Audiovisual rendering engine and interface for linear broadcast production

Deliverable D4.6

ICoSOLE identifier:	ICoSOLE-D4.6-iMinds-AVREngineLinearBroadcastProd_v04.docx
Deliverable number:	D4.6
Main author of Deliverable:	Philippe Bekaert (iMinds), Dave Lincoln (TaW), Dave Marston (BBC)
Internal reviewer:	Werner Bailer (JRS)

Work package / task:	WP4 / T4.4, T4.5
Document status:	Final
Confidentiality:	Public

Version	Date	Reason of change
1	2016-10-19	Input on live editing system
2	2016-11-09	Version for internal review
3	2016-11-10	Final version
4	2016-11-21	Added multi-panoramic video capture system API

The work presented in this document was partially supported by the European Community under the 7th framework programme for R&D.

This document does not represent the opinion of the European Community, and the European Community is not responsible for any use that might be made of its content.

Table of Contents

1 Executive Summary

This document describes the live editing application and the audio rendering tools for linear broadcast playout.

The live editing system allows the real-time manipulation of a broadcast mixing processor (engine, see D5.3) via an easy-to-use graphical user interface (the editing application). The engine is able to connect elementary I/O streams of various sources and mix those sources to one or more destination I/O streams. The editing application provides a global view on the state of the mixing engine, and provides the means to manipulate this state. The user interface is capable to run remotely in an off-site location (centralized studio infrastructure).

The live editing system can control, and receive video streams from, the multi-panoramic professional video capture system described earlier in D3.2. This document describes the different levels of API developed for that multi-panoramic video capture system.

The audio renderer converts audio with its associated metadata into audio signals that are suitable for playout, or into a format suited for a particular application or delivery system. With object-based audio (i.e. any audio that has associated metadata that describes it), it will require rendering in many places in the capture to final playout chain. These include rendering in production to ensure the mix sounds correct, and in the final playout for the home user. The BBC has developed software libraries for linear broadcast playout. There are both C++ and JavaScript libraries, allowing audio rendering on the broadcast side and also client-side rendering in the web browser.

2 Introduction

2.1 Purpose of this Document

This document describes tools for linear broadcast playout.

2.2 Scope of this Document

This deliverable covers the live editing application, the multi-panoramic video capture system APIs and the audio rendering tools.

2.3 Status of this Document

This is the final version of D4.6.

2.4 Related Documents

- D3.2 Professional Audio and Video Capture
- D4.4 Tools for content compositing and user interface for live playout
- D5.3 Broadcast playout engine and audio playout libraries

3 Live editing application for linear broadcast

3.1 Scope

The live editing system allows the real-time manipulation of a broadcast mixing processor (engine, see D5.3) via an easy-to-use graphical user interface (the editing application). The engine is able to connect elementary I/O streams of various sources and mix those sources to one or more destination I/O streams. The editing application provides a global view on the state of the mixing engine, and provides the means to manipulate this state. This includes addition/removal of input sources, real-time configuration of mixing pre-sets and transitions, and configuration of output streams. All inputs and the program output (mix output) can be monitored via low-latency previews directly in the application. The actions carried out by the editor can therefore be observed immediately, directly within the application. The user interface is capable to run remotely in an off-site location (centralized studio infrastructure).

3.2 Technical architecture and implementation

An overview of the GUI and engine architecture is provided in Figure 1. The user interface and engine exchange data via TCP-based zeroMQ sockets (for control commands) and via UDP single cast RTP streams (audio and video). This allows the engine and user interface to run remotely (see Figure 2). The user interface uses the same low-level components as the engine for media processing and stream parsing: The GStreamer multimedia framework (http://gstreamer.freedesktop.org). While GStreamer is used to read, mix and write streams in the Linux mixing engine, the user interface uses GStreamer's OS X platform features for high-performance and smooth display of video streams. In particular, we take advantage of the recently-added CoreAnimation sink types to create the best possible platform experience to the application user. The cross-platform GPU graphics API OpenGL has been used on both components to accelerate the processing and display of video streams, and to efficiently use hardware available in commodity devices.

