to point (x_1, y_1) with stroke radius r and color c. This event type usually makes up more than 90% of a lecture, as a stroke is nothing else than a set of lines.

    Using a set of lines in BIFS would be highly inefficient. The reason is that every single part of a stroke would create a new separate object to be hung in the scene tree thus making the resulting tree very complex. A better way to represent strokes is a polyline. If the starting point of a line corresponds

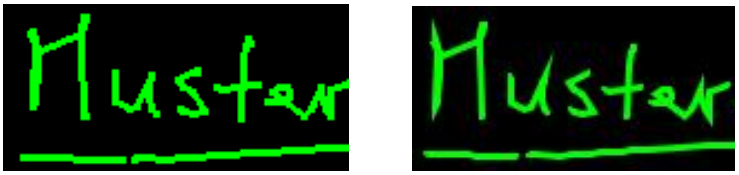

Figure 3: Left: Rendering of the E-Chalk client Applet. Right: Rendering of IBM's M4Play. In the right picture perceptual differences are caused by anti-aliasing and sharp edges due to the rendering of polylines.

to the end point of the previous line, the end point of the new line is added to the polyline. Otherwise a new polyline is started. For this, BIFS provides a node called `IndexedLineSet2D`. It is an array of points connected with straight line segments. We draw a circle at the beginning and the end of a stroke to improve its esthetic appearance. This avoids sharp edges and gives the impression of a round pen tip.

Due to the slightly different representation of a stroke in E-Chalk and in BIFS, the output is not pixelwise identical to the rendering output of the Java based E-Chalk client. The main reason is that MPEG-4 players use a stronger anti-aliasing and draw angles of connected line segments differently. Figure 3 shows a comparison. Finally, the output differs from MPEG-4 player to MPEG-4 player because every player uses slightly different rendering methods.

- `Image`
  This event gets an *id* and two coordinates and inserts image number *id* at the specified position. A forth argument specifies whether the inserted image may be updated. Images are tagged updatable if they actually show screenshots from a Java Applet inserted into the board.

  Images can be directly placed into the board by adding a rectangle object at the appropriate position and then placing the image onto it as a texture.
- `Text` This event takes a MIME encoded [2] string, two coordinates, a color and a font size. After this event, one of several possible events can follow:
  - `Text$Char`
    The event takes a character as first argument. A set of these events is used in between a `Form$Text` event and a `Text$End` event if the text is typed by the user. When the user presses a certain key on the keyboard the event is inserted. Special characters, like `backspace` or `delete` have their own sub-events.
  - `Text$SetTxt` and `Text$Str` both take strings as first arguments and are used when the user sets an entire textline at once using the text history (cursor up and down) or pastes text at the current cursor position respectively.

  Text events can be inserted using a `TextNode`. However, it is hard to guarantee

that the appearance is identical to the board server appearance because Java fonts may have a different appearance compared to the fonts, the MPEG-4 player is using. Typing of text is mapped by consecutively changing the string presented by the text node.

- `Image$Update`
  This event can happen any time after `Form$Image`. It takes two arguments: id_1 and id_2. Applets that are inserted into the board are replayed as consecutive screenshots. The command triggers a replacement of the image with id_1 with the image having id_2.
  Later image updates are directly supported by BIFS as any node can be updated, so Applet replay can be easily implemented.

- `Scrollbar`
  This event takes one integer parameter that specifies the new vertical offset of the board's top position.

  All objects drawn on the board are children of the `BOARD` node and their translation is meant to be relative to the parent's position. The `BOARD` node is of the type `Transform2D`, which has a property called `translation`. Changing the value of this property results in a change of the center position of the group node and thus to a position change of all children.

- `RemoveAll`
  This event clears the entire board and sets the board position back to beginning.

  It can be implemented by deleting the `BOARD` node, which also triggers the deletion of all its child nodes. After this, a new `BOARD` node is defined.

- `Undo`
  This event triggers the Undo-manager to undo the last drawn stroke or inserted image. The event is inserted when the user presses the respective button in the board toolbox.

  `Undo` events can be implemented by deleting the last inserted node. To achieve this, every inserted object gets a unique identifier.

- `Redo`
  This event is the inverse function of `undo`.

  It can be implemented by again defining and adding the last deleted node.

2.1.2 Audio

At the beginning of the lecture replay, a so-called `ObjectDescriptor` is created. It contains information about the audio file to be played back. This `ObjectDescriptor` can now be considered as an audio source that can be integrated in the scene. For this, BIFS provides a node called `Sound2D` with an `AudioSource` property that links to the above `ObjectDescriptor`.

