A SOFTWARE SYSTEM FOR MPEG-4 ENCODING OF MULTI-MEDIA STUDIO CONTENT FOR 3D TELEVISION.

Peter Woodward, Yakup Paker, Alan Pearmain

Queen Mary, University of London. UK

INTRODUCTION.

This paper presents a media encoder tool, showing the development of software tools and components leading up to the final implementation of a streaming 3D media transmission codec in MPEG-4 BIFS(1). The work was carried out under the PROMETHEUS project(1). The final project software and associated paper were presented at the IBC in Amsterdam in September 2002(2).

1 - THE APPLICATION REQUIREMENT.

The requirement for the software developed is the capture, display, encoding and transmission of multimedia content, as formed in a studio, using MPEG-4 technology. Additionally, the encoded transmission stream requires a client application to receive, decode and display the content. The content itself consists of two-dimensional video and audio and three-dimensional geometric content. Additionally the content contains timing data used for synchronisation and other control information such as configuration files, initial scene files and viewing parameters. The proposed application is expected to receive the data, from singular or multiple sources, which delivered the content in streams or files. The system is a juxtaposition of conventional audio/video codec application software, a 3D display and manipulation tool and a streaming network client-server application.

2 - THE PLATFORM.

The need of use of existing libraries, and the requirement for conventional media and three-dimensional graphics narrowed the choice to the Performer (2) library or the Java Media Framework (3). The final choice as the JMF for the target framework rested on the cross platform nature of a Java Implementation. The JMF and a three-dimensional graphics Java library, the JAVA3D (4) was implemented on both PC/Win32 and SGI/Unix configurations. The Performer system was only provided on Unix/Linux systems. The final implementation was delivered on a Linux configuration. The final decision to use JMF was vindicated later in the project when key software tools were added to the project by an external partner – the Ottawa University MCR Lab.

3 - THE MPEG-4 ALTERNATIVE.

In the initial phase of the PROMETHEUS project much emphasis was placed on the existing MPEG-4 decoding compositors and their associated tools. This implementation is called IM1. However, there were major problems with the complete usage of these systems. The main problem with the existing tools stemmed from the complexity of the MPEG-4 system. MPEG-4 provides for 3D geometry, for streaming 3D face and 3D body geometry and for 2D video and audio data. The MPEG-4 community did provide implementations of each of these components -for validation of the differing MPEG-4 bit-streams – but they were separate implementations. Each of the separate implementations were built and tested. However, the task of integration of separate audio, 2D video, face and body compositors to implement PROMETHEUS functionality was deemed out of the scope and time-scale of the project. The early experiments using the sample MPEG-4 3D Implementation are described in (5). Some time was spent evaluating the different MPEG-4 compositors. Our judgement was that the MPEG-4 community were focused more on the 2D compositors, the 3D compositors were more experimental. We felt it was better to work with a system under our own control, rather than waiting for the full 3D composer. Therefore we decided to develop our own encoder so we integrated the Ottawa encoder and decoders into the JMF/3D framework built at QMUL.

4 - CONTENT DESCRIPTION.

The encoder application is part of a full three-dimensional television system prototype. In summary, various partners were generating different content types to construct a virtual television studio, with 3D models of actors (avatars) who could be clothed, and whose facial expressions were captured and rendered on the avatars. Additionally it was expected to deliver standard

1 PROMETHEUS is a collaborative project, led by the BBC, for developing an end-to-end chain in 3D television, funded by an EPSRC/DTI grant under a UK Department of Trade and Industry LINK scheme.
television picture streams, that were encoded using MPEG. The camera viewing position was also to be sent to the encoder application. For the delivery of the video imagery and audio streams, file based encodings were anticipated. The 3D geometry was expected in an array of different channels. The scene context, viewing camera context and timing context were to be encapsulated in a single channel originating in a TV studio. The video imagery, containing pictures of a real actor was to be mapped onto a 2D surface in the scene, in a ‘video wall’ type arrangement and was to be transmitted in another stream. The 3D avatar geometry consisting of a 3D composition of a facial expression ‘head’ placed atop a motioned tracked avatar body was a further stream, and finally a clothing simulation generated cloth for wrapping around the complete 3D avatar was a final stream. The four streams were generated independently of each other, but would contain synchronisation data and were to be fed into the encoder application. In the final system only three streams were catered for:- The studio/camera/synchronisation channel, the video mapped 2D actor plane channel, and the 3D avatar composition channel.