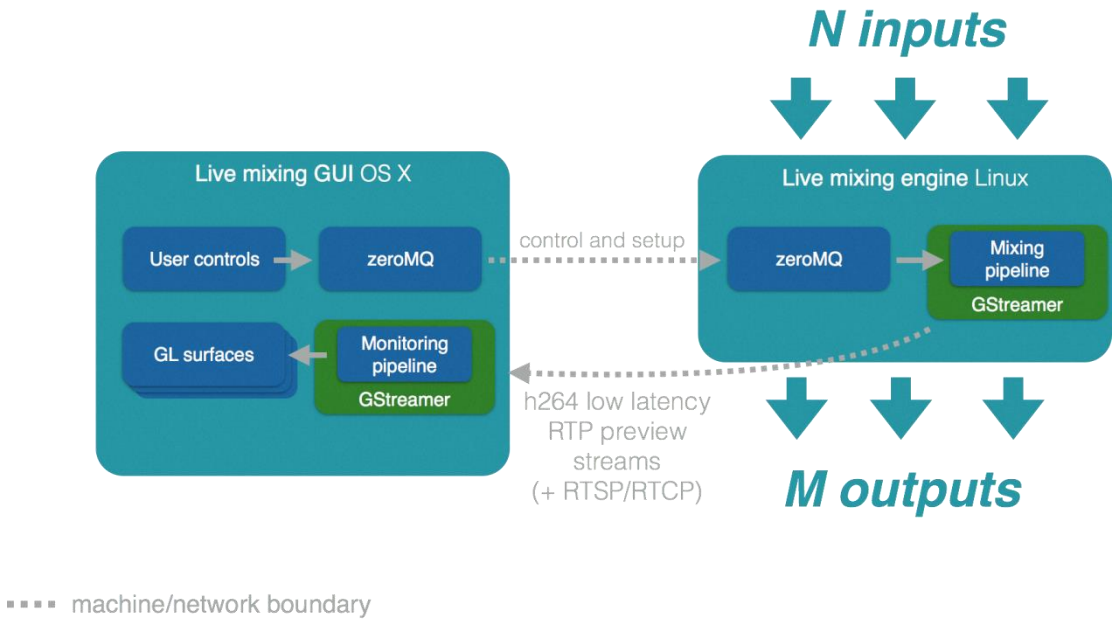

Figure 1: Live editing engine / GUI architecture overview.

3.2.1 Synchronisation aspects of engine and GUI

The user interface and engine are built on identical synchronisation mechanisms. What this means in practice, is that we had to make sure that all buffers previewed in the editing application at the same time, also correspond to buffers that are mixed at the same time instant in the mixing engine. This is achieved by using the RTCP stack that is responsible for the clock synchronisation of the RTP preview streams, and the GStreamer network clock that is optimised for synchronisation over unreliable

networks (i.e. internet). Note that internally, the mixing engine still uses PTP for high-accuracy synchronisation of professional equipment.

3.2.2 Multi-panoramic capture system camera control

An important further integration is the ability to control iMind's multi-panoramic capture system directly from TaW's live editing interface using the RESTful API developed by iMinds. To this end TaW added additional functionality to the live editing interface in order to:

- Map one or more virtual "cameras" to a given "screen" and allow the operator of the live editing interface to switch between the cameras;
- Allow the operator to pan, tilt, rotate and zoom the camera using the mouse in the live editing interface.

In practice this meant using, for example, 3 "physical" HDMI inputs, each mapped to 3 "virtual" cameras, and the operator could switch each "physical" input between the 3 cameras, thus giving control over 9 "virtual" cameras. At any one time three cameras could be viewed live in the editing interface, controlled by the operator (roll, pan, tilt & zoom) and taken on air.