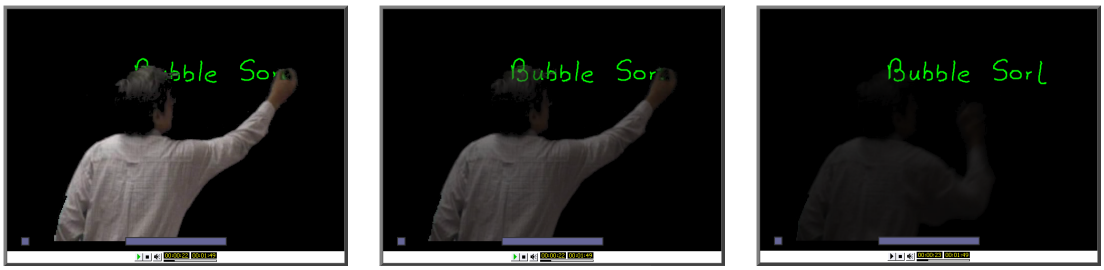

Figure 4: Interactively changing the opacity of the lecturer's video. With a slider it can be faded in and out.

2.1.3 Video

Similar to audio, there is another `ObjectDescriptor` for the video file created at the beginning of the lecture. A node called `MovieTexture` links to it and can be a property of any `Shape` node. In our case we use a `Bitmap` node, because this simplifies optional scaling. In order to match the board content with the lecturer, the `Shape` node's parent is a `Transform2D` node (not represented in Figure 2), to determine a certain spatial offset.

A problem arises, when we want to have the lecturer in front of the board, as there exists no video format supporting transparency yet. To achieve this, the `Shape`'s material property can be modified. Setting the `MaterialKey` node's `keycolor` property to black tells the rendering engine that every black pixel in a video frame should be considered transparent.

2.1.4 Interactivity

Another useful MPEG-4 feature is user interactivity. For example, it is possible to implement a slider that can control the video's overall-transparency. A `TouchSensor` node registers the mouse position in a defined scene area, which is then converted into a value between 0 and 1. This value is routed to the `MaterialKey`'s `transparency` property. The user can fade in or fade out the lecturer (see Figure 4).

2.2 Encoding

For compiling a textual BIFS file, an MPEG-4 encoder is needed, as for example `XmtBatch`, a tool included in the Java-based `IBM Toolkit for MPEG-4` [4]. 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 Open Source GPAC framework

[11] developed at the École 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 prefer the format also for better readability as the bt document architecture is very similar to XMT-A and the syntax is close to *VRML*[5]. GPAC also provides a command line tool named `mp4box` for compiling bt-files.

We decided to use this tool to be able to realize live content creation at a later point of time. Hence the code examples in the next section are in bt format. Unfortunately, incremental scene creations are not really supported by mp4box yet.

2.3 Code Examples

2.3.1 Scene Definition

After the obligatory `InitialObjectDescriptor` (not shown here), the initial scene containing the board, audio, and video is defined as follows:

```
# Root of the scene tree
DEF LECTURE OrderedGroup {
      children [
              Background2D {
                      backColor 0.0 0.0 0.0
              }
              # Define hook for audio node
              DEF AUDIO Sound2D {
                      source AudioSource {
                              # Object descriptor with ID 3
                              url [od:3]
                      }
              }
              # empty board
              DEF BOARD Transform2D {
                      translation 0 0
                      children [
                      ]
              }
              # Define hook for video node (for overlaid replay)
              Transform2D {
                      translation 0 20 # Video offset to better match instructor
                      children [
                              Shape {
                                      appearance Appearance {
                                              # Transparency settings for the video
                                              material DEF M1 MaterialKey {
                                                      keyColor 0 0 0
                                                      lowThreshold 0.1
                                                      transparency 0.1
                                              }
                                              texture MovieTexture {
                                              # Object descriptor with ID 4
                                                      url [od:4]
                                              }
                                      }
                                      # Video isn't scaled now but could be.
                                      geometry DEF VIDEO Bitmap{
                                              scale 1.0 1.0
                                      }
                              }
```

```
                    ]
              }
      ]
}
```

