Remote Collaborative Real-Time Multimedia Experience over the Future Internet

ROMEO

Grant Agreement Number: 287896

D3.3

Interim Report on 3D Audio - Video rendering algorithms

Document description	
Name of document	Interim Report on 3D Audio-Video rendering algorithms
Abstract	This document defines ROMEO audio-visual rendering algorithms developed for the different terminals
Document identifier	D3.3
Document class	Deliverable
Version	1.0
Author(s)	D.Doyen, S. Thiébaud, C. Thébault (TEC), C. Kim, I. Elfitri, H. Lim (US), M. Weitnauer, C. Hartmann, M. Meier (IRT), X. Shi (MULSYS), P. Apostolov (MMS)
QAT team	Nicolas Tizon (VITEC), Burak Demirtas (ARCELIK), Tasos Dagiuklas (UP)
Date of creation	03-Dec-2012
Date of last modification	31-Jan-2013
Status	Final
Destination	European Commission
WP number	WP3
Dissemination Level	Public
Deliverable Nature	Report

TABLE OF CONTENTS

TABLE OF CONTENTS **3**

LIST OF FIGURES **5**

1 INTRODUCTION **7**

1.1 Purpose of the Document **7**
1.2 Scope of the Work **7**
1.3 Objectives and Achievements **7**
1.4 Structure of the Document **7**

2 AUDIO RENDERING **8**

2.1 Introduction **8**
2.2 Mobile Terminal **8**
2.2.1 Binaural technology 8
2.2.2 Binaural Room Synthesis 9
2.2.3 BRS bottlenecks on mobile device 10
2.2.4 Status of current work 11
2.3 Portable Terminal **13**
2.3.1 Object-based Rendering 13
2.3.2 Status of current work 15
2.4 Fixed Terminal **15**
2.4.1 Spatial Audio Rendering Principles 15
2.4.1.1 Conventional Multi-channel Audio 15
2.4.1.2 Wave-Field-Synthesis (WFS) and Simplified WFS 16
2.4.2 Audio Rendering Implementation Framework 19
2.4.2.1 Conventional 5.1-channel rendering system 19
2.4.2.2 Simplified WFS system 19
2.4.3 Audio Rendering System Settings for the Fixed Terminal 21
2.4.3.1 5.1-channel rendering 21
2.4.3.2 Simplified WFS system setup 22

3 VIDEO RENDERING FIXED TERMINAL **24**

3.1 Introduction **24**
3.2 Fixed Terminal **24**
3.2.1 Experience acquired during the MUSCADE project (range of disparity) 24
3.2.2 Range of disparity defined for ROMEO 25
3.2.3 Depth Image Based Rendering Algorithm improvements 26
3.2.3.1 Overview 26
3.2.3.2 Disparity map interpolation 26
3.2.3.3 Occlusions identification 27
3.2.3.4 Occlusion border management 27
3.2.3.5 De-aliasing 29
3.2.3.6 Interpolated view rendering 30
3.2.4 Algorithm adaptation for parallelization 30
3.2.4.1 CUDA programming model 30
3.2.4.2 View interpolation CUDA kernel 31
3.2.5 Implementation into a GP GPU system 33
3.3 Portable Terminal **38**
3.3.1 Free view point 38
3.4 Mobile Terminal **39**
3.4.1.1 Introduction 39
3.4.1.2 Stereo rendering in Mobile Terminal Device 39
3.4.1.3 OMAP Display Subsystem Capabilities 40
3.4.1.4 Stereo rendering pipeline implementation 40

4 CONCLUSIONS AND FUTURE WORK **42**

5 REFERENCES **43**
APPENDIX A: GLOSSARY OF ABBREVIATIONS **45**

LIST OF FIGURES

Figure 1: Typical BRIR for the left ear 9

Figure 2: Block diagram of the overlap-discard partitioned convolution algorithm 10

Figure 3: Components of the sound field over time [10] 11

Figure 4: Sparkfun 9 degree of freedom IMU device with gyroscopes, accelerometer and magnetometer included 12

Figure 5: Basic building blocks to process and render object-based audio 14

Figure 6: First and second order reflection pattern generated by a sound source. Direct path (black), 1st order (blue) and 2nd order (yellow) 15

Figure 7: Loudspeaker layout for 5.1-channel system according to ITU-R BS. 775 standard, with a typical location for the LFE channel 16

Figure 8: Problem sketch 17

Figure 9: Sound field propagation generated by the algorithm 17

Figure 10: Sound field produced by a typical WFS system 18

Figure 11: Sound field produced by a simplified system 18

Figure 12: Typical multichannel audio rendering framework using external rendering hardware and ASIO protocol 19

Figure 13: Block diagram of main software algorithm 20

Figure 14: Block diagram of main hardware, ‡Alesis Digital Audio Tape protocol, †Multi-Channel Audio Digital Interface, §Hammerfall DSP 21

Figure 15: Loudspeakers available for 5.1-channel rendering in ROMEO 21

Figure 16: Multichannel audio rendering hardware in ROMEO 22

Figure 17: Arrangement of WFS setup at I-Lab 22

Figure 18: Loudspeaker panel 23

Figure 19: General configuration of the simplified WFS system 23

Figure 20: Overall ROMEO demonstration scenario 24

Figure 21: MUSCADE camera and relative view interpolation positioning 25

Figure 22: ROMEO camera and relative view interpolation positioning 26

Figure 23: Projection of the left disparity map 27

Figure 24: Interpolation scheme without border management 28

Figure 25: Interpolation scheme with border management 28

Figure 26: Interpolation results: without (top) and with (bottom) border management 29

Figure 27: Disparity map and corresponding interpolated video without de-aliasing 29

Figure 28: Disparity map and corresponding interpolated video with de-aliasing 29

Figure 29: Interpolation results: without (top) and with (bottom) de-aliasing 30

Figure 30: Disparity projection per thread 32

Figure 31: Projected disparity by all the threads 32

Figure 32: Projected disparity after filtering 33

Figure 33: Functional description 35

Figure 34: Concurrent execution 37

Figure 35: Asynchronous launches and CUDA streams 38

Figure 36: View interpolation for portable terminal 39

Figure 37: Stereo rendering pipeline example 40

Figure 38: Stereo rendering pipeline operation 41

1 INTRODUCTION

1.1 Purpose of the Document

The purpose of this document is to present a status of the work done in the field of audio-visual rendering within the ROMEO project. This document D3.3: “Interim Report on 3D Audio – Video rendering algorithms” provides technical descriptions of the work done in the field of the task 3.4 of the project.

As an interim report, it presents the current status of algorithm developments applied to the different terminals (fixed, portable and mobile). This document will be updated at the end of the second year of the project (D3.6 due in M24).

1.2 Scope of the Work

This deliverable reports the initial work carried out towards the development audio/video algorithms to improve the user experience for fixed and mobile terminal .The different terminals are receiving audio and video content through the hybrid ROMEO transmission chain. This is the purpose of the WP3 to work on the exploitation of these contents using respective capabilities of terminals.

1.3 Objectives and Achievements

The task 3.4 is dedicated to review and implement appropriate audio and video rendering algorithms taking into account the complexity and buffering for real-time processing and displays/loudspeaker systems. Spatial audio rendering techniques will be based on WFS combined with Ambisonics for fixed users and binaural spatial audio rendering will be considered for the mobile users. For the 3D, multi-view video content rendering is addressed by interpolating the in-between viewpoints.

1.4 Structure of the Document

This deliverable is structured as follows: Section 2 presents the audio rendering processing and section 3 the video rendering part for each terminal. Section 4 concludes this deliverable with some insights to future work.

2 AUDIO RENDERING

2.1 Introduction

This section introduces the principles behind the various audio rendering techniques used for the project. Moreover it provides an overview about the audio rendering work for the different terminals used in ROMEO. The algorithms and principles (state-of-the-art and beyond) are described as well as the current status of the implementation.

2.2 Mobile Terminal

The so-called mobile terminal of the ROMEO project targets mobile platforms such as smartphones and tablet devices. In the past years, these kind of devices evolved to constant companions of modern individuals. Due to the fact that the playback of audio with these devices is mainly via headphones, the audio rendering approach of ROMEO for this kind of terminal will also use the benefits of headphone sound reproduction.

2.2.1 Binaural technology

Binaural audio is a spatial audio technology, which is based on the principles of natural human hearing. Due to having two ears, humans can localise sound direction. Sound waves coming from a source reach the two ears at different times because they travel different distances. Shadowing effects by the human head also cause the sound pressure at each ear to have different levels. This means that human hearing system can simply detect the position of a sound source by processing mainly the inter-aural time differences (ITDs) and inter-aural level differences (ILDs) [1]. Shadowing by human head, reflection and diffraction by surrounding environments can also cause differences in the frequency spectrum of the signals received by the left and the right ear, which are used by the human auditory system to further improve the localisation of sound sources.

However, there are some positions such as in front or in the back of human head, which would produce identical ITDs and ILDs. If a sound comes from these locations, humans will get confused and cannot precisely determine the source position. This phenomenon is known as cone of confusion [3] and can be solved by head movements [4]. Using the changes of the ILDs and ITDs, which are referred to as dynamic binaural cues, the human auditory system is able to assign the correct position to the sound source. Another situation that makes human brain fail to detect proper positions of sound sources is when there are two or more sound sources, located in different locations, emitting the same sound with small time delays. These sounds will be perceived as a single sound source localised at the position of the sound source emitting the sound first. Such a situation is called the precedence effect [5].

The human head‘s size, the unusual shape of pinna, as well as the reflections off the shoulder and body will also affect the ability to localise a sound source. All of these cues can be represented as a head-related transfer function (HRTF) [2]. This function is a frequency response of the human outer ear. As HRTFs are also depending on the position of a sound source, they are typically measured in an anechoic chamber for every needed source position and stored in a database. If the transfer functions also include the directivity and characteristics of a sound source (e.g. a real loudspeaker), the acoustic properties of a room and the HRTF of a human head, these are commonly called Binaural Room Transfer Functions (BRTF). Converted into in the time domain, impulse responses are generated. These are called Binaural Room Impulse Responses (BRIR) [2]. Figure 1 shows a typical BRIR of a loudspeaker within an acoustically optimized monitoring room.

There are basically two ways to acquire BRIRs. The first option is to use real measurements of real sound sources at various positions and for all possible head rotations. This so-called data-based approach generates high quality BRIRs which leads to a very good spatial audio quality. The downside of this approach is that only source positions can be used during playback that have actually been measured. Furthermore, the measurement process is

relatively time intensive, but only has to be done once. This approach is therefore superior for static playback situations, for example to simulate a 5.1 playback system on a mobile device.

The second option is the so-called object-based approach. For this, the transfer paths of a given sound source in a virtual room are calculated depending on the position and orientation of the listener. Based on these transfer paths, BRIRs can be synthesized for anechoic conditions using a HRTF database and directivity patterns of the sound sources. Furthermore, virtual listening rooms can be included in the simulation process to provide a realistic room impression. This approach is very flexible as sound sources can be moved freely and dynamically, but comes with a highly increased computational complexity compared to the data-based approach and is therefore not able to produce the same *high* quality BRIRs with reasonable resources.

Thus it is necessary to choose the right approach based on the requirements and system constraints of the targeted application. For the mobile system, a data-based approach based on *Binaural Room Synthesis* was chosen. For object-based audio playback on the portable device, obviously only the object-based approach is possible.

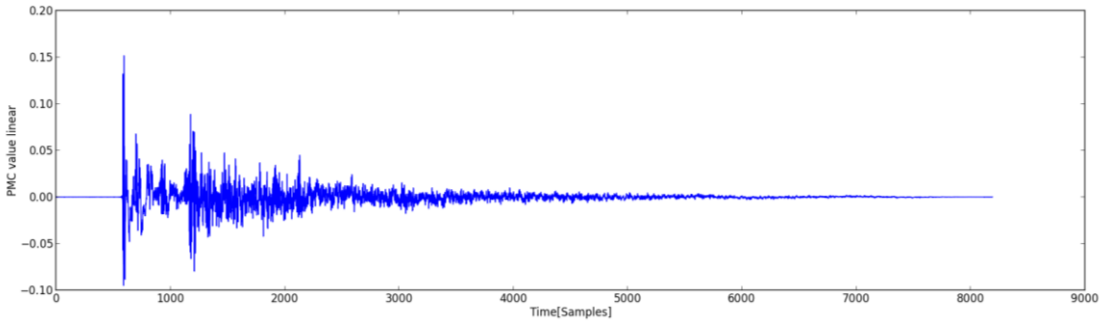

Figure 1: Typical BRIR for the left ear

2.2.2 Binaural Room Synthesis

The convolution of channel based audio signals with BRIRs generates a binaural signal. If this binaural signal is played back through headphones, it gives the listener a spatial impression, comparable to the one in this room at the recording position. This method is called binaural room synthesis (BRS) [6] and exists as some rare implementations for professional production, including a high-quality implementation of IRT. By employing a head-tracking solution, the binaural signal can be adapted to the rotation of the user's head. If this adaptation is conducted in real-time, the perception of the user is massively enhanced by dynamic cues. The localization of sound sources is much easier and front-back confusions are mostly avoided.

Since the whole adaption process (including head-tracking, filter selection and convolution) has to be conducted within less than 85 ms [7] and even the duration of a typical BRIR is longer than this value (e.g. the duration of the BRIR in Figure 1 is 169 ms), the signal processing has to be realized with the so-called partitioned convolution, which is also known as fast convolution. A simple block diagram of the fast convolution using the popular overlap-discard method is depicted in Figure 2.The basic idea is to split both the continuous input signal as well as the BRIR into smaller partitions. Then each input signal block is convolved with each filter partition. The results of these operations can then be combined into the desired output signal. The benefit of using this method is that the latency is reduced to the length of the input block size. Furthermore, as the convolution is carried out in the frequency domain, the computational complexity is reduced significantly compared to time domain convolution.