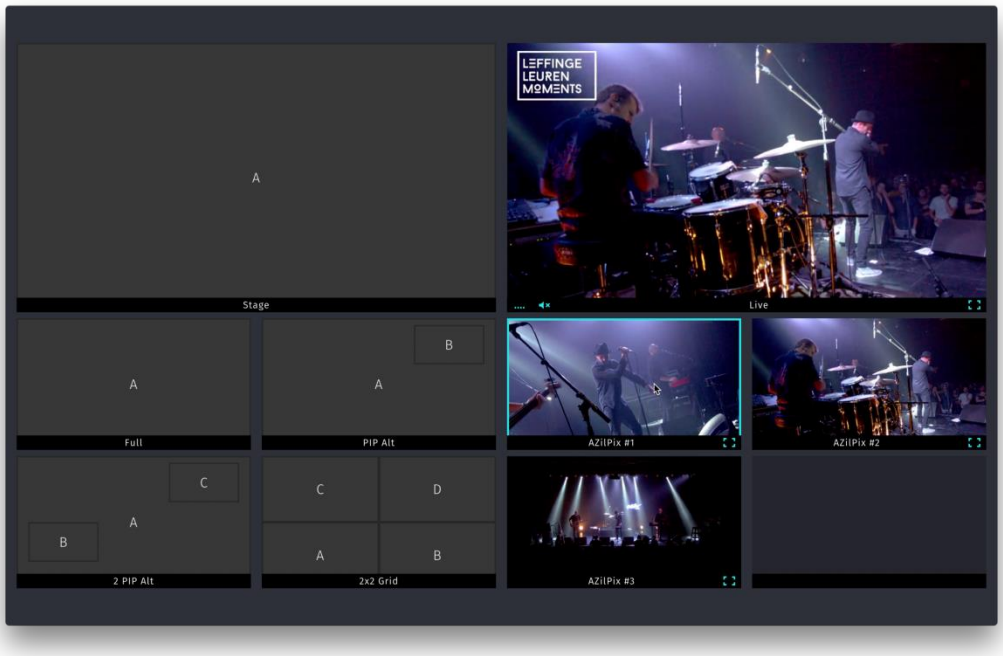

Figure 2: Live editing interface with three camera inputs

3.3 Results

3.3.1 Leffingeleuren Field Trial

At the Leffingeleuren field trial we demonstrated live mixing and camera control of 9 "virtual" cameras from iMinds's multi-panoramic capture system, both locally at the field trial and remotely with the live editing interface on the IBC floor in Amsterdam, in addition to the integration with VRT's Wall Of Moments, BIT's DASH encoder and JRS's UCG streaming app. Figure 3 shows the setup diagram and Figure 4 shows the editing application in action.

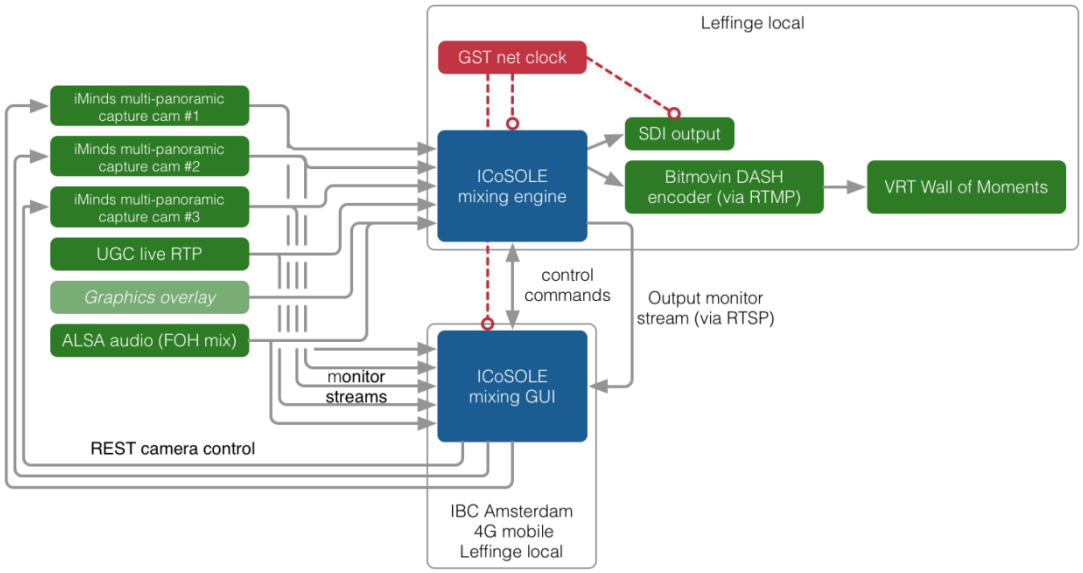

Figure 3: Professional capture production chain

Figure 4: Live editing interface being demonstrated at IBC

4 Multi-panoramic video capture system API

In order to enable (local and remote) control of the multi-panoramic video capture and processing engine, two levels of API have been developed: a low level API, and a RESTful web API. TaW implemented support for the RESTful web API to enable remote control of the multi-panoramic capture system from their mixing engine, as described in the previous section.

The low-level API has been developed in the context of the ICoSOLE project, by iMinds, based on numerous field tests and demos which allowed to abstract and synthesize a simple, yet powerful, set of commands and queries, using low-latency low-overhead inter-process communication by means of a mechanism named “FIFO queues” or “named pipes”.

The RESTful web API has been designed and implemented by AZilPix, the startup company founded in order to commercialize the multi-panoramic video capture system under the product name of Studio.One. It provides a web interface to the low level API.