2.3.2 Object descriptors for Audio and Video

```
RAP AT 0 {
      UPDATE OD [
            ObjectDescriptor {
                  objectDescriptorID 3
                  esDescr [
                        ES_Descriptor {
                              ES_ID 3
                              muxInfo MuxInfo {
                                    fileName "<path-to>/audio.mp3"
                              }
                        }
                  ]
            }
      ]
}

RAP AT 0 {
      UPDATE OD [
            ObjectDescriptor {
                  objectDescriptorID 4
                  esDescr [
                        ES_Descriptor {
                              ES_ID 4
                              muxInfo MuxInfo {
                                    fileName "<path-to>/video.mp4"
                              }
                        }
                  ]
            }
      ]
}
```

2.3.3 Stroke Encoding

The first line segment of a stroke (i.e. when a line segment's starting point is not connected to the ending point of the last line segment)

```
3e8f$Form$Line$17c$aa$17c$ab$ff00ff00$3
```

is encoded as BIFS Text as follows (the coordinate systems of E-Chalk and MPEG-4 differ, so the coordinates have to be translated):

```
AT 16015 {
# at timestamp 16015 (after 16 seconds)
      APPEND TO BOARD.children DEF OBJ1 OrderedGroup {
             children [
                    # Beginning circle
                    # (optical correction to simulate pen down)
                    DEF STARTCIRCLE1 Transform2D {
                           translation -81 129
```

```
                        children [
                                Shape {
                                        appearance DEF APP1 Appearance {
                                                material Material2D {
                                                  emissiveColor 0 1 0
                                                  filled TRUE
                                                }
                                        }
                                        geometry Circle {
                                                radius 1
                                        }
                                }
                        ]
                }
                # Line segment
                # Beginning of stroke starts with first line segments
                Shape {
                        appearance Appearance {
                                material Material2D {
                                        lineProps LineProperties {
                                                lineColor 0.0 1.0 0.0
                                                width 3
                                        }
                                }
                        }
                        geometry DEF LINES1 IndexedLineSet2D {
                                coord DEF LINESPOINTS1 Coordinate2D {
                                        point [-81 129 -81 129]
                                }
                        }
                }
                # (provisional) ending circle (simulate pen up)
                DEF ENDCIRCLE1 Transform2D {
                        translation -81 129
                        children [
                                Shape {
                                        appearance USE APP1
                                        geometry Circle {
                                                radius 1
                                        }
                                }
                        ]
                }
        ]
    }
}
```

Further line segments that belong to the stroke are appended consecutively. For example, the line segment

```
3e9f$Form$Line$17c$ab$17c$ad$ff00ff00$3
```

is appended to the last line segment as follows:

```
AT 16031 {
      REPLACE ENDCIRCLE1.translation BY -81 127
      APPEND TO STROKEP1.point -81 127
}
```

until the next line segment does not belong to the same stroke (i.e. it is not connected).

2.3.4 Scrolling

```
2717f$Scrollbar$b
```

is encoded as:

```
AT 160127 {
      REPLACE BOARD.translation BY 0 11
}
```

2.3.5 Image

```
329f9$Form$Image$0$ef$2e6
```

is encoded as:

```
AT 207353 {
    # first, create a new Object Descriptor with informations about the image
    UPDATE OD [
        ObjectDescriptor {
            objectDescriptorID 10
            esdescr [
                ES_Descriptor {
                    es_id 10
                    streamPriority 0
                    decConfigDescr DecoderConfigDescriptor {
                        objectTypeIndication 108
                        streamType 4
                        upStream false
                        bufferSizeDB 66000
                        maxBitrate 0
                        avgBitrate 0
                    }
                    slConfigDescr SLConfigDescriptor {
                    }
                    muxInfo muxInfo {
                        fileName "images/0.jpg"
                    }
                }
            ]
        }
    ]

    # now include a bitmap with the image as texture in the scene tree
    APPEND TO BOARD.children DEF OBJ62 Transform2D {
        translation 16 -559
        children [
            Shape {
                appearance Appearance {
                    texture ImageTexture {
                        url "10"
                    }
                }
                geometry Bitmap {
                }
            }
        ]
    }
}
```

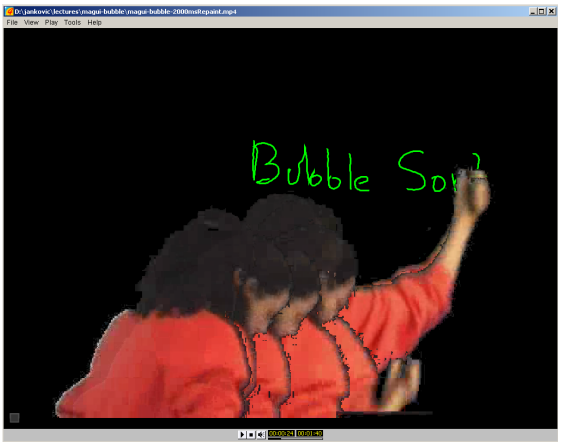

Figure 5: Repainting artifacts with overlaid video in *M4Play*, when nothing happens on the board. The problem is being worked around as described in the text.