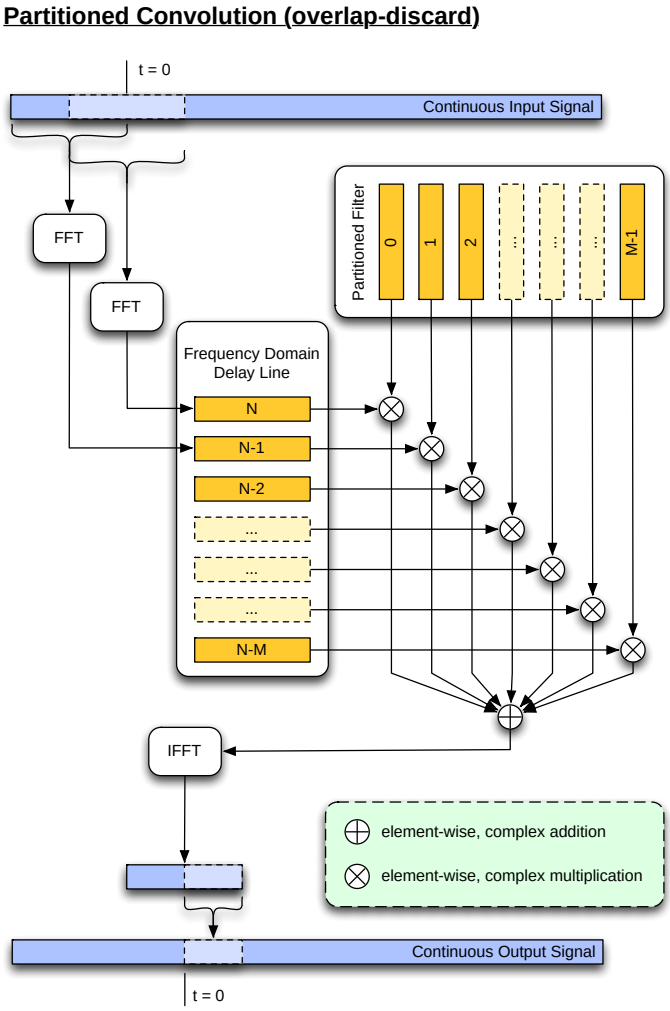

Figure 2: Block diagram of the overlap-discard partitioned convolution algorithm

For mobile applications, headphones are an interesting option to reproduce multichannel audio signals by transcoding multichannel audio into binaural audio. Using a convolution with BRIRs for the purpose of generating a binaural signal on mobile devices - especially not with a head-tracking solution - was not commercially available at the beginning of ROMEO.

2.2.3 BRS bottlenecks on mobile device

There are several major difficulties that occur, if a BRS system shall be implemented on a mobile platform.

The most obvious difficulty is the possibility to track the movements of the user with a head-tracking device suitable for mobile and outdoor applications. Therefore a small and lightweight tracking system, which provides resilient measured data even under highly diverse environmental conditions, has to be implemented. The most important requirements for the usage of head tracking with mobile MTB devices are:

- A tracking technology, which does not need a stationary reference point, a moving target could be measured relative to
- An update rate higher than 20Hz for smooth repositioning of audio objects
- A tracking precision < 1°
- An operation time of several hours (battery drain)

- Compact, lightweight and easy to use design
- Resilient against environmental influences like mechanic impacts as well as electric and magnetic interfering fields
- Reproducible at a reasonable price

Other difficulties are the limited computational power and RAM (Random-access Memory) capacity of state-of-the-art mobile devices. The computational demand of a BRS implementation for a 5.1 virtualization is still pretty high. Hence, the signal processing itself has to be enhanced and the BRIR have to be simplified. An enhancement of the signal processing can be reached by separating the impulse response into two parts. The first part contains only the direct sound of the impulse response and the second part only the reverberation, as it is schematically shown in Figure 3 . To save storage space, RAM and computational power, the same reverberation part can be used for all impulse responses. A simplification of the used BRIR can be reached by shortening the impulse responses to a minimum number of samples. Both measures have to be applied intensively to realize the signal processing in real-time on a mobile device. Hence, investigations are a necessary step for a successful implementation.

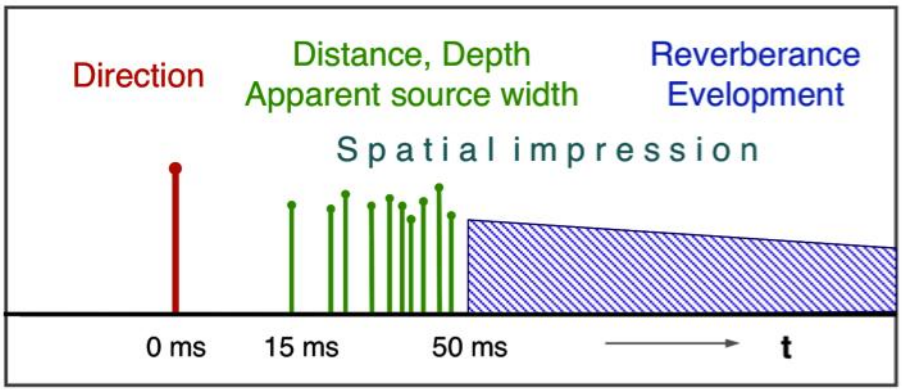

Figure 3: Components of the sound field over time [10]

2.2.4 Status of current work

ARM implementation

As a first step, a state-of-the-art Tablet device (Acer ICONIA A501 Tablet) with Android version 2.6 has been considered as development and testing platform. The first test implementations showed that the Android OS with version 3.0.1 (Kernel version 2.6.36.3) has an unacceptable delay for audio buffers. This was true for all available audio APIs (Application Programming Interface) of the Android SDK[1] (Software Development Kit) as well as for the Android NDK[2] (Native Development Kit). The NDK offers the possibility to use C programming code within a special Java wrapper to (theoretically) increase the performance of certain computations. The detected delay for each audio buffer was around 250ms, which is a serious conflict with the allowed maximum delay of 85ms. Due to this performance problem and the fact that MMS committed to use a Texas Instruments OMAP 5430 platform it was decided not to use Android as operating system for the mobile BRS implementation.

[1] http://developer.android.com/sdk/index.html

[2] http://developer.android.com/tools/sdk/ndk/index.html

The next step included several informal investigations and evaluations of existing platforms and operating systems based on ARM architecture. The so-called PandaBoard[3] offers an OMAP4432, which is the predecessor of the targeted OMAP 5430 platform. Since the Android OS caused an unacceptable delay, it was decided to use a common Linux distribution for ARM processors. The available structure offers the potential to apply the necessary signal processing with an adequate computation delay.

To conduct the described investigations regarding an optimization of the BRIR, a software tool has been implemented which allows creating and editing of BRIR databases for IRT's BRS implementation.

Mobile head-tracking device

Investigations already carried out showed, that on the basis of the aforementioned requirements only a few principles of operation seem to be qualified for the realisation of a mobile head-tracking device. Out of the six main tracking principles just inertial sensing and direct field sensing have the potential to be used in this field of operation.

The principle of inertial sensing is based on the attempt to conserve either a given axis of rotation or the position of a weight. It can be subdivided into two basic technologies, which are preferably used in combination: The mechanical gyroscope, a system based on the principle of conservation of the angular momentum and the accelerometer, which measures the linear acceleration of an object to which it is attached. Usually not only one sensor of each type is used for detecting the position, but at least one or more sensors per dimension are coupled in order to form a so-called inertial measurement unit (IMU). Hence, an IMU commonly provides three degrees of freedom (DOF) or more and can sense three-dimensional position values. Gyroscopes and accelerometer are both lightweight and can be produced very cost-effective. Most convenient however in the context of mobile tracking is the fact, that inertial sensors don't need an external reference to work. The downside of this characteristic is, that they are only able to provide relative measurements. As a consequence, inertial systems suffer from accumulated numerical errors, caused by unavoidable measuring inaccuracies, which lead to a constant drift. For this reason it is advisable to perform a periodic re-calibration of these systems, which will insure, more accuracy over time.

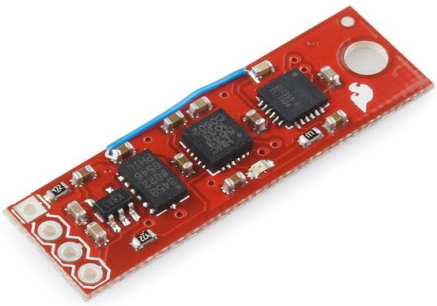

Figure 4: Sparkfun 9 degree of freedom IMU device with gyroscopes, accelerometer and magnetometer included

This re-calibration can be done with a magnetometer, belonging to the category of direct field sensing devices. With three magnetometers the orientation of an object with respect to the

[3] http://pandaboard.org/

magnetic field of the earth can be measured. One problem with this technology is that the Earth's electromagnetic field is inhomogeneous and yields angular errors in the orientation measurements. Therefore it's most beneficial to use this technology as an enhancement for the precision of inertial sensors within an IMU.

Another possibility is to use a head-tracking device entirely based on magnetic field sensing, where a magnetic coupling between an emitting coil and a receiving coil is used to measure the relative distance and orientation between these parts. A problem with this type of trackers is the distortion of the magnetic field by ferro-magnetic objects and other sources of electromagnetic fields.

As part of the investigation within ROMEO, low-cost versions of an IMU- and a magnetic field-based head-tracker are being developed and implemented, probably based on the board shown in Figure 4. They will be compared with commercial trackers with the same operating principle in terms of their performance characteristics. In addition a comparison with a state-of-the-art time of flight (TOF) based head tracker will be performed. These devices rely on the measure of distance between a moving target in relation to a fixed reference and usually use techniques like infrared or ultrasonic emission. TOF-systems are the head-tracking standard for fixed BRS installation and can be considered to be the most accurate tracking systems.

Another aspect of the investigation is concerning the effective separation of torso and head-movements. For this purpose there exist multiple technical strategies, which involve the utilisation of additional sensors, as well as the application of numeric reconstruction algorithms, such as Kalman filtering. This issue is very important for the mobile binaural playback; hence the change of the walking direction does not affect the orientation of the audio scene, what would be a highly unwanted effect.

2.3 Portable Terminal

The portable terminal of the ROMEO project targets devices such as laptops. Since audio with these devices is either played back via integrated / external loudspeakers or via headphones, the audio rendering approach of ROMEO for this kind of terminal will take into account both playback solutions. Since the usage of binaural signals with headphones offers a much better spatial impression, the mainly targeted playback solution will also use headphones.

2.3.1 Object-based Rendering

Considering the representation of a (possibly 3D) audio scenery for spatial audio rendering, a common approach in the research field is the use of individual audio objects instead of channel-based audio formats [8][9]. These audio objects carry additional metadata, such as position, gain or directivity. Furthermore, such an audio scene may be extended with a room model to enhance the spatial audio impression and capture the creative intent. Such a description of a spatial audio scene is in principle independent of the playback system used and very flexible. It is therefore possible to provide renderers for output formats or playback technologies, e.g. binaural audio or stereophony.

In Figure 5 the basic building blocks generally needed to process object-based audio are depicted. Using the object-based audio scene description the rendering parameters specific to the targeted playback system are generated for each sound source. These parameters include gain factors, delays or synthesized impulse responses. If necessary, the scene description can be pre-processed to adjust and optimize certain parameters within the scene to the capabilities of the targeted playback system. This process is repeated every time the audio scene changes, e.g. by a moving sound source and commits the generated parameters to the audio processing engine. This audio processing engine is computing the output signals in real-time based on the input signals and its parameter set.

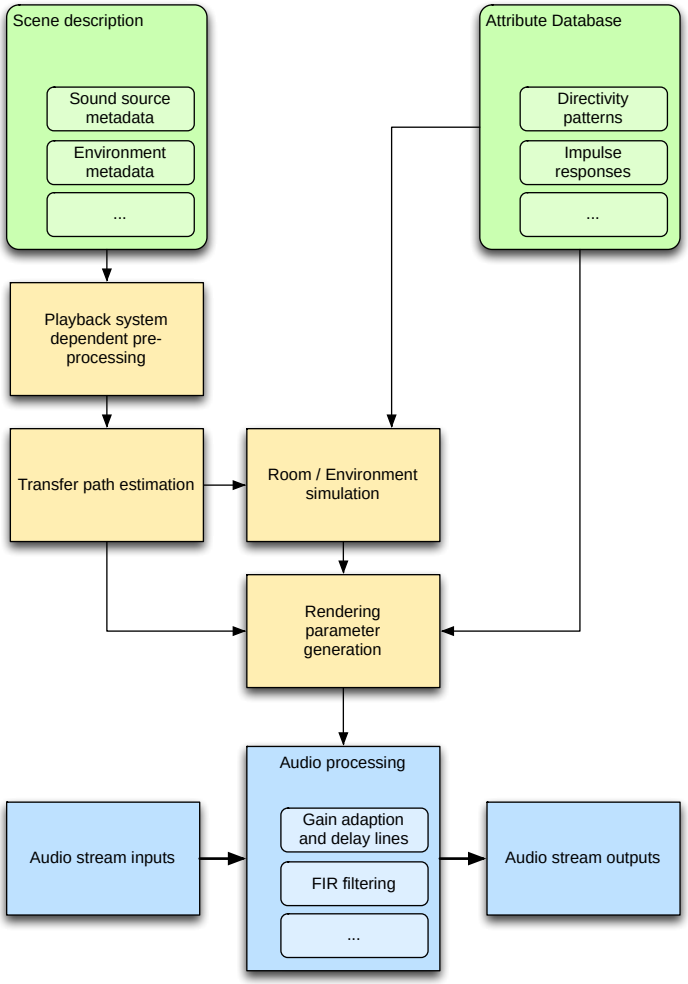

Figure 5: Basic building blocks to process and render object-based audio

For binaural rendering, the rendering parameters mainly consist of artificial BRIRs, which are calculated based on the acoustic transfer paths of each audio source to the listener. The BRIRs can be synthesized by correctly modeling time delays, attenuation and a HRTF database. For stereophonic playback, basically two options exist. The first one is to use an identical process as for binaural rendering, but replacing the HRTF database with impulse responses of multi-channel main microphones, thus simulating a conventional microphone recording for multi-channel setups. The second option is to use techniques like Vector Based Amplitude Panning, thus mapping the audio source signals to the discrete output channels with different attenuation values and time delays [11].

To enhance the spatial impression of the audio playback, additional room simulation may be added to parameter generation process (Figure 6). Based on a geometric description of the room, two methods can be used to simulate virtual environments. The first is the so-called mirror-image method. The basic idea is that every reflection generated by a sound source and a surface can be substituted by introducing another virtual sound source. The second method is to use ray-tracing algorithms to calculate the reflection pattern generated by a sound source. As each of these two methods has its advantages and drawbacks, they are often used in combination as a hybrid approach [12].