Rather than designing ad-hoc interface consoles (with buttons, joysticks, T-bars or other elements familiar to broadcast operators), we decided to make Studio.One an open engine, from ground up. Studio.One can be used stand-alone, by means of a basic keyboard and mouse interface, but we understand from prospective users that it will most often be integrated in a production environment, such as TV-studios, concert halls, convention centers and surgical operation rooms, where it is desired to control not just Studio.One, but also other equipment (sound, light, effects, screens, ...) from a single interface. Studio.One therefore makes integration with automation and interfacing systems easy, rather than imposing its own interface console.

4.1 Low level API

This section describes the low level API : the communication mechanism, the command and settings format, and a concise overview of the commands and settings themselves.

4.1.1 Communication mechanism

The AZilPix Studio.One system can be controlled by writing ASCII text command or settings lines to a special file we call the command FIFO. The command FIFO is not a regular file, but a shared memory connection between client applications and the studio one system. It does have a file name, and appears in directory listings, like any ordinary file. The command FIFO is uni-directional: the client application writes commands to it. The Studio.One system reads them.

The Studio.One system communicates settings back to the client application by means of a second special file, we call the return FIFO. Like the command FIFO, the return FIFO is a uni-directional shared memory connection. However, it is written by the Studio.One system and read by the client application.

Per-frame camera metadata can optionally by communicated by the Studio.One system to a client application via a third special file, we call the metadata FIFO.

4.1.2 Command and settings format

Each setting or command is plain ASCII text, one line per command or setting

Settings and commands with argument have a name starting with a lower case letter and ending in =, followed by one or more arguments. For instance:

```
rgbGain=4.1 -0.2 3.6    # sets Red, Green, and Blue gain in dB
frame=1000      # jump to frame 1000
capturePath='/media/Projects/surgie'        # sets the capture path for subsequent recording
```

Commands without arguments have a name starting with an upper case letter. For instance:

```
DarkCalibrate      # performs dark level and noise calibration procedure for all cameras
```



```
Quit                    # terminates Studio.One

Exception:
?     lists all settings
```

Camera or view settings or commands are prefixed by camera<n> resp. view<n>. For instance:

```
camera3.rbGain=4.1  5.2              # sets camera 3 hardware red and blue
channel gain in dB
view6.ListPresets                          # lists view (virtual camera) 6
presets
```

4.1.3 Command and settings overview

Studio.One low level API implements 180 commands and settings requests in the following 11 classes:

1. **System:**

Control protocol version, FIFO names, listing settings and terminating the system.

2. **Recording:**

Setting recording file name and starting/stopping recording

3. **Playback:**

Toggling playing/pausing, setting playback speed, retrieving the number of frames, setting and retrieving a frame range, jumping to a given frame.

Setting animation mode:

- Normal: active view and virtual camera pan-tilt-roll-zoom can be modified at will without storing
- Replay : replays previously programmed virtual camera cuts and motion
- Edit cuts / pan-tilt-roll-zoom / or both : allows to set the active view and/or modify virtual camera orientation and zoom, storing the actions as a key frame on the time line

4. **Exporting:**

Set path, filename, and video encoding quality parameters for exporting an edit to h264 video.

5. **Screens:**

Video output configuration. Studio.One renders up to 9 views (virtual cameras) out of 1 up to 32 physical cameras. For instance, two different rectilinear views can be generated out of a physical camera with a fish eye lens mid-front stage during a concert. The first view shows the singer of the band. The second one the lead guitarist or the base player. Parameters affecting view rendering, as well as keyframing options, presets and transitions are described in section 11.

Three of these views can be assigned a special meaning:

- The primary program out view, or active view.
- A secondary program out view, or alternate view.
- The edit view: typically, a view being prepared before being displayed on the program outputs.

AZilPix Studio.One outputs views onto 1 up to 4 screens:

- Screen 0 is the operator screen, showing a mosaic of all views, or the edit view, with or without overlays
- Screen 1 shows the edit view if there are additional program output screens, or the active view, if it is the only output besides the operator screen.
- Screen 2 and 3 are program outputs, showing the active and alternate view respectively

Each of these screens can display either conventional rectilinear video, or panoramic (Virtual Reality, VR) video.

6. Rendering:

First, raw images from the cameras are corrected, denoised, bayer demosaicked and converted to the rendering system color space. Camera image preprocessing parameters control this, together with the per-camera software color correction parameters.

Next, camera images are warped and blended in each view to be rendered. Warping is the process of geometrically transforming a raw camera image, e.g. a fish eye image into a linear perspective view. If more than one camera is enabled in a view, the enabled cameras images are blended together automatically, providing a stitched view if the cameras are properly aligned. Warping and blending parameters control the process.

