Pontifícia Universidade Católica
do Rio de Janeiro

Roberto Gerson de Albuquerque Azevedo

Supporting Multimedia Applications in Stereoscopic and Depth-based 3D Video Systems

TESE DE DOUTORADO

Thesis presented to the Programa de Pós Graduação em Informática of the Departamento de Informática, Centro Técnico Científico, PUC-Rio as partial fulfillment of the requirements for the degree of Doutor.

Advisor: Prof. Luiz Fernando Gomes Soares
in memoriam

Rio de Janeiro
December 2015

Pontifícia Universidade Católica
do Rio de Janeiro

Roberto Gerson de Albuquerque Azevedo

Supporting Multimedia Applications in Stereoscopic and Depth-based 3D Video Systems

Thesis presented to the Programa de Pós Graduação em Informática of the Departamento de Informática, Centro Técnico Científico, PUC-Rio as partial fulfillment of the requirements for the degree of Doutor.

in memoriam
Prof. Luiz Fernando Gomes Soares
Orientador
Departamento de Informática – PUC-Rio

Prof. Alberto Barbosa Raposo
Departamento de Informática – PUC-Rio

Prof. Sérgio Colcher
Departamento de Informática – PUC-Rio

Profª. Simone Diniz Junqueira Barbosa
Departamento de Informática – PUC-Rio

Prof. Guido Lemos de Souza Filho
UFPB

Prof. Rudinei Goularte
USP

Prof. José Eugenio Leal
Coordenador Setorial do Centro
Técnico Científico – PUC-Rio

Rio de Janeiro, December 1st, 2015

Roberto Gerson de Albuquerque Azevedo

Roberto Gerson de Albuquerque Azevedo is an associate researcher at the TeleMídia Lab. in PUC-Rio. His research interests include the areas of three-dimensional multimedia and hypermedia authoring and systems. Roberto received his B.Sc. degree in Computer Science from the Federal University of Maranhão and his M.Sc. degree in Informatics from the Department of Informatics at PUC-Rio.

Bibliographic data

Azevedo, Roberto Gerson de Albuquerque

Supporting Multimedia Applications in Stereoscopic and Depth-based 3D Video Systems / Roberto Gerson de Albuquerque Azevedo ; advisor: Luiz Fernando Gomes Soares. — Rio de Janeiro: PUC, Departamento de Informática, 2015.

v. 149 f. : il. ; 30 cm

Tese (Doutorado em Informática) – Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2015.

Inclui referências bibliográficas.

1. Informática - Teses. 2. Aplicações Multimídia. 3. Documentos Multimídia. 4. Vídeos 3D. 5. Estereoscopia. 6. Vídeo com profundidade. I. Soares, Luiz Fernando Gomes. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

To my parents, Tarcísio and
Carminda, and my advisor,
Luiz Fernando.

Acknowledgments

First of all, I am very grateful to my advisor, Luiz Fernando Gomes Soares, for his wisdom, encouragement, friendship, and advice. He helped to shape me in so many ways, not only as a researcher but also as a human being.

Also, I would like to give a special thank you to the members of my reading committee, Alberto Raposo, Sérgio Colcher, Simone Barbosa, Guido Lemos, and Rudinei Goulart, who have found time to read this work carefully and proposed much valuable and interesting input.

Over the last years, I have been part of the TeleMídia family. I warmly thank all my colleagues from the lab., who have helped to build a very fruitful environment for work, friendship, and all sorts of discussions.

I also gratefully thank my entire family who has been very comprehensive about my absence during the development of this work. Especially, I thank my wife, Vanessa Leite, who have always supported me with all my decisions, encouraged me to pursue my dreams, and helped me with all sort of emotional issues.

Finally, I thank CAPES and FAPERJ for their financial support.

Abstract

Azevedo, Roberto Gerson de Albuquerque. Soares, Luiz Fernando Gomes (Advisor). **Supporting Multimedia Applications in Stereoscopic and Depth-based 3D Video Systems.** Rio de Janeiro, 2015. 149p. PhD. Thesis - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Two-dimensional video technologies have evolved quickly in the last few years. Even so, they do not achieve a realistic and immersive view of the world since they do not offer important depth cues to the human vision system. Three-dimensional video (3DV) technologies try to fulfill this gap through video representations that enable 3D displays to provide those additional depth cues. Although CSV (Conventional Stereoscopic Video) has been the most widely-used 3DV representation, other 3DV representations have emerged during the last years. Examples of those representations include MVV (Multi-view video), 2D+Z (2D plus depth), MVD (Multi-view plus depth), and LDV (Layered-depth Video). Although end-to-end 3DV delivery chains based on those 3DV formats have been studied, the integration of interactive multimedia applications into those 3DV delivery chains has not yet been explored enough. The integration of multimedia applications with 3D media using those new representations has the potential of allowing new rich content, user experiences and business models. In this thesis, two approaches for the integration of multimedia applications into 3DV end-to-end delivery chains are proposed. First, a backward-compatible approach for integrating CSV-based media into 2D-only multimedia languages is discussed. In this proposal, it is possible to add depth information to 2D-only media objects. The proposal consists of extensions to multimedia languages and a process for converting the original multimedia application into its stereoscopic version. It does not require any change on the language player and is ready-to-run in current CSV-based 3DV delivery chains and digital receiver's hardware. Second, extensions to multimedia languages based on layered-depth media are proposed and a software architecture for the graphics composition of multimedia applications using those extensions is presented. As an example, both proposals are implemented and integrated into an end-to-end 3DV delivery chain based on the Brazilian Digital TV System.