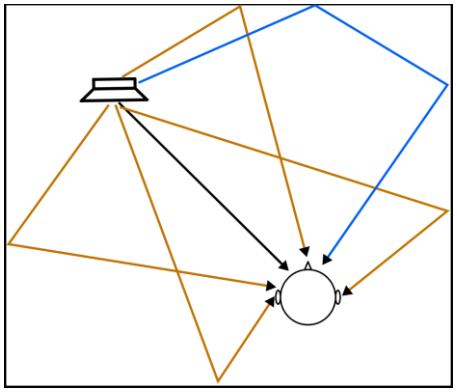

Figure 6: First and second order reflection pattern generated by a sound source. Direct path (black), 1st order (blue) and 2nd order (yellow)

2.3.2 Status of current work

The core functionalities and libraries for object-based rendering have been implemented in C++. This includes geometric transfer path calculation as well as generation of filters and impulse responses. Furthermore, all currently needed audio processing sub-blocks have been implemented. To test and evaluate various features of the scene description language developed within this project, a flexible rendering platform has been started, which can be used to incrementally implement and test new features as required by the scene description language and evaluation results. The current rendering prototype is capable of rendering binaural signals for static scene in anechoic conditions.

2.4 Fixed Terminal

For the fixed terminal, conventional multichannel audio rendering, particularly in 5.1 channels, is introduced as the main configuration. Additionally, Wave Field Synthesis (WFS) is also under investigation towards more practical use with reduced number of transducers.

2.4.1 Spatial Audio Rendering Principles

This section describes the principles and backgrounds for the conventional multichannel audio rendering system and the WFS system.

2.4.1.1 Conventional Multi-channel Audio

Through the recent years, spatial audio reproduction systems have progressed in various configurations using multiple loudspeakers, in order to deliver to the users the perception of auditory scenes realistic in terms of sound source distribution, or enriched in terms of spaciousness or envelopment. The mechanism and cues for the spatial audio perception have been identified through a number of research works. It is widely known that the difference in the arrival time of sound between the two ears (Interaural Time Difference) and the difference in the amplitude of sound between the ears (Interaural Level or Intensity Difference) are the main cues for lateral sound source localisation. Additionally, the spectral difference of sound caused by the head, the structure of outer ear, and the torso acting as filtering elements are known as the main vertical localisation cue. The frequency content, loudness, and the amount of reverberation are known to affect the perception of the distance or depth. On the other hand, the similarity of the signals detected at the two ears is known to be related to the perception of apparent source width and envelopment of sound. The conventional multichannel surround audio reproduction systems have been developed generally to render these cues, mainly in the horizontal plane. The number of loudspeakers has been increasing through research and investigations since the days of 2-channel stereo, to overcome the limitations in the reproducible auditory scene. Amongst the various configurations, the 5.1-

channel system is the most widely accepted and standardised one, which will thus be introduced for the ROMEO project.

The loudspeaker layout, seen from above, is specified in ITU-R BS.775 and shown in Figure 7, with a typical location for the Low-Frequency Effects (LFE) channel denoted by the '.1' component. The front loudspeakers maintain compatibility with two-channel stereo systems and reinforce front-oriented sound reproduction. The two "surround" loudspeakers contribute in creating sound images behind the listener and in providing lateral energy for good envelopment. The LFE channel is responsible for the low frequency content (below a maximum of 120 Hz) that cannot be handled by the other 5 loudspeakers. The EBU comments that this channel content should be considered optional and the overall reproduction should sound satisfactory even without this channel. The dotted rectangles in the frontal area describe the recommended screen placements.

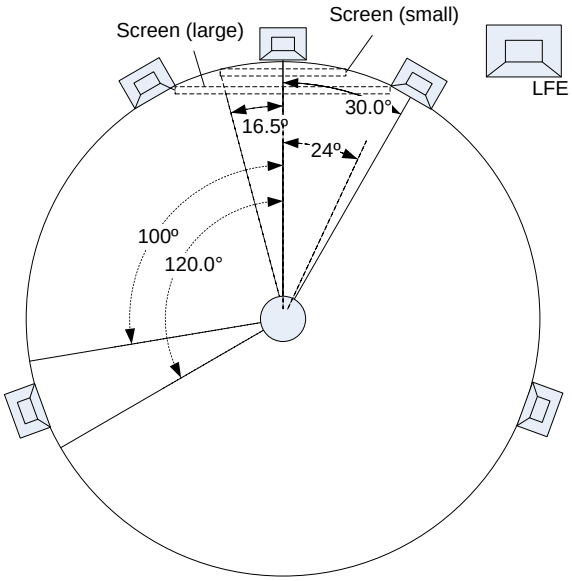

Figure 7: Loudspeaker layout for 5.1-channel system according to ITU-R BS. 775 standard, with a typical location for the LFE channel

2.4.1.2 Wave-Field-Synthesis (WFS) and Simplified WFS

WFS is a practical application of the Huygens' principle [13]. The main feature of this methodology is its ability to automatically reproduce spatial sound equivalent to the original one generated at the source position within an entire domain by using secondary sound sources [14]. The secondary sources establish an active boundary, rendering a virtual lifelike soundscape in the given region. The boundary (control sources) can are composed with different combinations of acoustic monopole and dipole. However, the bulky dipole source can be omitted for simplification of the system as we are not interested in the sound field outside the domain.[15] Theoretically, Rayleigh I integral Eqn. (1) allows us to implement the WFS system with a series of acoustic monopole sources.[16]

$$P(\vec{r},\omega) = \rho_0 c \frac{jk}{2\pi} \iint_S \left(V_n(\vec{r_S},\omega) \frac{e^{-jk|\vec{r}-\vec{r_S}|}}{|\vec{r}-\vec{r_S}|} \right) dS \quad , \tag{1}$$

where ρ_0 denotes the air density, V_n the normal component of the particle velocity, k is a wave number, ω is the angular frequency of a wave, and c is the speed of sound. $\vec{r}$ denotes a position vector of a listening position inside the domain surrounded by an arbitrary surface S. $\vec{r}_S$ is a position vector on the surface S. φ denotes the relative angle between the vector from a position on S to the listening position L and the normal vector $\vec{n}$ in Figure 8.

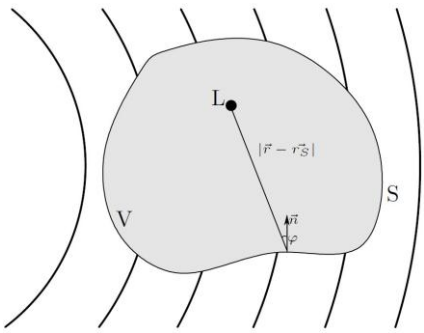

Figure 8: Problem sketch

In the equation a continuous distribution of secondary sources has been supposed.[17] However, no acoustic line source controllable precisely in detail for WFS has been discovered yet. In practical realization, the secondary source can be implemented only with arrays of number of discrete loudspeakers. The practical limitation requires the equations described in discrete form. Some simulation models in the discrete form have been developed at the I-Lab for effective evaluation of the algorithm in various given environments. A typical example of them is illustrated in Figure 9where a non-focused virtual acoustic source is supposed to be at x=0 and y=-2, i.e. outside a listening room, and a listener is situated in the middle of the room.

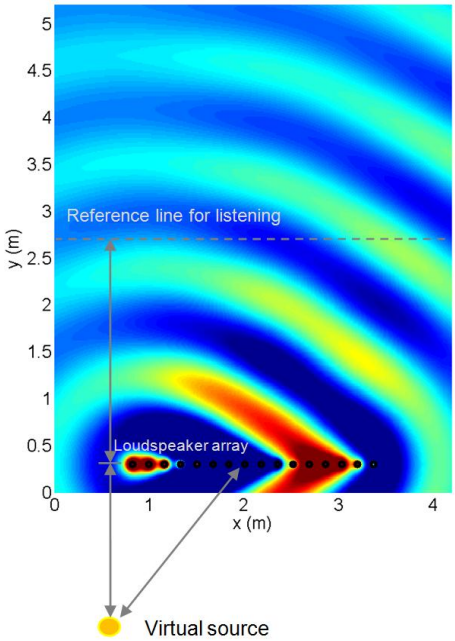

Figure 9: Sound field propagation generated by the algorithm

According to further requirements, the function of the WFS system can be enhanced to enable to reproduce sound fields generated by moving sources and focused sources, i.e. sources inside the listening area in Figure 10.

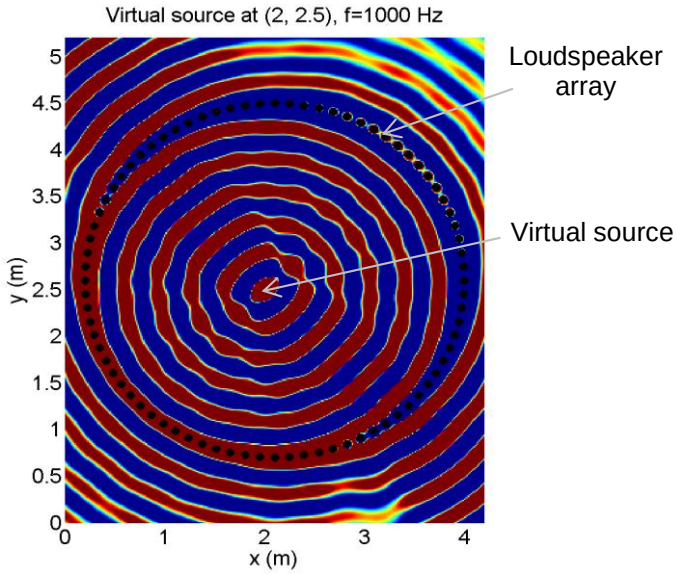

Figure 10: Sound field produced by a typical WFS system

Existing conventional WFS systems generally provide the advantages of sound performance on a large scale. The project's focus is on the fundamental development of the simplification of WFS and its application to practical 3D audio. The scaled-down WFS has an attractive advantage over the established conventional WFS where a suitable space is unavailable, and can provide a major contribution to home or private users. The solution is developed for WFS with minimum (or less) loudspeakers and defines limitations of the system, e.g. the size of a 'sweet area' (see Figure 11). For the purpose more effective distribution of loudspeakers and shape of the array can be developed through the project

.

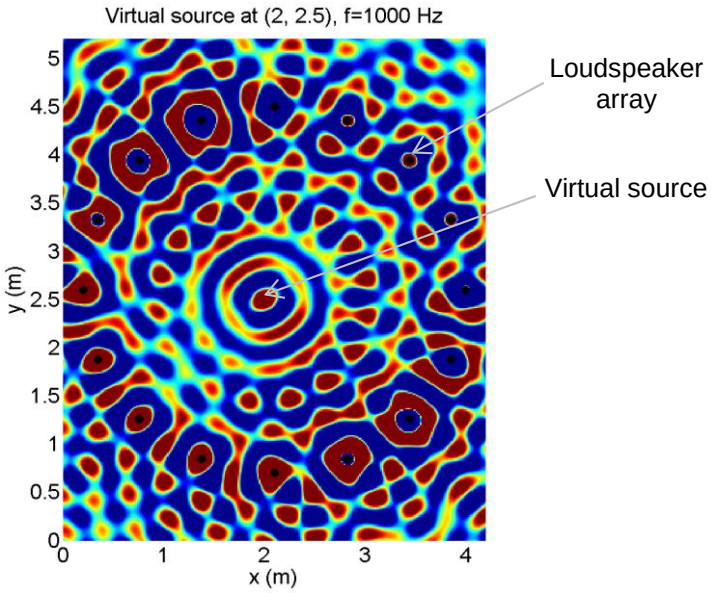

Figure 11: Sound field produced by a simplified system

Recall that the active boundary should be necessarily discretized in realisation. The reproduced sound is supposed to be identical to the original one only within a certain frequency range that is largely limited by the distance between the two neighbouring

loudspeakers. The highest limitation to the frequency range corresponds to the spatial Nyquist frequency Eqn. (2).

$$f_{nyq} = \frac{c}{2\Delta x} \ , \tag{2}$$

where c is the speed of sound and Δx is the distance between the two loudspeakers. Spatial aliasing can happen when the signal's frequencies exceed the Nyquist frequency.[18] Spatial aliasing might result in a spatial distortion of the sound field between the listener and source in the listening area. In order to reduce the number of the drivers for the simplification Δx in Eqn. (2) can be defined by optimising the combination of f_{nyq} and the relative position of the listener to the virtual source.

2.4.2 Audio Rendering Implementation Framework

2.4.2.1 Conventional 5.1-channel rendering system

Most modern consumer PCs and audio devices support 5.1-channel surround audio rendering. In PC applications, the processing units and D-A converter, in the form of an internal chipset or of an external hardware device, are controlled by software applications by means of various audio data transfer protocols through interfaces that allow for the required bandwidth for the multichannel audio. Particularly for professional audio production under Windows environment, the ASIO (Audio Stream Input/Output) protocol is the most widely known multichannel audio driver format. Developed by Steinberg, this is designed to bypass the generic communication layer of the operating system and to enable direct connection between the audio device and the software application. It therefore allows for audio data transfer for larger number of channels with low buffering latency. In order to handle a large number of reproduction channels, professional audio rendering devices usually operate as external hardware with the input/output ports and connectors. They are connected to the controlling PC via interfaces such as USB, Firewire, and PCI Express, with the former two being most widely utilised. Figure 12 conceptually describes the framework, the data flow and the operations, which is applied in the audio rendering system with the ROMEO project.