Eventually, the resulting warped and blended image is post-processed, using post processing parameters. This includes post-processing shading corrections and image enhancement. Shading happens according to the well established American Society of Cinematographers (ASC) Color Decision List (CDL) scheme. Image enhancement includes a simple unsharp masking for sharpening, but also a state of the art enhancer based on the bilateral grid filter.

A range of guides and image monitors can be applied to the edit view and displayed in overlay on the operator screen. Information display modes, guides and monitors are controlled by their own set of parameters. Guides and monitors include

- Over-exposure indicator
- Focus peaker
- Grids
- Raw camera histrogram
- Rendering color space histogram
- RGB and lightness waveform monitor
- Vectorscope
- Overzoom indicator

7. Scenes:

In order to facilitate live video production, Studio.One allows real camera and virtual camera (“view) parameters such as camera gain, shutter time, view pan-tilt-zoom-roll parameters, and lens focussing distance and iris settings to be stored in presets. A corresponding set of camera and view presets is called a scene. Studio.One provides commands to storing and recalling scenes, and making transitions.

8. Keyframes:

Keyframing is the way to program edits in post production mode (when playing back a recording). Studio.One provides commands to store cuts and camera motion keyframes on the time line (tagged by time code), and to remove them, or copy-paste them, and to query the list of keyframes.

9. Tally:

Cameras can be equipped with a tally light, indicating which camera(s) is (are) active at any time as a means of feedback to actors being filmed. The tally lights can turned on or off from the low level API.

10. Cameras:

Physical camera commands and settings include

- Camera calibration: extrinsics, intrinsics, lens distortion, vignetting, color calibration settings, and dark field and noise profile calibration.
- Camera device control : master gain, red/blue gain (white balancing), shutter time, analog gain (sensitivity)

- Software shading correction: typically used to software tune away color and intensity differences between cameras in post production. During live operation, it is recommended to leave theses settings to neutral value, and use device control instead.
- Lens control: cameras are equipped with canon ef photography lenses and a controller we designed ourselves, allowing software control of focussing distance and iris, and reading back the zoom setting (focal distance) during live operation.

11. Views:

Views are usually rectified cut-outs from the image of one of the physical cameras. We also call them “virtual cameras”. However, multiple physical cameras can be enabled in a view, activating our live stitching rendering by means of warping and blending.

Studio.One provides commands to

- Activate/desactivate physical cameras in a view
- Control software pan, tilt, roll and zoom
- Control blending

4.2 RESTful web API

The RESTful web API, designed and implemented by AZilPix (not on ICoSOLE funding), enables a slightly higher level of control via the web. It is implemented in the form of a software agent running as a separate process next to the video capture and processing engine, on the same server.

The RESTful web API is self-documented by means of a so called swagger user interface, as is the common practice for such interfaces.

The RESTful API was supported by TaW in their video mixing engine (section 3 of this document), and by AXON BV, one of the co-founders of AZilPix, in their cerebrum product for automation control. Cerebrum allows to quickly implement interfaces (desktop, tablet of consoles with buttons, joysticks, T-bars, etc...) for a wide variety of broadcast related equipment and systems, including Studio.One now.

Figure 5: Studio.One demonstrated at IBC 2016 with control from AXON cerebrum, also using the RESTful web API.

5 Audio Rendering for Linear Broadcast Playout

An audio renderer converts audio with its associated metadata into audio signals that are suitable for playout, or into a format suited for a particular application or delivery system. With object-based audio (i.e. any audio that has associated metadata that describes it), it will require rendering in many places in the capture to final playout chain. These include rendering in production to ensure the mix sounds correct, and in the final playout for the home user.

The BBC has developed software libraries for linear broadcast playout. There are both C++ and JavaScript libraries, allowing audio rendering on the broadcast side and also client-side rendering in the web browser.

The principle behind the rendering libraries is to use the Audio Definition Model [1] to describe the representation of the audio content and so enable rendering to the required output formats.

5.1 Audio Definition Model

The Audio Definition Model (ADM) is a meta-data model standardised by the ITU Radiocommunication Sector [1]. This is format-agnostic approach allows for a multitude of audio content formats to be represented with the flexible ADM model, including channel-based, object-based and scene-based (higher-order Ambisonics) audio signals or combinations thereof. The rendering tools are then designed to adapt the given representation to the target reproduction system(s), but in theory the content and reproduction formats are independent.