2.3.6 Undo

```
297500$Undo
```

is encoded as:

```
AT 2716928 {
      DELETE OBJ215
}
```

3 Playback

One of the biggest problems resulting of storing E-Chalk lectures in MPEG-4 is that the appearance depends on the player used by the viewer. Even though MPEG-4 is an established standard, there are just a few players available that are able to play back BIFS content. Most of the known video players like VLC [9] or QuickTime Player [1] can only handle MPEG-4 video and audio (parts [14496-2] and [14496-3] of the ISO standard). Of course, this supplies most user needs. The reason for this restriction is the immense range of features covered by the MPEG-4 specification. Different profiles and layers are defined in the specification to assure compatibility. Unfortunately, the demand for advanced content (such as vector data) is not very large. However, we did some experiments with three of the most common BIFS-compliant MPEG-4 players.

3.1 Envivio Plug-In

This is actually not a standalone player but rather a plug-in for the QuickTime Player and the Windows Media Player. It is not free anymore, probably due to the MPEG-4 licensing policy. Unfortunately, we were not able to experiment so much with it because it crashes when updates are made on an `IndexedLineSet2D` node. Of course, polylines

are absolutely essential for displaying the stokes on the chalkboard and so the plugin was useless for our purpose.

3.2 Osmo4-Player

Together with mp4box the GPAC project also provides the Osmo4 player. Older versions seem to have a problem with scenes of increasing complexity. Depending on the scene and the underlying machine, it took about 1 minute until the framerate began to drop down noticeably. But the latest version (v0.4.0 at the time of writing this document) can play back a 90 minute lecture (board and audio) at constant 30 frames per seconds without any problems.

However, adding the semi-transparent video of the instructor does not lead to satisfactory results, as the player apparently cannot handle chroma keying and the black part of the video occludes the board.

Another problem is fast-forwarding and rewinding. It is implemented in the player but seems to have problems with complex scenes. Another downside is that the player has to be compiled by the user as it is only available in source code form (due to MPEG-4 licensing politics).

3.3 IBM M4Play

M4Play is a part of the IBM Toolkit. It has a major advantage: It is written in Java and can be used as an Applet. As a result, it is platform independent and there is no need to download and install a special player on the client side. M4Play provides all functionalities needed for E-Chalk lecture replay. However, transparency seems to work correctly only if there is activity in the background. To avoid repainting errors (see Figure 5), an invisible dummy painting operation is generated every 100 ms.

Unfortunately, it is computationally very expensive to calculate the key color transparency in Java. Hence the frame rate drops down to about 5 frames per second (depending on the machine the player is running on). But even if the video is not displayed, the player seems to have problems with very complex scenes. As the BIFS scene of an E-Chalk lecture gets more and more complex, the player again starts dropping frames after about one hour playback and gets slower and slower. Last but not least a rewind/fast-forward operation is not implemented. Aside from these disadvantages, the player is capable of replaying E-Chalk lectures, as can be seen in Figure 6.

4 Comparison

What are the advantages and disadvantages of MPEG-4 compared to the E-Chalk format?

The perceptual difference of the appearance of the rendered chalkboard is nearly unnoticeable even though not exactly the same image is rendered by different players. However, the mapping between E-Chalk format and BIFS is lossless. File sizes also are nearly the same for board-only lectures (usually between 1 and 10 MB for a 90 minute