Since the channel data was originating from entirely different processes, the requirement for an inter-process distribution standard was raised. The CORBA architecture was chosen as the most appropriate. For the encoder system this meant that the different channels would be encapsulated in a CORBA distribution structure. There would be CORBA sources generating data, and passing the data into storage farms. There would also be CORBA clients attaching to the storage farms extracting the data. The data itself would be the application specific geometry, control or synchronisation information derived from the specific partner’s generation or simulation engine. The content is converted into Java 3D, and simultaneously converted into a MPEG-4 BIFS (1) representation using the Ottawa MPEG-4 encoder system (6). The Ottawa encoder parses VRML (7) content and converts it to Java 3D and to BIFS, the BIFS content can be stored in files or transferred to a decoder application that decodes and reconstructs the initial three-dimensional geometry described in the VRML content.

5 - THE MEDIA FRAMEWORK.

The final delivery application was written entirely in Java using key components from the Java 3D (J3D) library, and the Java Media framework (JMF). The application framework was fitted around the JMF and the key 3D and audio functionality was provided by the J3D library. In the following sections we outline the key aspects of these two libraries. More focus is provided on the JMF as the J3D is integrated as a plug-in module to the JMF.

6 - SUPPORT MODULES.

The JMF plug-in processor paradigm leads to the development of stand-alone functional modules, which can be developed and tested in isolation. Subsequently the plug-in modules are placed into a main driving processor chain that connects the different functional components into a complete processing suite. Following this concept we developed five classes of functionality. These five classes provided the infrastructure of the final encoder application. The components were developed and then tested using sample driving applications. The classes were the renderer, the format, the data source, the codec and the processor. The format and data source provided the channels for each content source, the codec, and renderer were used a plugins, and the processor provides the controller chain for the whole system.

Format Support Module.

The format component is used by the JMF as an attribute of a media source in order to connect the plug-ins together. A renderer can expect to render a given audio or video format, a decoder can expect an input format and produce an output format. A multiplexer can composite audio and video formats. Formats therefore provide the JMF system chain with the exact connection orders of plug-in modules for given media channels within a specific processor/plug-in configuration. As mentioned above the system was expected to receive three geometry channels for processing. The scene/viewpoint/synchronisation channel from the studio – hereafter called the studio channel. The 2D video mapped actor surface for placing within the studio scene – hereafter called the actor plane channel. The combined high-resolution head and body avatar geometry – hereafter called the mesh channel.

The first stage was to build a channel for handling customised formatted data. Initially we built a ‘text format’ -which was expected to receive all the data. The scene geometry was provided in VRML format. The initial scene data was to be provided in text format, however, all the other data was application specific and would be passed between the generation system and the encoder system using CORBA buffers. Once the text format pattern has been established, we designed and implemented a binary format for the transmission of encoded geometry through the system. Using the CORBA binary format as a generic format, we implemented three instance types of CORBA format for channelling the studio channel data, the actor plane channel data and the mesh channel data.

Data Source Support Module.
The format component is an attribute of a source of media content. The format gives an encoding to a particular chunk of media data. The media data may be contained in a file or streamed in real-time from a media acquisition device. The channel that reads stored data or transports streaming data in a specific format in the media framework is called a Data Source. A data source consists of a reading stream process and a data source manager that handles the stream and provides the external interface to the stream for the media framework. The media framework is required to create a data source from a media location – such as a file or network resource. Once a data source location has been properly created, the media framework controller interacts with the raw stream via the data source to connect and begin the reading and processing of the data identified in the media location.

The JMF provides both a ‘push’ and a ‘pull’ paradigm for data source types. The push type data source is suited for streaming servers and the pull source for clients. The encoder was seen, as a streamer to the decoder. The initial design decision to use the push data source was taken in this context. Whichever type of stream chosen, the stream itself carries out the actual reading of data. Since the application interface was CORBA based an implementation of a CORBA client and server was required to form the bridge from the external data acquisition processes to the encoder processing system. Our partners at the BBC Studio Group implemented the Java interface to CORBA for the external connections, being one side of the ‘bridge’, and we implemented the Java CORBA to JMF connection, the other side of ‘bridge’. The CORBA server stores data in buffer farms, and the CORBA client receives data from the farms synchronously or asynchronously according to the generation conditions. These two parts provided the connection –theoretically over a wide area network – from distributed data gathering sources to a single multi-threaded encoder engine destination.