The ADM can be used to represent complex audio scenes within the ICoSOLE system in a format-agnostic manner, which can then be rendered to a multitude of output devices and formats. As an example, the content could be represented as a 13-channel surround-sound-with-height signal plus several discrete sound objects with time varying positional meta-data. This could then be rendered to a 13-channel surround-sound-with-height loudspeaker layout, plus a 5-channel surround layout, stereo and a binaural headphone reproduction. Thus a single audio scene representation allows delivery to conventional broadcast formats plus novel immersive home theatre systems and immersive headphone listening on mobile devices.

There is an interaction between the content and reproduction representations, which can affect the quality of the listening experience as found in subjective evaluations [2] and there is room for further developing rendering algorithms to improve adaptation.

5.2 C++ libraries

A set of C++ libraries (the BBC Audio Toolbox) has been created during this project to enable audio rendering for linear broadcast playout, as well as in native interactive applications. A key component of the libraries is support for handling the ADM and also reading and writing to Broadcast Wave Format files that contain ADM meta-data in XML [3] (bbcat-adm and bbcat-fileio). This enables production, storage and playback of complex audio scenes.

The key libraries related to this deliverable are the digital signal processing library (bbcat-dsp) and the rendering library (bbcat-render). These respectively contain core DSP components for rendering and the rendering algorithms for converting ADM representations to loudspeaker or headphone signals, where possible creating an immersive 3D impression in the reproduction.

The following rendering techniques are implemented in the C++ libraries:

- Object-based rendering to flexible loudspeaker layouts via vector-base amplitude panning (VBAP) [4] and all-round Ambisonic panning (AllRAP) [5].
- Higher-order Ambisonics (HOA) rendering to flexible loudspeaker layouts using all-round Ambisonic decoding (AllRAD) [5].
- Object-based rendering to headphones using binaural synthesis with head-related transfer functions [6], [7].
- Object-based rendering to headphones using a combination of the above loudspeaker rendering techniques with binaural room impulse responses [8], to give a virtual listening room impression.

- HOA rendering to headphones using decoding to an array of virtual loudspeakers, followed by binaural synthesis. By using an ideal sampling of the sphere, such as the t-designs proposed in [5], an optimal decoding can be achieved.

Channel-based signals are rendered using the object-based rendering techniques, with static position data. For binaural rendering, it is possible to transform the sound scene in real-time based on the listener position and orientation, which has applications to virtual reality systems but for linear playout this is not relevant.

Several of the libraries are available open-source[1] to encourage usage of the ADM and implementation in standard audio production tools. The rendering algorithms and DSP components that are core to the rendering are not released.

5.3 Binaural Signal Processing

The core signal processing units for binaural rendering are a dynamic convolution engine and a variable fractional delay interpolator. These components allow rendering of moving sources in 3D space to a headphone signal with low-latency and avoiding signal discontinuities due to filter switching. Dynamic listener movements are also possible in this system e.g. when using a head tracker. The architecture of the binaural renderer is shown in Figure 6.

The binaural impulse responses from the HRTF database, and separate onset delays, from the time-of-arrival (TOA) database, are loaded using the AES69 [9] format, which provides the positional meta-data corresponding to each measurement. This allows for personalisation of the binaural rendering, loading a listener's own head-related transfer functions (HRTF), or otherwise a content creator's preferred filter set.