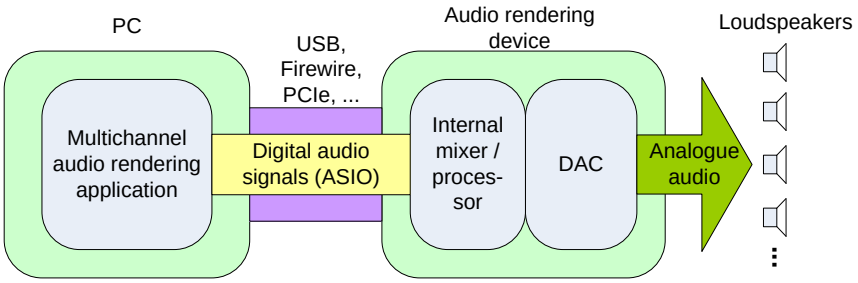

Figure 12: Typical multichannel audio rendering framework using external rendering hardware and ASIO protocol

2.4.2.2 Simplified WFS system

The WFS software generates signals for each loudspeaker and hence the virtual sources in the scene. The software is developed with MS Visual C++ in MS Windows 7 64-bit environment. In addition, several models have been developed for the simulation of WFS algorithm. Regarding the verification of the main driving function, it is helpful to start by analysing a simple case in the first instance. The advantage of this is that not only the system can be modelled easier but the sensitivity of the system and spatial reverberant effect of the room can also be determined. This is important for the WFS implementation, because the big shortfalls of WFS in practical situations are that it requires both an anechoic environment for accurate sound field reproduction [19][20] and continuous bulky loudspeaker arrays surrounding a domain. Therefore, it is important to quantify the reverberant effects of the given

room and to reduce the number and size of drivers in quantity and volume. The earlier considerations in the literature on this subject of WFS appear in [21, 22]. The effect of reverberation is very limited in the current project as long as the interior walls and ceiling of the room, except a window glass and door, are entirely covered by sound absorptive panels and the floor also by thick absorptive fibreboard.

The main process of DSP is illustrated in Figure 13. The data loaded from the input consists of the audio signal, source information, sampling frequency etc. The source information includes the spatial position vector of each virtual sound source. In addition, the distance from the loudspeaker array to the listening position, the number of drivers and a gap between them are given in advance.

The first step into the algorithm is calculation of the distance between the driver and source. The driving function calculates the time in each sample that the sound front takes from the virtual source arriving to each driver, based on the input data. Then the convolution of the output driving signal is performed with the impulse response of each driver. The signal after the convolution is described in a frequency domain through the FFT process so that the directivity can be included. The directivity of each driver according to the frequency and angle of emission is measured before the WFS algorithm starts. Generally, it can be measured in the calibration process in advance. The frequency contents are transformed back into time domain via the inverse FFT (IFFT) process. The output vectors can be saved as audio files.

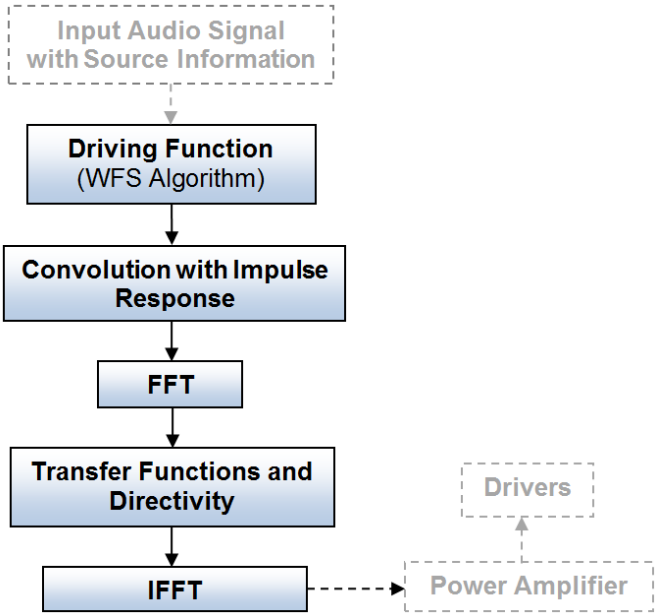

Figure 13: Block diagram of main software algorithm

In practical realization, the secondary source can be implemented with arrays of number of discrete drivers. The hardware modules for WFS are illustrated in Figure 14. It is supposed that the measurement system obtains beforehand the audio signals and additional objective information for the original sources.

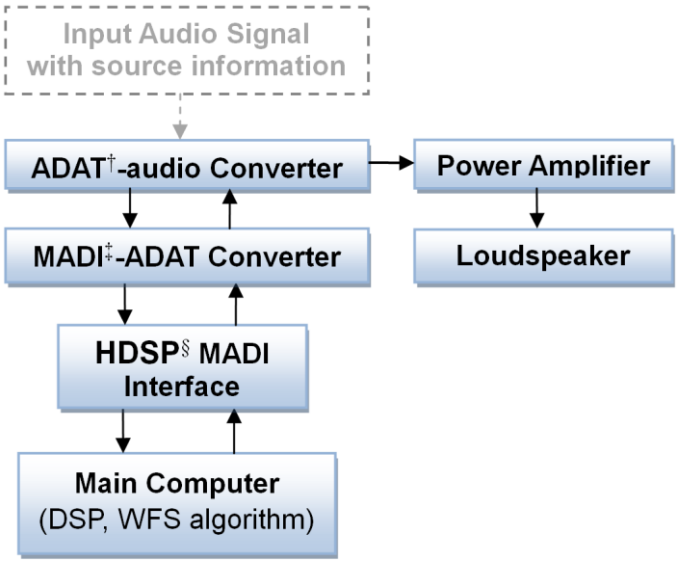

Figure 14: Block diagram of main hardware, ‡Alesis Digital Audio Tape protocol, †Multi-Channel Audio Digital Interface, §Hammerfall DSP

2.4.3 Audio Rendering System Settings for the Fixed Terminal

In this section, the system and settings available for the spatial audio rendering with the fixed terminal are introduced.

2.4.3.1 5.1-channel rendering

For the 5.1-channel spatial audio rendering, I-Lab, University of Surrey has 16 units of Genelec 8030A loudspeakers along with 2 Genelec 7060B loudspeakers for the LFE channel, as can be seen in Figure 15. These can be set up at various places for audio-specific applications, or for cases where visual displays of various sizes need to be accompanied by the multichannel audio.

Figure 15: Loudspeakers available for 5.1-channel rendering in ROMEO

These are driven by a dedicated external audio rendering hardware, MOTU 828 mk2 (Figure 16). This is connected to a PC with Windows 7 OS running the software applications, via Firewire.

Figure 16: Multichannel audio rendering hardware in ROMEO

As previously mentioned, the 5.1-channel rendering is supported by most professional and consumer level hardware, and thus by most available software applications including media players and digital audio workstations (DAWs). In the ROMEO project, the audio decoding module as a software application has the 5.1-channel rendering feature integrated. It communicates with MOTU 828 mk2 using the ASIO driver to feed the decoded audio signals to the corresponding channels

2.4.3.2 Simplified WFS system setup

All the loudspeaker panels are controlled individually by a powerful PC and hardware control modules situated outside the studio as shown in Figure 17. The control PC is equipped with HDSP cards which are compatible with fast PCI express buses. Digital signals are then fed into D/A converters. Each of them converts 64 channels of the digital inputs into analogue outputs. Loudspeaker cables connecting the control modules with drivers are fed through the wall to each loudspeaker panel. The control software includes a series of audio APIs as libraries to implement complicated audio DSP techniques and WFS algorithms.

The WFS system takes place in a studio lab (4.2 x 5.2 m^2) at the University of Surrey. A plan view of the new WFS system in the lab is illustrated in Figure 17.

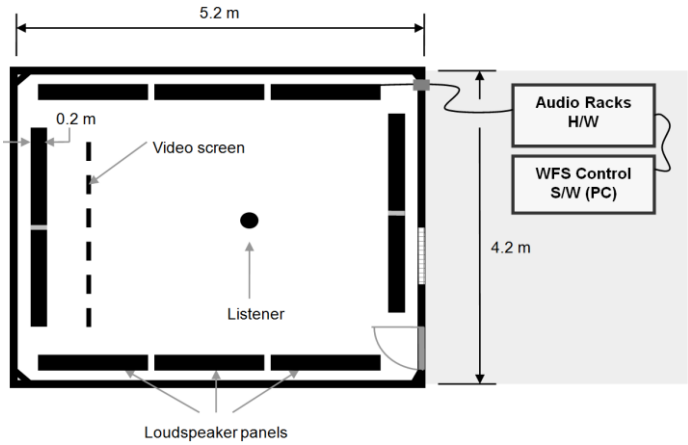

Figure 17: Arrangement of WFS setup at I-Lab

Currently ten loudspeaker panels (see Figure 18) as secondary sources stand close to the walls of the studio inside. In order to prevent an unwanted reflected sound the drivers can be mounted into the walls of the reproduction room. In addition, layered loudspeaker arrays and absorptive walls are also considered.

Figure 18: Loudspeaker panel

In many practical cases, e.g. at ordinary private houses, bulky loudspeaker arrays for typical WFS systems are not available to be mounted permanently in a small room. From the practical point of view, hardware of conventional WFS systems, especially loudspeaker arrays, should be scaled down in size and also in number.

Regarding positions of the virtual sources, a further simplification of the active boundary is possible. If the sources are created in a particular side only, the array can be shorten. In this case the array serves as an acoustic window through which the sound entering from the side is supposed. This simplified version of a wave field synthesis can be suitable for modular construction of the WFS system as the active boundary covering a complete 360 degree angle can be readily extended from it.

Unlike many other existing WFS methods, our approach aims to implement an simplified solution by using practically reduced number of drivers in a limited listening angle corresponding to the view angle to a video screen as shown in Figure 19. By requiring only limited hardware resources, the unique characteristics of the proposed approach can provide a practical and cost-effective WFS system for general home users. In addition, the system may be equipped with simple and easy user interfaces as well.

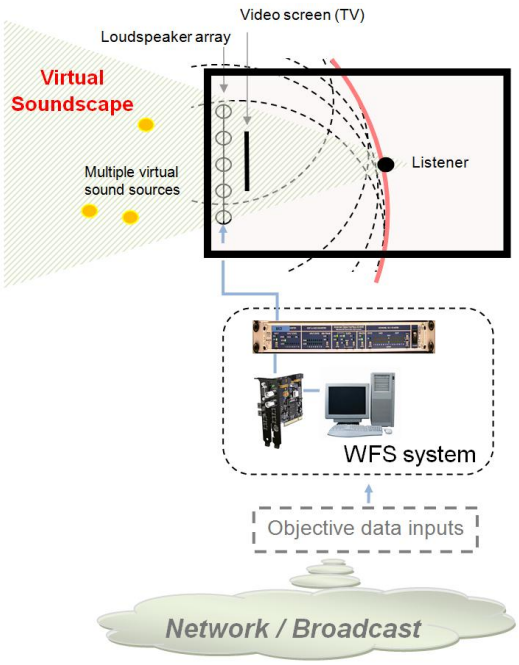

Figure 19: General configuration of the simplified WFS system

3 VIDEO RENDERING FIXED TERMINAL

3.1 Introduction

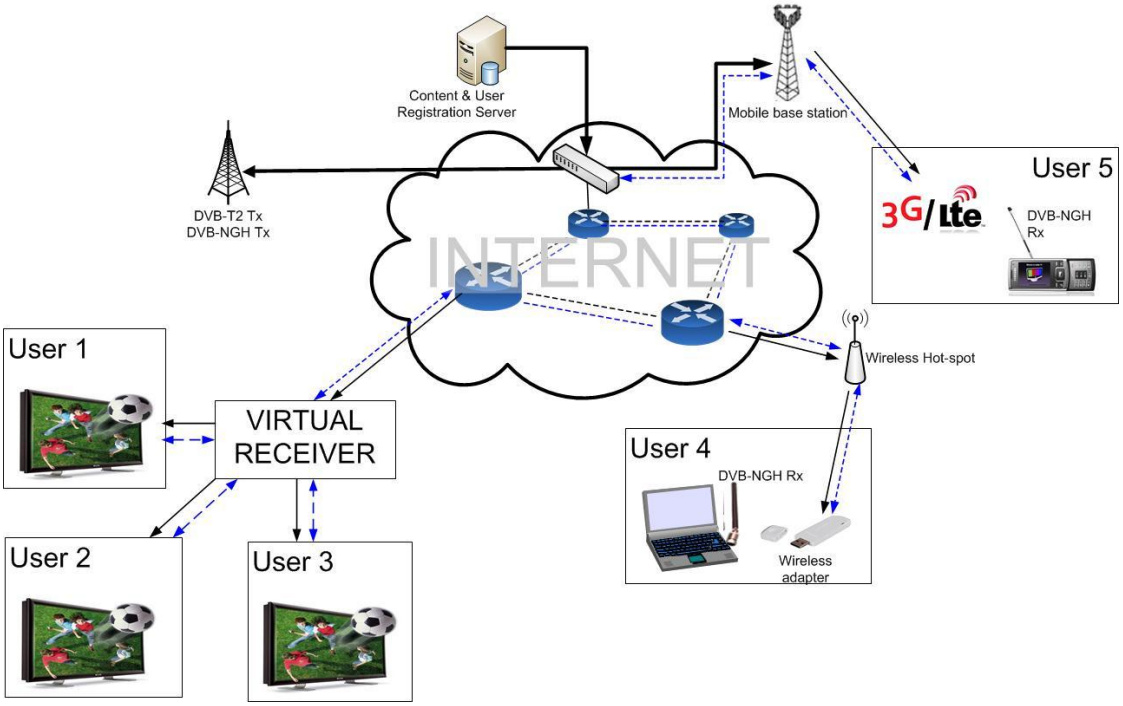

Figure 20: Overall ROMEO demonstration scenario

As described in Figure 20, the final ROMEO demonstrator will address different users having different terminals. Each terminal has its own video processing capability and dedicated video rendering will be provided depending on this capability. There are 3 different terminal types:

- The fixed terminal corresponding to the highest processing capabilities
- The portable terminal
- The mobile terminal

In the following section, respective processing will be presented showing adaptation of the processing to terminal capabilities.

3.2 Fixed Terminal

3.2.1 Experience acquired during the MUSCADE project (range of disparity)

In the field of the fp7 European project MUltimedia SCAlable 3D for Europe (MUSCADE), an end to end chain has been demonstrated from the multi-view capture up to a multi-view rendering. The project was targeting two different display technologies: multi-view auto-stereoscopic displays (e.g. Alioscopy 8-view display) and light-field displays. As described in [23] the light field display can render a field of view of about 70°. To achieve that, the MUSCADE content was shot to ensure a large range of disparity. Details of shooting are described in the MUSCADE deliverable [4] .