All that was required of the stream to function were the correct format, a location of media content and a reading function for extracting data from the buffer farms. The CORBA Stream has a correct format set, for mesh, studio or actor data. It has a valid media location – effectively a reference to a CORBA server farm channel. The reading function for the stream initially creates the connection to the remote CORBA service, and the in subsequent reads, receives fresh updated data into it’s own localised buffer data objects. The Data Source reader time-stamps each buffer object with a time derived from a local clock. Alternatively the time stamp extracted from the studio channel can be used as a precursor to proper synchronisation between the different channel. Once the data has been extracted via CORBA into the media framework the buffers containing 3D geometry or control data are passed to the encoders and renderers.

**Encoder Support Module.**

The data read into data sources is encapsulated into media framework buffers. The buffers are passed from data source through the plug-in modules connected in the processor chain. The encoder receives CORBA Format data of one of the three expected types, mesh, actor plane or studio format. The encoder processes the data source by extraction of the necessary data from the buffers and then encoding this data. The encoded data is then sent to the decoder. This extraction/encoding/transmission mechanism is to be applied to the 3D mesh data and the actor plane data. However, time limitations meant than only the studio format channel buffers was handled by the encoder system.

For the studio data, the data source recovers a VRML scene file, parses, encodes and displays the content J3D renderer. The scene contains a viewing camera, replicated in the J3D scene graph. Additionally, the MPEG-4 representation of the initial scene also contains a viewing camera node. The MPEG-4 scene and viewpoint are immediately transmitted to a decoder which displays the scene. As the processing phase of the encoder progresses the updates to the scene are extracted, applied to the encoder 3D scene and also encoded into BIFS and sent to the decoder where the geometrical data is extracted and applied to a decoder 3D scene.

**Renderer Support Module.**

The renderer is a final destination for a data source and it’s associated buffers. The renderer implementation initially creates a complete viewing Java3D scene graph. When the scene file is read and converted to Java3D/BIFS the Java3D nodes are added as content into the pre-existing viewing graph. The viewing updates are applied in the encoder, which simply changes the local viewing position in the renderer. The renderer also provides the interactive facility for localised (i.e. non-CORBA) updates. As the user manipulates the mouse the actions are translated into transformations and rotations, these transformations are virtually identical to the viewing updates from the CORBA source. They are applied locally to the encoder J3D scene viewpoint, and encoded to BIFS and transmitted to the decoder.

**Processor Support Module.**

The final part of the encoder application is the processors. The processors are controllers for the content generation streams. The processors used for the encoder system described are generic processor provided by the JMF. They are built and configured...
from custom formatted data sources, and consist of the chains of data sources, encoders and renderers described above. Once the processors are started, they in turn start the data source streams processing to begin and the data starts to flow through the entire system.

7 - THE ENCODER APPLICATION

The complete encoder application is conceptually nothing more than a system of configured processors, coupled with data channel support data. The management and configuration of the processors is handled by an executive which links the processors together. The encoder application has a user interface to manage the building of the channel stream. The figure below (Figure 1) gives a schematic diagram of how each of the modules described here fit in the entire encoder application. Additionally a screen grab of the encoder application is shown, this shows the user interface coupled with a high resolution head model used during the project.

![Figure 1 – The QMUL JAVA 3D/JMF/MPEG4 Encoder](image)

8 - CONCLUSION

The QMUL encoder/decoder system demonstrated an interface to a television broadcaster virtual production system. The software did suffer performance limitations in the chain from the studio through to the encoder. The latencies were unacceptable for a live broadcast scenario. Due to the limitations of time at the final project integration setup it was impossible to properly identify the location of the bottleneck. We confirmed that the studio CORBA to encoder CORBA transmission was not the cause of the delay. This is reassuring because, at first consideration, the studio-to-encoder link was expected to be the most likely bottleneck. The latency is therefore located in the modules that interface CORBA to the JMF within the encoder system. For pre-recorded updates that were read from file the delay in processing was in the order of 5 to 10 seconds. For updates that were streamed in real-time the latency rose dramatically to 45 to 60 seconds. In both live and recorded updates the encoder updates suffered from bursts of processing. The two problems of ensuring smooth and timely streamed updates still remain. In these two areas further development needed. Further development is also necessary on synchronisation of the various data channels and on encoding of the 3 dimensional mesh.

The complete PROMETHEUS is fully described in the paper presented at the IBC in September 2002. (8).

REFERENCES.