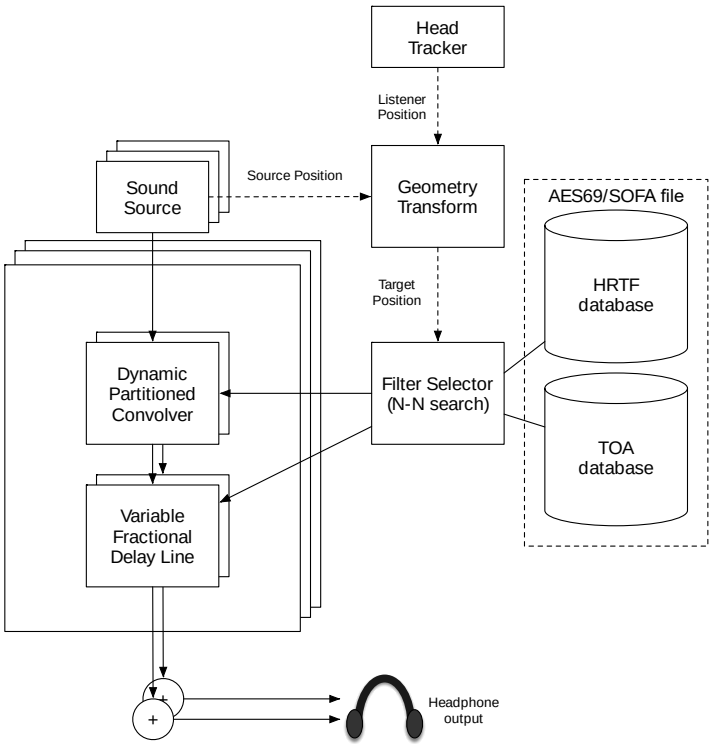

Figure 6 - Binaural Rendering Structure

The convolution engine is a uniform-partitioned fast convolution engine, which allows rendering with long impulse responses and click-free dynamic impulse response switching using crossfading. Separate

[1] BBC Audio Definition Model software: www.bbc.co.uk/rd/publications/audio-definition-model-software

real-time insertion of the onset delay is performed using a variable fractional delay line. This reduces comb-filtering artefacts that would occur by cross-fading between impulse responses with different onset delays and also conveniently allows personalised adaption of the inter-aural time difference, a first-order personalisation related to head dimensions [7].

Another key component is the nearest neighbour search which is performed using a kd-tree [10], this allows the nearest measurement to the target rendering position to be used. Modern binaural filter sets are measured at a resolution equivalent to just-noticeable differences in source position, so nearest-neighbour interpolation is adequate.

5.4 JavaScript libraries

Some of the rendering functionality in the C++ libraries has also been implemented within JavaScript libraries for client-side rendering within a web browser. This can be used for interactive applications but also adaptive rendering of linear content, rendering to the reproduction system connected to the user's device, which could be for example stereo, 5-channel surround or headphones. Moving the rendering to the client-side allows maximum flexibility in the reproduction formats, for example compensating for non-standard changes in loudspeaker layout or applying a personalised binaural filter set for the user.

Rendering to loudspeakers and headphones has been implemented in a JavaScript library (bbcat-js). VBAP and binaural synthesis are supported in the library for rendering object-based signals to loudspeakers and headphones respectively. In addition, rendering of first-order Ambisonics to headphones using binaural processing is supported.

5.5 BBC Renderer

A highly configurable application named the BBC Renderer has been created to utilise the BBC Audio Toolbox libraries and allow rendering of Audio Definition Model content representations to given target loudspeaker or headphone systems. The application can be run in real-time or offline e.g. for batch processing of files, it also provides useful analysis features such as true peak and loudness meters [11]. It can also be configured to record ADM BWF files based on the input signal.

The application provides several control interfaces for setting the scene parameters in real-time. A JSON-based renderer control protocol has been implemented, as well as an Open Sound Control interface. This enables control from remote applications such as a digital audio workstation or a real-time stream within IP-studio.

Figure 7 shows the BBC Renderer application as configured to render an ADM BWF file to a binaural signal.