This very large range of disparity was not optimized for auto stereoscopic multi-view displays. There are too main drawbacks working which such high range of disparities:

[4] http://www.muscade.eu/deliverables/D2.2.2.PDF

- For transmission purposes, the disparity value should be encoded using a standardized 8-bit encoder (2 disparity maps encoded together using MVC encoder). To fit with the 8-bit limitation, the disparity between satellite and central cameras is normalized compared to the one between central views. After transmission, on the rendering side, the 8-bit representation is too low to allow a precise Depth Image based Rendering (DIBR) processing. It is not possible to process the disparity with a sub-pixel precision.
- Large disparities on multi-view display can create undesirable cross-talk. Only a limited number of views are displayed on such screen. Any misalignment of the lens array on the screen or/and a wrong position of the user create a cross-talk. The level is directly related to the range of disparity. The current level of the technology cannot ensure any cross-talk free system. It is then important to ensure that the level of disparity is not too high. The MUSCADE content shot with large baseline (above 30cm) was then processed to avoid such problem.

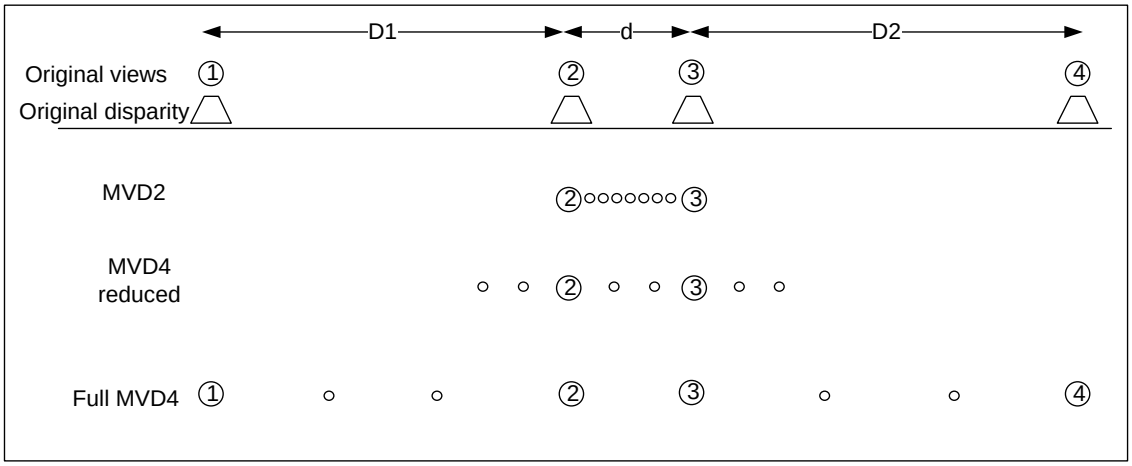

Figure 21: MUSCADE camera and relative view interpolation positioning

As described in Figure 21, the camera baselines are different between central cameras (d) and between central and satellite cameras (D1 or D2). The ratio between D1 and d is between 4 to 5. In MUSCADE, a scalability was defined whether the end user receive the full content (called MVD4) or only part of it (MVD2 corresponding to central views). In Figure 21 we describe the relative position of interpolated in the different configuration. In MVD2 case, only central views are received and interpolated are in between central views (2 and 3). In theory in MVD4 case, we should apply a scheme corresponding to the full MVD4 mode. But the range of disparity induce by such mode was too high. The cross-talk was just not acceptable. It was then mandatory to modify the interpolation scheme by creating this reduced mode. The relative disparity between each view is globally 1/3 of the disparity between central views.

3.2.2 Range of disparity defined for ROMEO

To avoid having too large range of disparity, we defined a camera set-up with smaller baseline for satellite cameras than in MUSCADE. The specification of the shooting have been described in the ROMEO deliverable D3.1: *Report on 3D media capture and content preparation.*

In Figure 22 The camera and relative view interpolation positioning are described. One of the camera set-up chosen in Romeo is an equivalent distance between each camera (called DR). One associated interpolation format is described in Figure 22. An adaptation to user preferences will be described in D3.4 deliverable.

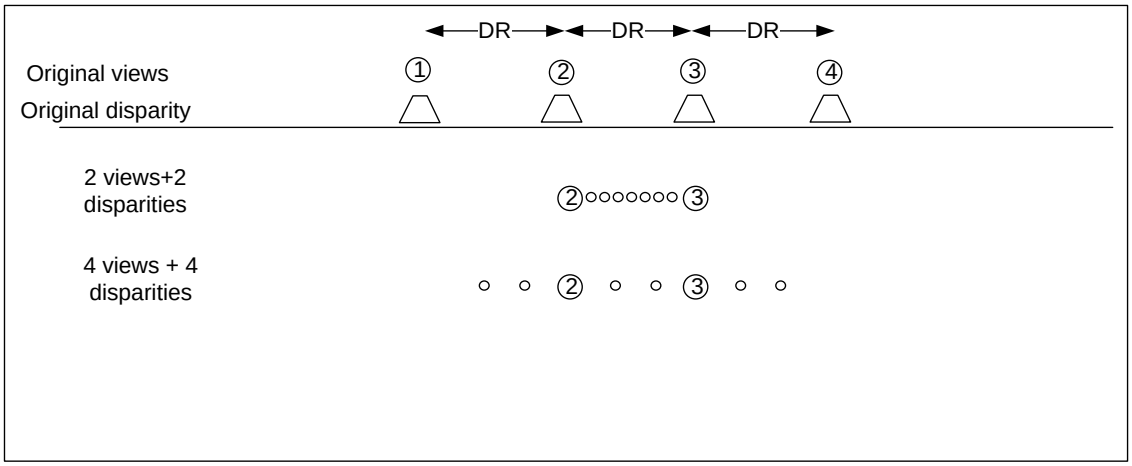

Figure 22: ROMEO camera and relative view interpolation positioning

3.2.3 Depth Image Based Rendering Algorithm improvements

3.2.3.1 Overview

To address the multi-view auto-stereoscopic displays it is necessary to generate extra views. Also for depth adjustment, it is necessary to interpolate intermediate views corresponding to the 3D level requested by the user. In both cases, these views are interpolated using a Depth Image Based Rendering (DIBR) algorithm [24]. These interpolations are done using the nearest available views on the each side of the interpolated view as well as the disparity map associated to this left view (the nearest available view on the left).

The disparity map corresponding to the interpolated view is first interpolated. During this disparity map projection, information on the occluded areas (part of the 3D scene that is not visible in one view) is determined. And then the video information in the left and the right views pointed by the interpolated disparity values are used and combined depending on the occluded area information to render the interpolated view.

3.2.3.2 Disparity map interpolation

The main part of the disparity map interpolation consists in projecting the left disparity map. Each disparity value is first scaled, i.e. multiplied by a coefficient between 0 and 1 corresponding to the position ratio of the interpolated view in relation with the corresponding original views considered for the interpolation. The coefficient is near 0 if the interpolated view is near the left view. This coefficient is near 1 if the interpolated view is near the right view. Then each disparity value is projected at the position defined by this scaled disparity. This position is rounded to the nearest pixel location (Figure 23).

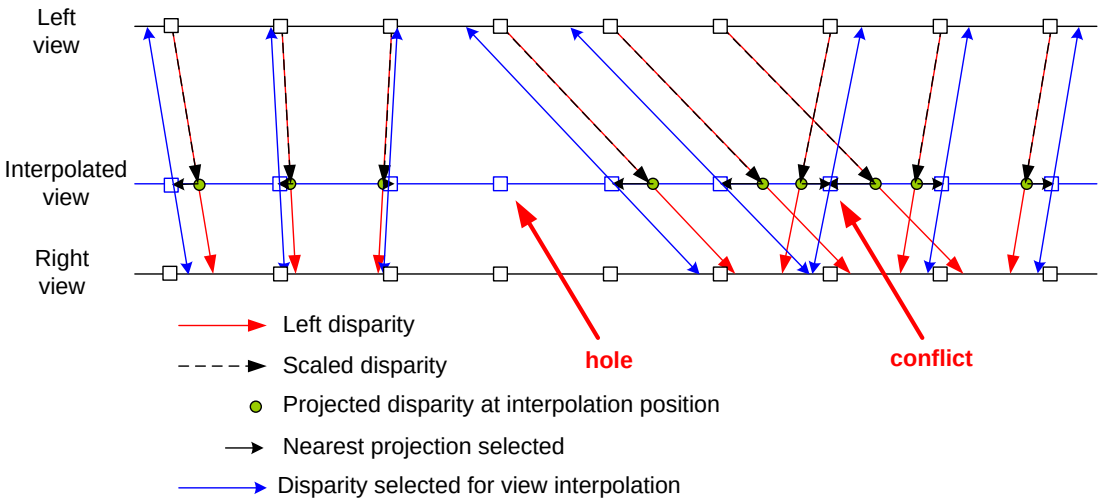

Figure 23: Projection of the left disparity map

In order to solve conflicting disparity values (different disparity values pointing to a same pixel location), the left disparity map is scanned from left to right so that information concerning occluding pixels overwrites information concerning occluded pixels. In order to fill the disoccluded areas (the holes), during the projection each disparity value is assigned to all pixels on the right side of the previously assigned pixel up to the pixel the current disparity value points at.

3.2.3.3 Occlusions identification

It is necessary to identify the regions of the interpolated view that are only visible in one view in order be able to render them using only this view. These regions correspond to parts of the scene that are occluded in the other view.

For this purpose, the object positions in both views are required. This information is obtained from the left disparity map, which contains the position of the objects in the left view. The position of the objects in the right view is obtained by projecting this disparity map onto the position of the right view using the same process as the one described in the previous section. The projection of the left disparity map is used instead of the provided right disparity map in order to avoid additional errors. The occluded areas are then identified by comparing the projection of both disparity maps at the synthesized view position.

Information on what is occluded in the right view is obtained by comparing the disparity values in the left disparity map and the disparity values they point to in the right disparity map. Positive differences indicate objects visible in the left view but occluded in the right view. This comparison process is applied by scanning the left disparity map from left to right and by projecting it onto the view to be interpolated.

Information on left occlusions view is obtained by a similar process by comparing the disparity values in the right disparity map and the disparity values they point to in the left disparity map. Positive differences indicate objects visible in the right view but occluded in the left view. This comparison process is applied by scanning the right disparity map is scanned from right to left and projected onto the view to be interpolated.

3.2.3.4 Occlusion border management

The occlusion information tells which view (left or right or both or none of them) can be used to render each pixel. Accordingly during video interpolation, in the occluded areas unidirectional

interpolation is used to synthesize the new view, while in the rest of the picture bidirectional interpolation is used, as shown in Figure 24.

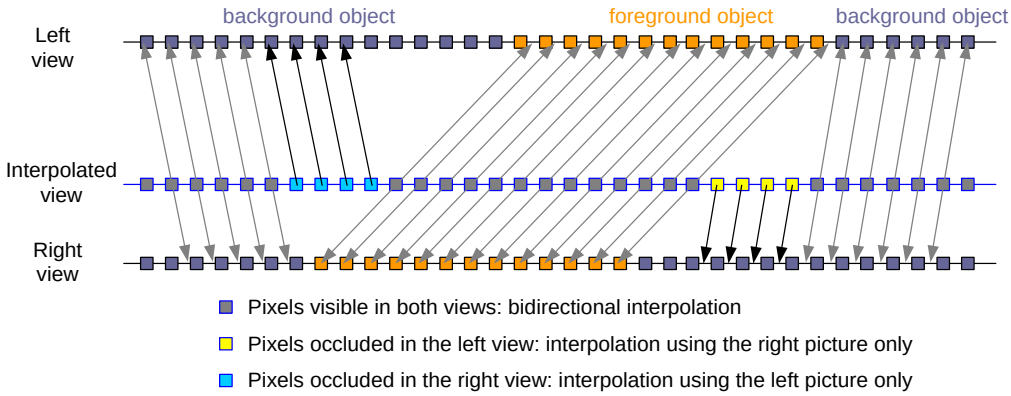

Figure 24: Interpolation scheme without border management

The abrupt transition between these unidirectional (in the disoccluded area) and bidirectional interpolations can however be visible for various reasons. For instance, the disparity map does not necessarily match perfectly the object borders. Particularly in case of blurred borders it is difficult to define where the object ends and where the background starts. This can generate echoes in the new view. Also differences of luminance, color, reflections or flare between left and right views can create halos around objects.

In order to reduce the visibility of the aforementioned artifacts, the transitions between the unidirectional and the bidirectional interpolations are softened on the borders of the occluded areas. The coefficients used for the bidirectional interpolation are changed on the right of the regions occluded in the left view, and on the left of the regions occluded in the right view (Figure 25).

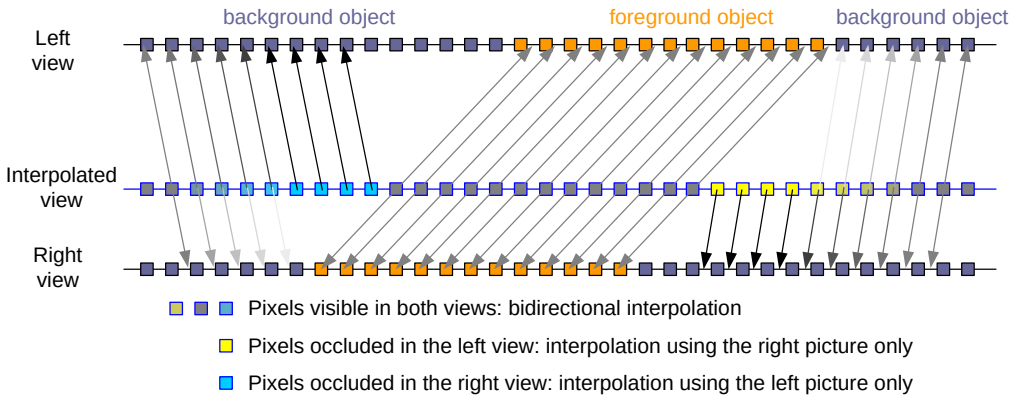

Figure 25: Interpolation scheme with border management

The value of the blending coefficient depends on the results of the occluded area borders management, resolving issues of areas located just between regions occluded in the left view and regions occluded in the right view, and also the case of regions occluded in both views.