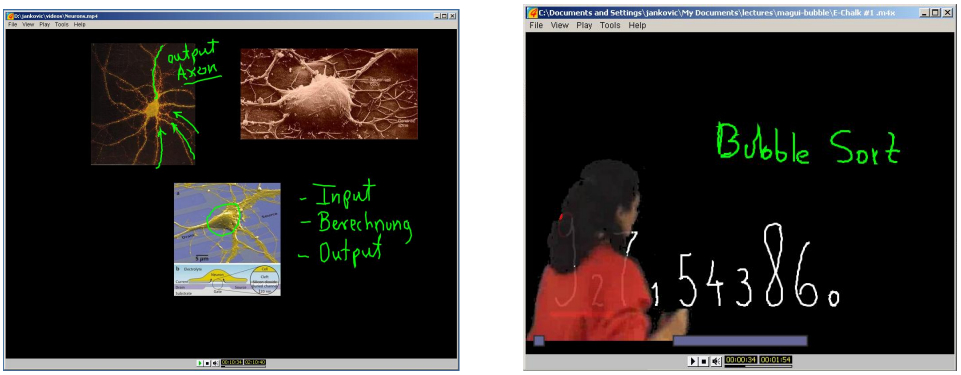

Figure 6: Two screenshots of E-Chalk lectures in MPEG-4 format replayed with IBM's *M4Play*. The left image represents a typical board scene with strokes and images, the right image has an additional overlaid semi-transparent video of the lecturer.

lecture). If an overlaid video is also to be stored, the file size depends on the codec and bitrate used for video conversion. In general, lectures stored in MP4-files are smaller than lectures stored in E-Chalk's proprietary format. The WWV format is quite simple and less optimized. However, the file size range in the same order of magnitude (about 100 - 300 MB for a 90 min. lecture). This makes it possible to playback a given lecture using a DSL connection, no matter which format is used.

An advantage of MPEG-4 over E-Chalk's format is that it offers many features that do not exist in the E-Chalk client. It is very easy to implement user interactivity and strokes can be smoothed very easily by the various available drawing functions. All MPEG-4 player offers scaling without requiring any extra effort. Of course, using a standard like MPEG-4 does not require the maintenance of a proprietary editing and replay infrastructure. Live broadcasting is an essential feature of the E-Chalk client that is not possible (yet) using MPEG-4. One reason for this is that the conversion from WWV format to MP4 cannot be done in real time. Another reason is the lack of support for incremental scene creation in all available MPEG-4 content authoring tools. Another problem is that the scene graph is built up during playback. This makes it difficult to have random seek points for fast-forwarding and rewinding. Although this is an essential feature when playing back E-Chalk lectures, none of the tested MPEG-4 players is currently supporting it.

5 Conclusion

MPEG-4 is a powerful standard with a huge range of possibilities. However, what sounds promising first turns out to be the problem next. It is nearly impossible to implement all MPEG-4 features in one player. For this reason, players can only play back parts of the standard. For most end-user applications implementing only audio and video is sufficient. BIFS content is not widely supported yet and only a few players are available.

Another reason for this may be the MPEG consortium's licensing policy. The MPEG-4 Standard is not free and many developers seem to avoid using it if there is an alternative for their purposes. Furthermore, it will probably take some time, until small devices like PDAs posses enough power to be able to replay complex BIFS scenes.

Hopefully, there will be a higher demand for MPEG-4 video with advanced content like BIFS in the future, so that more and better MPEG-4 players are developed. However, as long as the players do not support random seek and live transmission are impossible MPEG-4 does not represent a true alternative to the proprietary E-Chalk format and the Java-based replay client.

References

[1] Apple Computer, Inc. QuickTime player. `http://www.apple.com/quicktime/mac.html`, 2006.

[2] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. RFC 2046, November 1996.

[3] G. Friedland, L. Knipping, and R. Rojas. E-Chalk Technical Description. Technical Report B-02-11, Fachbereich Mathematik und Informatik, Freie Universität Berlin, May 2002.

[4] IBM Research. IBM Toolkit for MPEG-4. `http://www.alphaworks.ibm.com/tech/tk4mpeg4`, 2006.

[5] ISO/IEC JTC1. Virtual Reality Modeling Language (VRML). ISO/IEC 14772-1, 1997.

[6] L. Knipping. *An Electronic Chalkboard for Classroom and Distance Teaching*. Ph.D. thesis, Institut für Informatik, Freie Universität Berlin, 2005.

[7] F. Pereira and T. Ebrahimi. *The MPEG-4 Book*. Prentice Hall PTR, 2002.

[8] R. Rojas, G. Friedland, L. Knipping, and E. Tapia. Teaching with an intelligent electronic chalkboard. In *Proceedings of ACM Multimedia 2004, Workshop on Effective Telepresence*, pages 16–23, New York, New York, USA, October 2004.

[9] VideoLAN. VLC media player. `http://www.videolan.org`, 2006.

[10] W3C. Synchronized Multimedia Integration Language (SMIL) (last visited: 2006-06-16). `http://www.w3.org/AudioVideo`, 2005.

[11] GPAC Project on Advanced Content (last visted: 2006-06-16). `http://gpac.sourceforge.net/`.