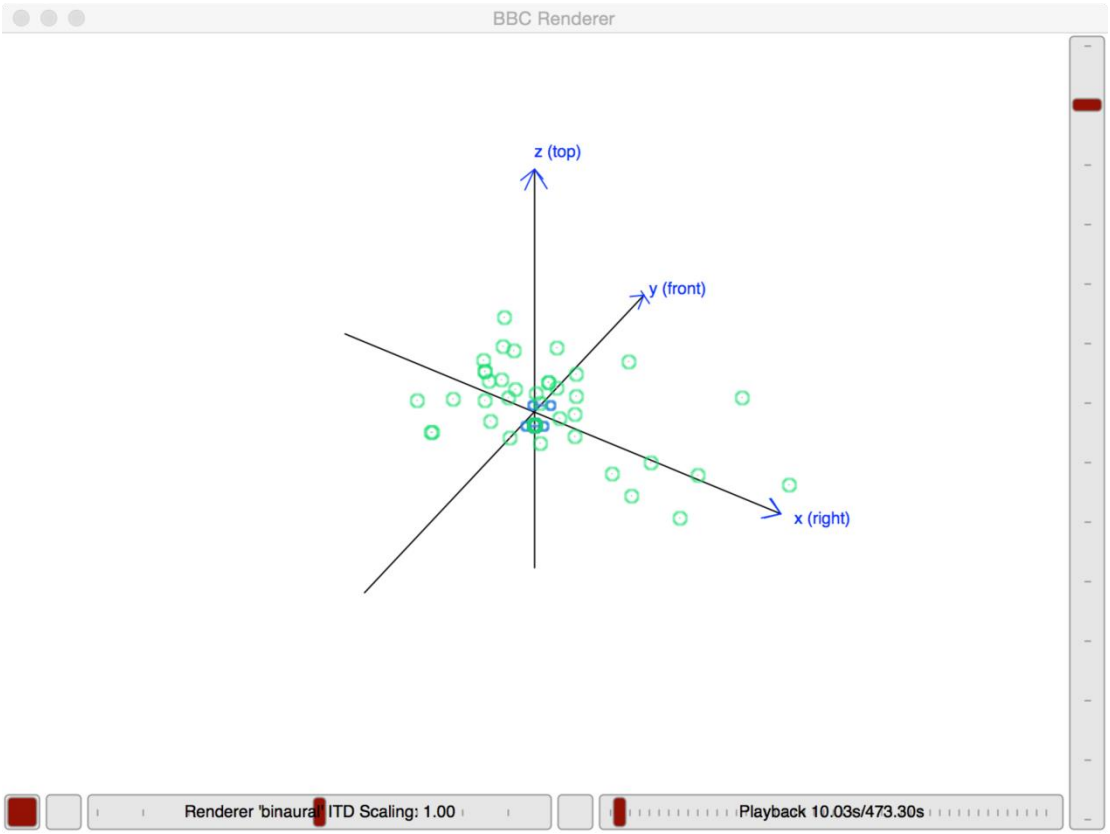

Figure 7 - BBC Renderer application

The renderer has also been implemented as a plug-in for the IP studio system. This was used in the Dranouter field trial to allow real-time rendering to both headphones and loudspeakers within the IP-based broadcast production system.

6 Conclusions

This document describes the live editing application, the multi-panoramic video capture and processing engine APIs, and the audio rendering tools for linear broadcast playout. These tools enable playout from the ICoSOLE system to linear playout targets, which can either be traditional broadcast channels or web-based streams.

The live editing system provides the means for synchronised mixing of streams from different sources. The system is designed to work distributed in the network, which enables remote production.

The audio rendering tools enable rendering of an audio mix for the consumer both on the broadcast and on the client side, using a set of audio channels and related metadata according to the audio definition model (ADM) as input.

7 References

[1] ITU Radiocommunication Sector, “Recommendation BS.2076 - Audio Definition Model,” 2015.

[2] N. Zacharov, C. Pike, F. Melchior, and T. Worch, “Next generation audio system assessement using the multiple stimulus ideal profile method,” in *International Conference on Quality of Multimedia Experience*, 2016.

[3] ITU Radiocommunication Sector, “Recommendation BS.2088 - Long-form file format for the international exchange of audio programme materials with metadata,” 2015.

[4] V. Pulkki, “Virtual Sound Source Positioning Using Vector Base Amplitude Panning,” *J. Audio Eng. Soc.*, vol. 45, no. 6, pp. 456–466, 1997.

[5] F. Zotter and M. Frank, “All-round ambisonic panning and decoding,” *J. Audio Eng. Soc.*, vol. 60, no. 10, pp. 807–820, 2012.

[6] F. L. Wightman and D. J. Kistler, “Headphone simulation of free-field listening. I: Stimulus synthesis,” *J. Acoust. Soc. Am.*, vol. 85, pp. 858–867, 1989.

[7] J.-M. Jot, V. Larcher, and O. Warusfel, “Digital Signal Processing Issues in the Context of Binaural and Transaural Stereophony,” in *Proceedings of 98th Convention of the Audio Engineering Society*, 1995.

[8] P. Mackensen, U. Felderhof, G. Theile, U. Horbach, and R. S. Pellegrini, “Binaural room scanning—A new tool for acoustic and psychoacoustic research,” *J. Acoust. Soc. Am.*, vol. 105, no. 2, p. 1343, 1999.

[9] “AES69:2015 AES standard for file exchange - Spatial acoustic data file format,” 2015.

[10] J. L. Bentley, “Multidimensional binary search trees used for associative searching,” *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[11] ITU Radiocommunication Sector, “BS.1770-3 - Algorithms to measure audio programme loudness and true-peak audio level,” 2011.

8 Glossary

Partner Acronyms

BBC	British Broadcasting Corporation, UK
BIT	Bitmovin GmbH, AT
DTO	Technicolor, DE
iMinds	iMinds Vzw, BE
JRS	JOANNEUM RESEARCH Forschungsgesellschaft mbH, AT
TaW	Tools at Work Hard+Soft Vertriebsgmbh, AT
VRT	De Vlaamse Radio en Televisieomroeporganisatie NV, BE

Acknowledgement: The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 610370.