The size of the altered area around the occlusion is defined depending both on the width of the occlusion and the difference of disparity between the background and the foreground objects. Thus since this disparity difference depends on the inter-axial camera separation a specific parameter has been introduced in order to avoid excessive filtering. Results are shown in Figure 26.

Figure 26: Interpolation results: without (top) and with (bottom) border management

3.2.3.5 De-aliasing

Aliasing can appear on the object borders during view synthesis [25]. This effect occurs because in the 1-layer disparity map format which is used, only one disparity value is stored per pixel. Therefore in the disparity map object borders are located at pixel borders, and thus also the object borders in the interpolated view.

Figure 27: Disparity map and corresponding interpolated video without de-aliasing

To solve this issue, the aliased borders are detected in the interpolated disparity map (by comparing the disparity values) and a sub-pixel shape is virtually added to smooth these contours. Therefore one line of pixels along the object borders uses a 2-layer representation, this means two disparity values, and thus a combination of background and foreground information.

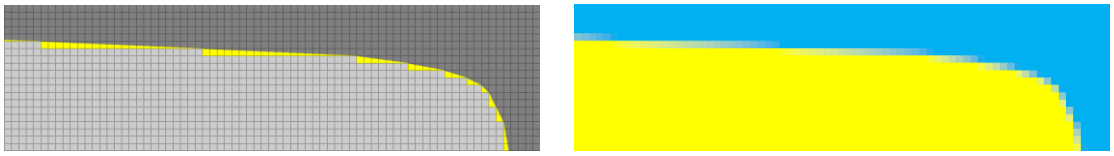

Figure 28: Disparity map and corresponding interpolated video with de-aliasing

This process is illustrated on a basic example in Figure 27 and Figure 28. Figure 27 shows on the left an interpolated disparity map and on the right the result of interpolation without de-

aliasing. Figure 28 shows on the left the same disparity map with the virtual shape of foreground information added onto the background of the aliased contour and on the right the corresponding result with de-aliasing.

Figure 29 shows the result on real content. It can be observed how the aliasing and the “cut-out” appearance are reduced.

Figure 29: Interpolation results: without (top) and with (bottom) de-aliasing

3.2.3.6 Interpolated view rendering

The video information of the view to synthesize is interpolated using the video samples of the left and the right views and the disparity value (or the two disparity values in case of a de-aliased contour). Linear interpolation is used to point at sub-pixel position in the left and the right views. Finally the video samples pointed by the disparity value(s) in the left and the right views are combined depending on occlusion information and the occlusion border management results.

3.2.4 Algorithm adaptation for parallelization

In order to reach real-time constraint, the Depth image based rendering described in the previous section must be integrated into a GP-GPU (General Purpose Graphic Processor Unit) processor. The target processor is a Nvidia GPU processor which allow developing code in CUDA.

3.2.4.1 CUDA programming model

The CUDA programming model is based on an extension of C, allowing the programmer to define C functions called “kernels”, that will be executed N times in parallel by different CUDA threads on the GPU. It means that each thread executes the same C code, applied to different data, depending on the thread index.

These threads are group in “warps”. The number of thread per warp depends on the hardware architecture (32 in our case) and cannot be modified by the programmer. All the threads of a warp will execute the same instruction. It means that if only one thread have to execute a part of code after a conditional test, all the other threads of the warp have to wait during this

execution, reducing the global efficiency. By definition, all the threads of a warp are synchronized.

Then, the threads are organized in “blocks” of threads, defining a “grid”. The total number of threads executed by a kernel is equal to the number of blocks multiplied by the number of threads per block. The number of threads per block and the architecture of a block (1D, 2D or 3D) have to be defined by the programmer, depending on the problem to be solved, and performance considerations.

The variables used in the thread’s C code will be affected by the compiler to registers. The number of registers is limited in each multiprocessor composing the GPU. If too many registers are required, the number of blocks of threads resident in a multiprocessor is reduced, reducing the device occupancy.

In a block, the threads can be explicitly synchronized by the programmer and can share a small amount of memory, the “shared memory”. If this amount of shared memory required per block is too high, the number of blocks of threads resident in a multiprocessor is reduced.

3.2.4.2 View interpolation CUDA kernel

The auto-stereoscopic monitor is a HD display (1920 x 1080 pixels) where eight views are shuffled, at a sub-pixel level. It means that each view composing the multi-view content has a resolution of 1/8 HD. In comparison, on a 3D stereoscopic monitor supporting side by side input video, the horizontal input resolution is divided by two.
Since the final resolution of the interpolated picture is lower than full HD and in order to limit the amount of shared memory required by thread block, it seems an acceptable trade off to perform the disparity projection with a horizontal resolution of 960 disparity pixels per line.
Even if the target hardware for Romeo is a Tesla board having a CUDA 2.0 capability and implying a maximum of 1024 threads per block, the CUDA code has been developed to be compatible with CUDA 1.3 capability , to support hardware of the previous generation. In this case, the maximum number of threads per block is 512.

The number of threads per block has been set to 480 and these blocks are 1D blocks (blocDim.x=480, blocDim.y=1). The 480 threads of a block are executed in parallel to perform the projection of a full disparity line. So, each thread has to process 2 disparity pixels. To process a full picture, a grid composed of 1080 blocks of threads have to be launched. It means that 518400 threads are launched to perform a projection. Since a line is composed of 1920 pixels (or 5760 sub-pixels), each thread has to compute 4 RGB output pixels (or 12 sub-pixels).

To save memory access time and to allow threads cooperation, shared memories have been used to perform the projection. These memories are faster than global memory and can be shared by all the threads of a block of threads. The shared memory DispIn[960] receives the input disparity, DispOut[960] the projected disparity and RightView[960] a mask indicating the de-occluded areas.
The variable ‘x’, represents the horizontal position of the disparity pixel to be processed by the current thread. Since each thread has to compute 2 disparity pixels, each thread has to compute the interpolated disparity pixels ‘x’ and ‘x+1’. The thread 0 computes the pixels 0 and 1, the thread 1 computes the pixels 2 and 3…, until thread 479 computing the pixels 958 and 959.
The first step consist in initializing the DispIn[960] shared memory. Each thread initializes DispIn[x] and DispIn[x+1] with the disparity values.
Then the pixel disparity from x-Dmax to x-Dmin+1 (Dmax and Dmin representing the disparity range) are projected, as indicated in Figure 30. If a projected disparity provides an output disparity for pixels x or x+1, the disparity value is stored in DispOut[x] or DispOut[x+1]. When several disparities are projected to x or x+1, indicating an occlusion, the smaller disparity value is stored, corresponding to the front plane.

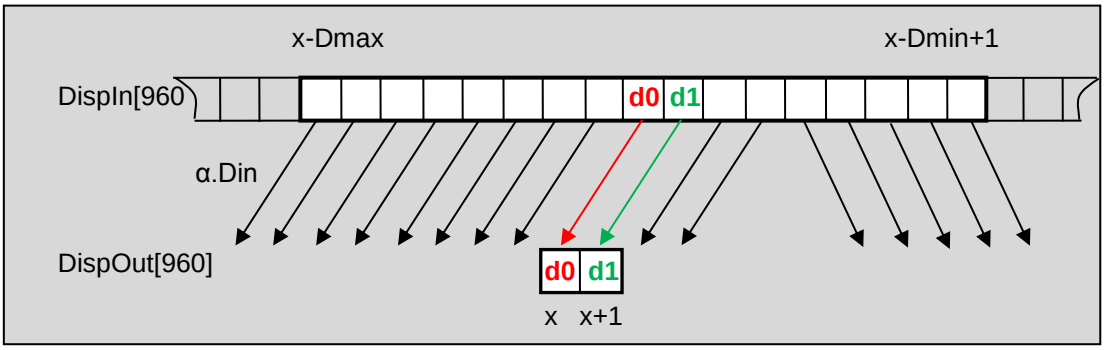

Figure 30: Disparity projection per thread

When a block of thread has been executed, the DispOut[960] shared memory contains the projected disparity, as presented in Figure 31. The occluded part in the left view did not receive any disparity value. In this case the RightView[960] shared memory receives the value 1, indicating that the projected video view will receive the pixels from the right view.

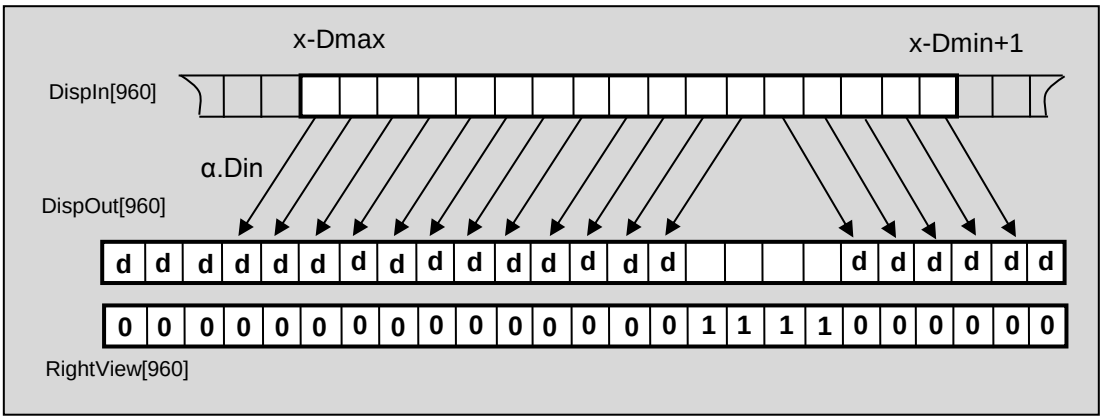

Figure 31: Projected disparity by all the threads

Then, the holes in the projected disparity map corresponding to occluded areas are filled with the background values. To minimize artefact close to the right to left view transition, a filter is applied on RightView[960] (Figure 32).

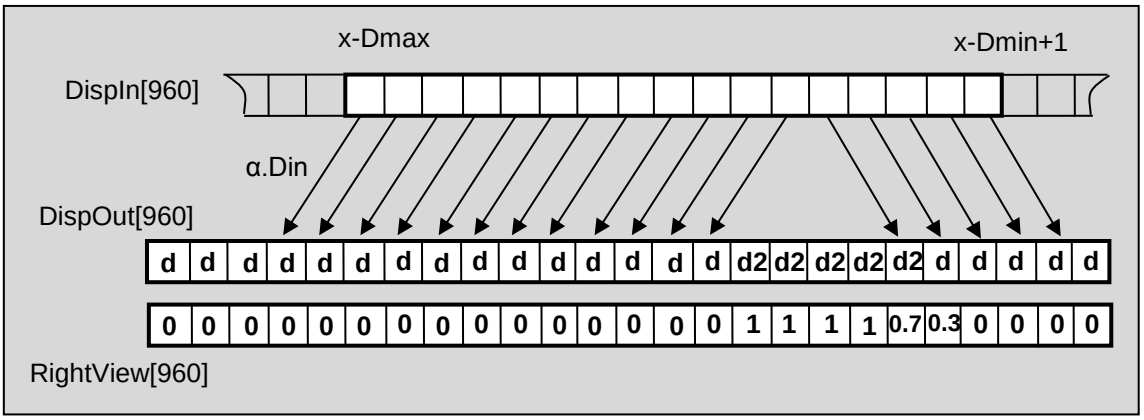

Figure 32: Projected disparity after filtering

These information DispOut and RightView are used to create a new interpolated video view, thanks to input video views RightView and LeftView, with the relation:
Out[x] = RightView[x].RightRGB[x] + (1- RightView[x]).LeftRGB[x].

In fact, only the sub pixels corresponding to the view currently projected are computed: the disparity is projected to compute the disparity map Di. Then for each output sub-pixel, if this sub-pixel has to display the view Vi (according to the scheme described in 3.2.2), the value of this projected sub-pixel is computed and written to the output auto-stereoscopic image.
Disparity projections depend on the number of views to be displayed and on the number of input views, but they do not depend on the way the monitor shuffles the views. Only the final step of sub-pixels interpolation depends on this shuffling. It means that disparity projections are the same for several different auto-stereoscopic monitors displaying the same number of views, only the interpolation step has to be customized.

To perform this interpolation operation, the input views RightView and LeftView must be coded in RGB. The decompression step provides a YUV 4:2:0 video stream, where the chroma information is sub-sampled in both directions. Another CUDA kernel has to be developed to convert YUV 4:2:0 to RGB 4:4:4.

3.2.5 Implementation into a GP GPU system

The CUDA kernel, as described in the previous paragraph, is a massively parallel code executed by the GPU, called the device. But CUDA is an heterogeneous language, allowing to program C code to be executed by the device, and executed by the host represented by the CPU or the multiple CPUs. The CUDA programming model assumes that the CUDA threads are executed on a physically separated device that operates as a coprocessor to the host running the C program. Both the host and the device maintain their own separated memory spaces in DRAM, referred to as host memory and device memory, respectively.

Therefore, a program manages global, constant and texture memory spaces visible to kernels through calls to the CUDA runtime. This includes device memory allocation and de-allocation as well as transfer between host and device memory. The data to be processed must be transferred from host memory to device memory, then the result of computation must be transferred from device memory to host memory (if the GPU is not used as a graphic card displaying the computation result on a monitor).

To be optimized, a CUDA based system should take advantage of the programming model to parallelize as much as possible the different operations. This includes, the code executed by

the CPUs that can be multi-threaded, the data transfers in both directions that can be performed in parallel with the kernel execution, and the massively parallel threads of the kernel executed by the GPU.

The following diagram (Figure 33) represents the functional description of the CUDA rendering object, in a PC environment. This intermediate step was defined to help the integration, it can be considered as a stand-alone video player. It presents the different buffers allocated in the host (CPU) memory space or in the device (GPU) memory space, the data flow, and the processing operations.

The video buffers DataVid[0..3] and DataDisp[0..3] containing the frames of decompressed video and disparity, are allocated by the player part in the CPU memory space. The video buffers Host_VidPin[0..3], Host_DPin[0..3] and Host_ResultPin are also allocated in the CPU memory space, but these allocations are performed in CUDA, by the object in the rendering library.

The decompressed video frames are provided in 4:2:0, planar. The computed result is transferred in Host_ResultPin, in RGBA format (four bytes per pixel), HD 1920x1080 resolution.

The GPU board can be considered as a computing resource (a GP GPU), receiving the MVD4 video stream and providing the multi-view result to be displayed to the CPU. Optionally, the multi-view stream could be directly displayed, exploiting the possibility of interoperability between CUDA and OpenGL for instance. In this case the output buffer is a buffer shared by CUDA and OpenGL, and the GPU board is considered by the system as a graphic card.

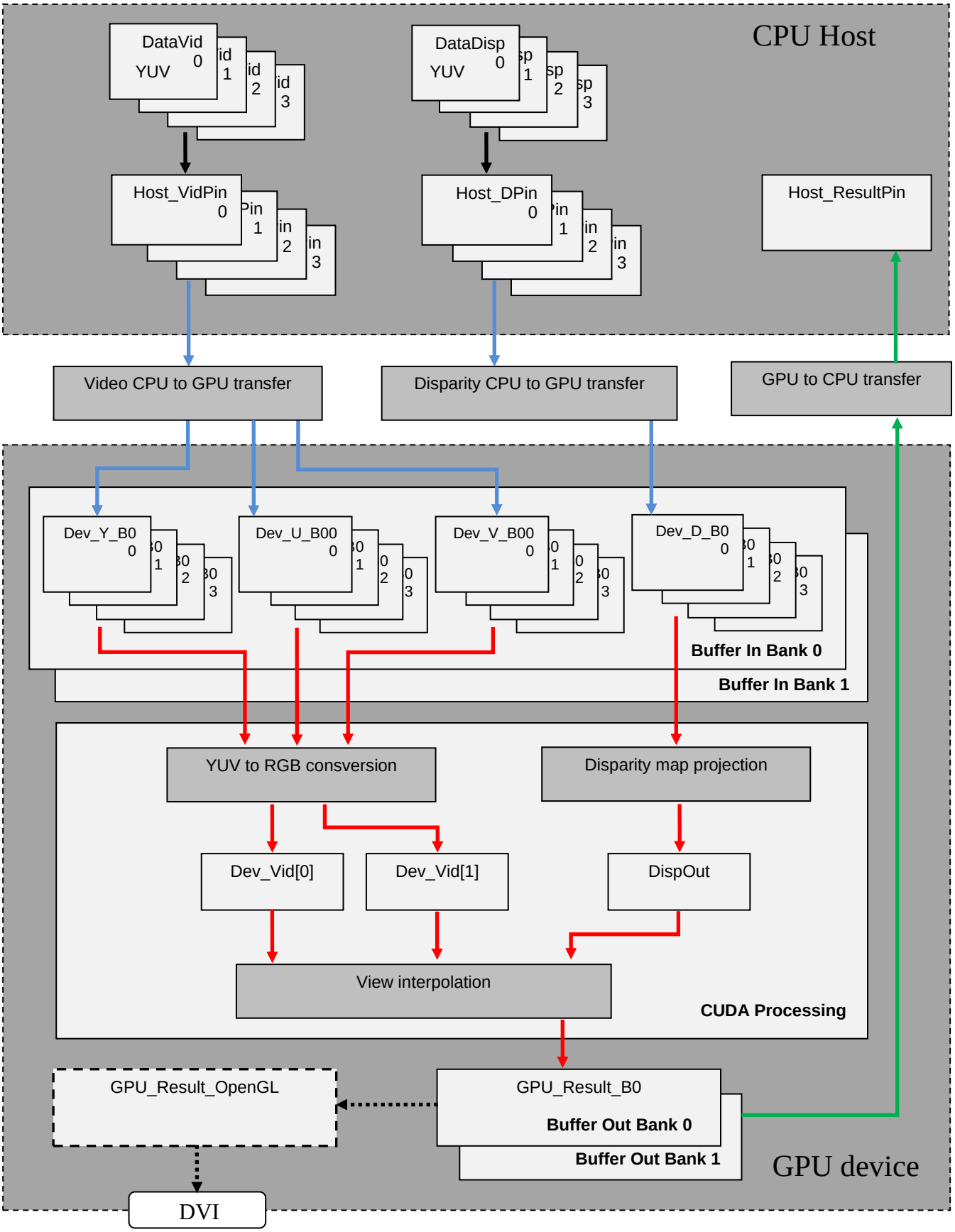

Figure 33: Functional description

To be executed concurrently, the transfer operations and CUDA computing must be able to access to different memory banks. The input video stream, comprising the video and the disparity, can be written by the CPU in memory Bank 0 or memory Bank 1. The CUDA processing can process one of the memory banks during the transfer of the other. In the same way, the multi-view result can be written by the GPU in two memory banks. This result stored in one of the memory banks can be transferred to the CPU during the processing of the other. These concurrent operations, represented by the different colors, will be detailed later in this document.

To be transferred efficiently, the data to be transferred from or to the GPU must be stored in page-locked CPU memory, also known as pinned memory. This memory guaranties that the data are physically present in CPU memory for the transfer, and allows concurrent transfers and CUDA kernel execution. Since this memory is no more accessible by the operating system, the application must minimize the amount of allocated page-locked memory to avoid decreasing the overall system performance.

Even if only two views are required to compute each projection/interpolation, the four video views are transferred in a memory bank during the CUDA processing of the other memory bank. After decoding, these views are provided to the GPU in YUV 4:2:0. The buffers Dev_Y_Bi[0..3] receive the Luma of the four input views, and the buffers Dev_U_Bi[0..3], and Dev_V_Bi[0..3], receive the two planes of chroma. In 4:2:0 mode, the Luma buffers have the same size as the input video resolution, and the chroma buffers have a half resolution in both directions.

The 4:2:0 to 4:4:4 and the YUV to RGB transformations are performed by the GPU in CUDA to save time, compared to the CPU. Firstly, the amount of data to be transferred from the CPU to the GPU memories is reduced in 4:2:0. Secondly, the computation can be massively parallel and take advantage of the GPU architecture. For an input video size of SizeH x SizeV pixels, a CUDA kernel executes SizeH x SizeV / 4 threads (518.400 threads in HD) in parallel. The results of these operations are written in the buffers Dev_Vid[0] and Dev_Vid[1].

All these buffers are allocated in global memory. To accelerate the reading operations to these buffers, the data are accessed thanks to textures. These textures are not represented in the block diagram, but are dynamically bound to the buffers before the reading operations.

The disparity has to be transferred from the CPU to the GPU memory in the buffers Dev_D_Bi[0..3]. This disparity is coded with eight bits in the Luma channel of the video stream. To transform this eight bit disparity to a floating point disparity representing a real disparity in term of horizontal pixels, some parameters may be required. These parameters could be fixed, coded as constants, or could be dynamic depending on the video stream, and provided as metadata.

During the disparity projection, all the threads of a video line must be able to access to the same memory buffer DispOut. This buffer must be declared as a shared buffer. The position of the projection in between the two input views is provided by a phase parameter. When a line of disparity has been projected to this position, the video output interpolation can be performed. Then the next line is processed. Only the output sub-pixels representing the current projected view are computed. For a eight view rendering, requiring eight disparity projections Pi, the projection Pi (with i from 0 to 7) is performed and the output sub-pixels dedicated to the view i are written in the output BufferOut. It means that after each projection Pi, only 1/8 of the interpolated sub-pixels are computed and written in the buffer. The number of sub-pixels to be computed is imposed by the auto-stereoscopic monitor, 3x1920x1080 for a HD monitor.

But the resolution of the input video can be (and should be) lower than HD. The video buffers Dev_Vid[0], Dev_Vid[1] are read via a texture programmed to perform an automatic linear interpolation.

The previous diagram (Figure 33) is a static view presenting the hardware architecture. The following diagram (Figure 34) is a dynamic representation of the different operations. It shows the level of system parallelism where the memory transfers in or from the two memory banks, the GPU processing and the CPU processing can be performed concurrently.

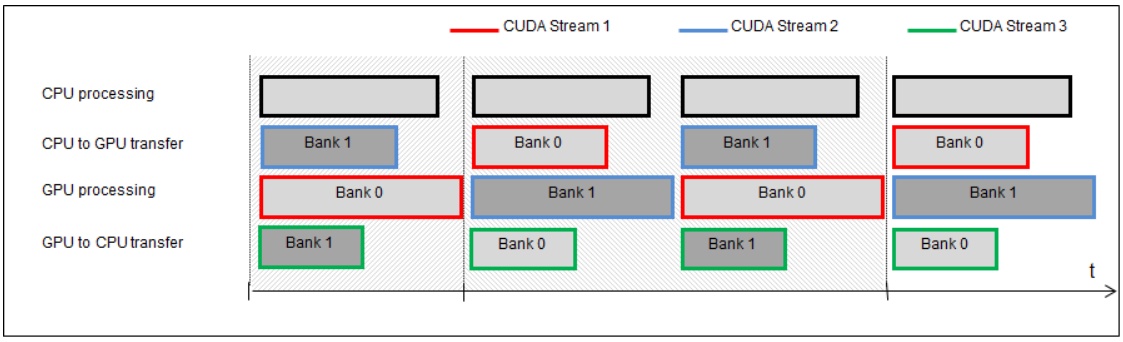

Figure 34: Concurrent execution

In order to facilitate concurrent execution between the CPU host and the GPU device, some function calls are asynchronous: control is returned to the host immediately after the call.

A CUDA application can manage concurrency through streams. Thanks to these streams, associated with the asynchronous function calls, a system can simultaneously transfer data from the CPU memory to the GPU memory, process some data available in GPU and transfer other data from the GPU memory to the CPU memory.

A stream is a sequence of commands that are executed in a given order (sequential execution). Different streams, on the other hand, may execute their commands out of order with respect to one another or concurrently. It means that commands that could be executed concurrently must be launched in different streams. But if some commands launched in different streams have to respect external dependencies, a synchronization mechanism must be implemented. For instance, the interpolated video frame Fi-1 can be transferred from the GPU memory to the CPU memory, while the GPU is processing the frame Fi. To be executed concurrently these two operations have to be launched in two different streams. But the GPU to CPU memory transfer of the interpolated frame Fi must only begin when the processing of this frame is finished. Since these operations are launched in two different streams, an explicit synchronization mechanism is required.

To be able to perform three different operations concurrently, we must use three CUDA streams. These three streams are represented in red, blue and green. They are associated to two memory banks. The stream 1 accesses to the memory bank 0 and the stream 2 accesses to the memory bank 1. The stream 3 is optional; it performs the GPU to CPU memory transfer when this operation is required (for instance, this operation is not required if the processing result is displayed directly from the graphic card to an external monitor). It must be synchronized with the other streams to wait for the end of processing.
The following graph (Figure 35) presents the way the streams are managed. Since the functions launching the memory transfers and the CUDA kernel are asynchronous, the calls are not blocking, allowing different streams to be executed concurrently.

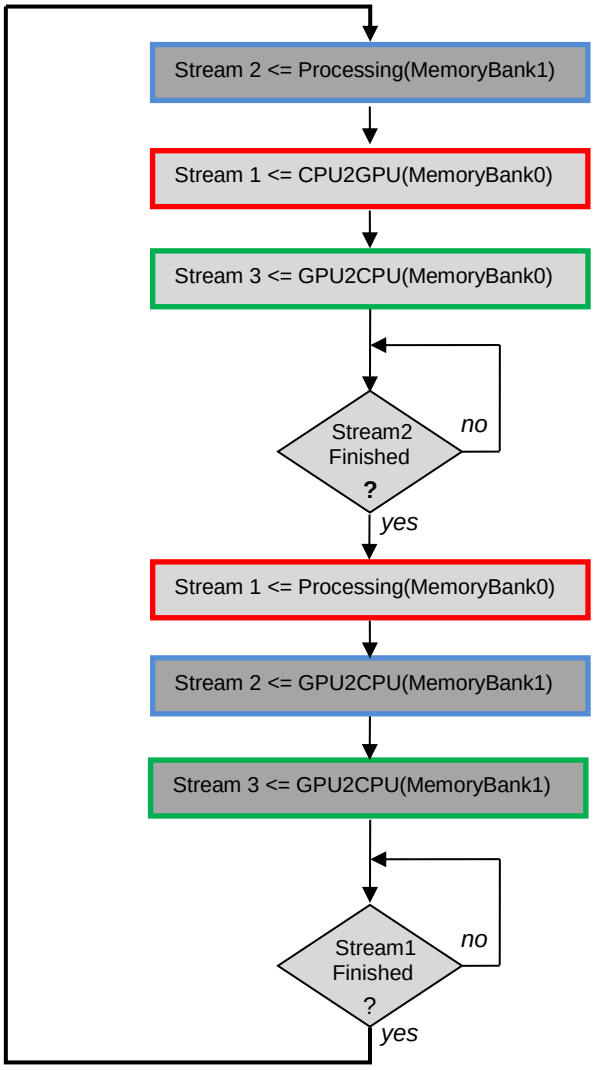

Figure 35: Asynchronous launches and CUDA streams

3.3 Portable Terminal

The portable terminal of the ROMEO project targets devices such as laptops equipped with an autostereoscopic display. Rendering algorithms targeted for such terminal are similar to the ones described for fixed terminal. But of course the implementation will be simplified to be compatible with hardware constraints.

3.3.1 Free view point

the video rendering applied on the portable terminal is depending on the display technology.Some laptops have today auto-stereoscopic display. They will be able to display stereo content (2 views). The goal of the video rendering processing is to deliver this stereo from ROMEO available content. As any ROMEO terminal is receiving DVB stereo content it is possible to display it directly after decoding. Nevertheless to address user requirement, it is possible to consider changing the view point as soon as additional information are available. The minimum required is the availability of disparity maps associated to the central views. In

this case it is possible to render new video frame from central views and then create the expected view point.
Technology required for such viewpoint modification is similar to the one described for the fixed terminal. It is based on DIBR technology. Figure 36 illustrates the different scenarios possible to generate stereo from ROMEO content:

- The simple stereo scenario does not require any DIBR processing. It is only displaying the incoming DVB content.
- The "viewpoint from central views" requires disparity maps from central views to allow the interpolation of one view in between view 2 and 3. The depth of the displayed signal is adapted to the user requirement. It can be anywhere between the view 2 (lower depth) and the view 3 (higher depth) when combined with view 2.
- The last case is more general considering we can have access to more than central views information. On Figure 36, the view 1 is decoded and used in combination with a view generated from view 2 and view 3. In this case the depth level is higher than the one in the simple stereo case.

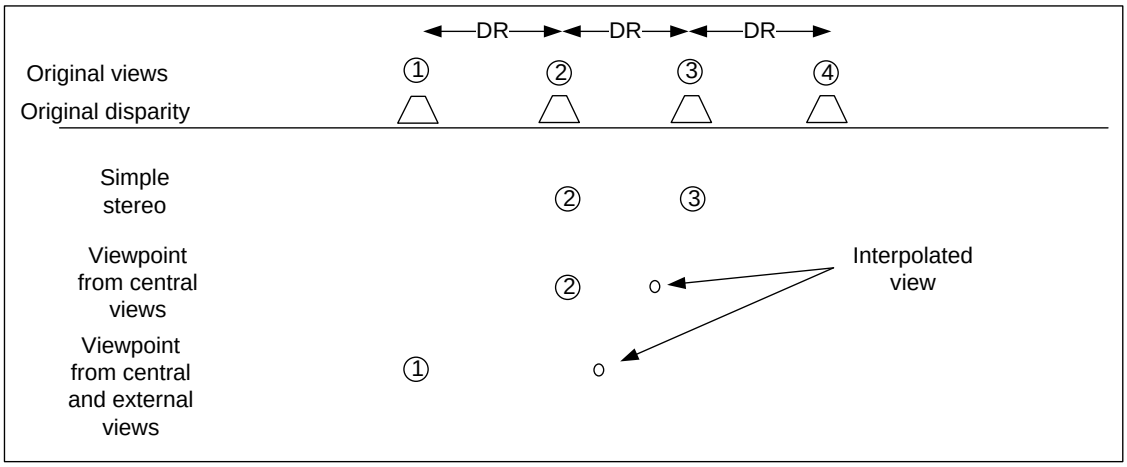

Figure 36: View interpolation for portable terminal

3.4 Mobile Terminal

3.4.1.1 Introduction

Because the graphics subsystem of the mobile processors is not very powerful a depth image based rendering algorithm pipeline cannot be realized in real time. That is why the Romeo system integrates video encoding formats via side-by-side sub-sampling so the devices with reduced computational power can render stereo video streams. The normal video rendering in the mobile terminal is realised in the display subsystem using OMAP4/5 hardware. For the second stage of the project an experimental implementation of an algorithm for depth image based rendering using GFX of the OMAP with reduced frame rate and resolution will be tested.

3.4.1.2 Stereo rendering in Mobile Terminal Device

Mobile terminal device supports top-bottom, left-right, full-frame and sub-sampling modes. Video rendering is realized in the display subsystem, where the buffers, rotation and pixels interleaving are managed. For this purpose the assistance of the OMAP H/W is used.

3.4.1.3 OMAP Display Subsystem Capabilities

OMAP family processors include several HW engines to support different display devices:

- 1 Graphics pipeline (GFX)
- 3 video pipelines (VID1-VID3)
- 1 Write-back pipeline (WB)
- 2 LCD outputs (LCD1-LCD2)
- 1 TV output
- Panel support with MIPI DPI protocol

Common supported features are:

- Rotation 0, 90, 180, and 270 using DMM-TILER engine
- Synchronized buffer update
- Hardware cursor (using the graphics pipeline or one of the video pipelines)
- Independent gamma curve support on primary LCD output, secondary LCD output and TV output
- Multiple-buffer support
- Mirroring/Flip-flop support (using DMM-TILER engine)
- Programmable color phase rotation (CPR)
- Alpha blending support

3.4.1.4 Stereo rendering pipeline implementation

On the Figure 37 is shown stereo rendering pipeline implementation on OMAP HW for complex case with top-bottom full-frame stereo format. The pipeline is realized using several blocks:

- Vid1 and Write-back - used for video rotation and scaling. The output is in the memory.
- Vid3 - send the stream to LCD1 and DSI (display serial interfaces) to Stereo LCD device

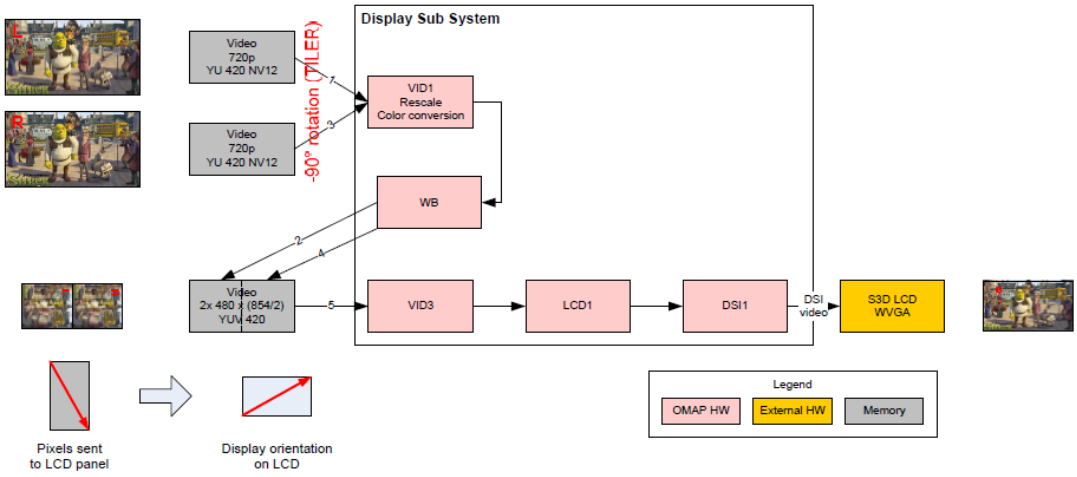

Figure 37: Stereo rendering pipeline example

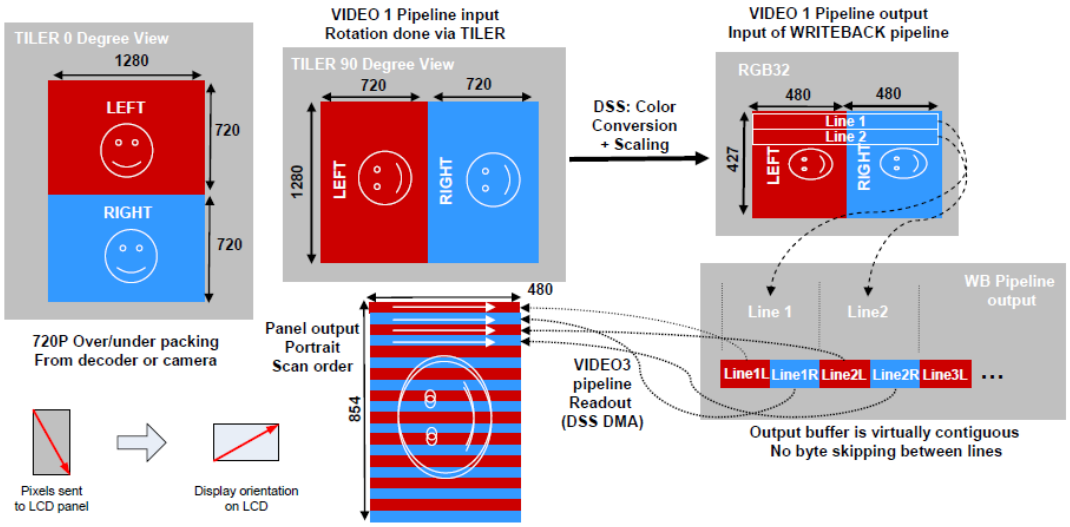

Figure 38: Stereo rendering pipeline operation

On Figure 38 is shown the diagram of pipeline operations:

Stereo Video frames 1080p/720p are in YUV420p format (12bpp) and sent alternatively each at 30 fps to VID1 path. (with/without 90deg rotation). If the case is "Top-Bottom" frames alignment and a 90-degrees rotation, the VID1 path is used.

- Video frames resized to 480 x (854/2) in YUV420p format at 30 fps.
- WVGA image in YUV420p format sent to LCD at 60 fps.
- In the second stage of the project a simplified version of the "Depth Image Based Rendering Algorithm" will be implemented on Mobile Terminal Device for experimental purposes. The algorithm will be tested with streams with reduced frame rate and sizes.

4 CONCLUSIONS AND FUTURE WORK

This deliverables describes initial studies and first developments done in the field of audio and video rendering algorithms. It is related to the task 3.4 of the project. This is an interim version of the deliverable which will be finalized end of the second year (M24).

In this document, various audio and video rendering techniques/algorithms are introduced including the framework and the system to implement the required features. An overview of the audio and video rendering work is provided for the different terminals used in ROMEO. The algorithms and principles (state-of-the-art and beyond) are described as well as the current status of the implementation.

More specifically on the audio part, for mobile terminal, binaural room synthesis is presented as well as an implementation of this technology on an ARM processor. The use of a mobile head-tracking device is also described. Object based rendering technology is presented and the field of portable terminal.

For fixed terminal, multi-channel audio rendering as well as Wave Field Synthesis (WFS) are described. The audio rendering framework is then presented including a simplified version of the WFS.

On the video part, the basic DIBR algorithm is presented including recent algorithm improvements. The experience acquired from another European project MUSCADE is described. Multi-view rendering for fixed terminal is described as well as simplified version for portable terminal. The stereo rendering pipeline of the mobile terminal is also presented.

These algorithm descriptions will be completed for the final version of this document (due M24) including the final achievements of this Task 3.4.

5 REFERENCES

[1] J. Blauert, “Spatial Hearing, the Psychophysics of Human Sound Localization”, Cambridge, MA: MIT Press, 2001

[2] C. I. Cheng, and G. H. Wakefield, “Introduction to head-related transfer functions HRTF‘s: Representations of HRTF‘s in time, frequency, and space”, J. Audio Eng. Soc., vol. 49 no. 4, pp. 231–249, 2001

[3] F. Rumsey, “Spatial Audio”, 2nd ed. Oxford, England: Focal Press, 2001

[4] P. Mackensen, “Auditive localization head movements, an additional cue in localization”, Ph.D. dissertation, Technische Universitat Berlin, Berlin, Germany, Jan. 2004

[5] U. Zolzer, “DAFX - Digital Audio Effects”, 1st ed. Chichester, England: JohnWiley Ltd, 2002

[6] D. Menzel, H. Wittek, G. Theile, and H. Fastl, “The Binaural Sky: A Virtual Headphone for Binaural Room Synthesis”, In: Tonmeistersymposium, 2005

[7] U. Felderhoff, P. Mackensen, G. Theile, “Stabilität der Lokalisation bei verfälschter Reproduktion verschiedener Merkmale der binauralen Signale” [Stability of localisation versus distorted reproduction of binaural cues], 20. Tonmeistertagung, 1998

[8] Potard, G: „3D-audio object oriented coding“, *University of Wollongong, Diss., September 2006*

[9] Geier, M.; Ahrens, J. and Spors, S.: „Object-based audio reproduction and the audio scene description format“, *Organized Sound*, 15(3): 219-227, October 2010

[10] G. Theile and H. Wittek, “Principles in surround recordings with height,” in 130th AES Convention London, no. 8403. AES, 2011

[11] Pulki, V.: “Virtual Sound Source Positioning Using Vector Based Amplitude Panning”, J. Audio Eng. Soc., vol. 45, no. 6, 1997

[12] Vorländer, M.: “Auralization – Fundamentals of Acoustics, Modelling, Simulation, Algorithms and Acoustic Virtual Reality”, Berli Heidelberg, Spinger, 2008

[13] S. Spors, H. Teusch, and R. Rabenstein, “High-Quality Acoustic Rendering with Wave Field Synthesis,” Vision, Modeling, and Visualization, pp. 101-108, 2002.

[14] A. D. Pierce, “Acoustics. An Introduction to Its Physical Principles and Applications,” McGraw-Hill, Inc., 165, 1991.

[15] E.G. Williams, “Fourier Acoustics, Sound Radiation and Nearfield Acoustical Holography,” Academic Press, 1999.

[16] D. de Vries, “Sound Reinforcement by Wave Field Synthesis: Adaption of the Synthesis Operator to the Loudspeaker Directivity Characteristics,” J. Audio Eng. Soc., 44 (12), pp. 1120-1131, 1996.

[17] J. Ahrens, R. Rabenstein, and S. Spors, “The Theory of Wave Field Synthesis Revisited,” Audio Engineering Society Convention, 124, 2008.

[18] R. B. Randall, and B. Tech., “Frequency Analysis, Brüel & Kjær,” Denmark, 1987.

[19] G. Theiler, and H. Wittek, “Wave field synthesis: A promising spatial audio rendering concept,” Acoust. Sci. & Tech. 25, 6, 393-399, 2004.

[20] J.-J. Sonke, ‘‘Variable acoustics by wave field synthesis,’’ Thela Thesis, Amsterdam, Netherlands, ISBN 90-9014138-3, 2000.

[21] S. Spors, A. Kuntz, and R. Rabenstein, “An approach to listening room compensation with wave field synthesis,” Proc. AES 24th Int. Conf., pp. 70–82, 2003.

[22] E. Corteel, and R. Nicol, “Listening room compensation for wave field synthesis. What can be done?,” AES 23rd Int. Conf., Paper 7, 2003.

[23] T. Balogh, “The HoloVizio System,” Proc. SPIE 6055, (2006)

[24] C. Fehn, “Depth-image-based rendering (DIBR), compression, and transmission for a new approach on 3D-TV”, Proc. SPIE 5291, Stereoscopic Displays and Virtual Reality Systems XI, 93 (May 21, 2004)

[25] P. Robert, C. Thébault, P.H. Conze, “Disparity-compensated view synthesis for S3D content correction”, Proc. SPIE 8288, Stereoscopic Displays and Applications, XXIII, 82882F (February 6, 2012)

APPENDIX A: GLOSSARY OF ABBREVIATIONS

B	
BMP	Bitmap
C	
CA	Consortium Agreement
D	
DoW	Description of Work
E	
EC	European Commission
G	
GA	Grant Agreement
GM	General Meeting
J	
JPEG	Joint Photographic Experts Group image compression
P	
PB	Project Board
PC	Project Coordinator
PNG	Portable Network Graphics
Q	
QAT	Quality Assurance Team
QMR	Quarterly Management Report
T	
TIFF	Tagged Image File Format
W	
WP	Work Package