Keywords

Multimedia Applications; Multimedia Documents; 3D Video; Stereoscopy; Video-plus-depth; Layered-depth-video;

Resumo

Azevedo, Roberto Gerson de Albuquerque. Soares, Luiz Fernando Gomes (Orientador). **Suporte a Aplicações Multimídia em Sistemas de Vídeo 3D Estereoscópicos e Baseados em Profundidade.** Rio de Janeiro, 2015. 149p. Tese de Doutorado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Tecnologias de vídeos bidimensionais (2D) têm evoluído rapidamente nos últimos anos. Apesar disso, elas não permitem uma visão realista e imersiva do mundo, pois não oferecem importantes dicas de profundidade para o sistema visual humano. Tecnologias de vídeo tridimensionais (3D) têm como objetivo preencher essa lacuna, provendo representações que permitem a reprodução de informações de profundidade em displays 3D. Embora a representação baseada em vídeos estereoscópicos ainda seja a mais utilizada até o momento, novas representações de vídeo 3D têm emergido, tais como MVV (Multi-view video), 2D+Z (2D plus depth), MVD (Multi-view plus depth) e LDV (Layered-depth video). A integração de aplicações multimídia com mídias 3D tem o potencial de permitir novos conteúdos interativos, novas experiências com o usuário e novos modelos de negócio. Nesta tese, duas abordagens para a integração de aplicações multimídia em cadeias de transmissão de vídeo 3D fim-a-fim são propostas. Primeiro, uma abordagem que é compatível com cadeias de transmissão de vídeo 3D baseado em vídeos estereoscópicos é discutida. A proposta consiste em extensões para linguagens multimídia 2D e um processo de conversão de aplicações multimídia 2D para sua versão estereoscópica. Essa proposta não requer nenhuma alteração no exibidor de linguagens multimídia 2D para a apresentação de mídias estereoscópicas. Em uma segunda abordagem, extensões adicionais a linguagens multimídia também são propostas visando a integração de aplicações multimídia em cadeias de vídeo 3D baseado em profundidade (2D+Z ou LDV). Além disso, uma arquitetura para a composição gráfica dessas aplicações, baseada no conceito de LDV e que permite a integração de objetos de mídia baseado em profundidade em exibidores de aplicações multimídias é apresentada. Como um exemplo de aplicação prática das proposta desta tese, ambas são implementadas e integradas em um sistema de vídeo 3D fim-a-fim baseado no Sistema Brasileiro de TV Digital.

Palavras-chave

Aplicações Multimídias; Documentos Multimídia; Estereoscopia; Video com profundidade; Vídeos em camadas com profundidade;

Table of Contents

List of Figures

Abbreviations and Acronyms

2D	Two-dimensional
2D+Z	2D plus depth
3D	Three-dimensional
3DTV	Three-dimensional Television
3DV	Three-dimensional Video
AFX	Animation Framework eXtension
API	Application Programming Interface
AR	Augmented Reality
ATTEST	Advanced Three-dimensional Television System Technologies
BIFS	BInary Format for Scenes
CSS	Cascading Style Sheets
CSV	Conventional Stereo Video
DIBR	Depth-Image-Based Rendering
DMB-T	Digital Multimedia Broadcasting—Terrestrial
DOM	Document Object Model
DTV	Digital Television
EPG	Electronic Programming Guide
FBO	Framebuffer Object
FPGA	Field-Programmable Gate Array
fps	frames per second
FTV	Free-viewpoint Television
Full-HD	Full-High Definition
GPGPU	General-purpose Computing on Graphics Processing Units
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HD	High-Definition
HDR	High Dynamic Range
HTML	HyperText Markup Language
HVS	Human Visual System

IBR	Image-based Rendering
iDTV	Interactive Digital Television
IPTV	Internet Protocol Television
IR	Infrared
ISDB-T	Integrated Services Digital Broadcasting—Terrestrial
ITU	International Telecommunication Union
ITU-T	ITU Telecommunication Standardization Sector
LC	Liquid Crystal
LCD	Liquid Crystal Display
LDI	Layered-depth Image
LDV	Layered-depth Video
MPEG	Motion Picture Expert Group
MSR	Mixed Scene Resolution
MVD	Multi-view plus depth
MVV	Multi-view video
NCL	Nested Context Language
NCLSC	NCL Stereo Converter
QoE	Quality of Experience
S3D	Stereoscopic 3D
SbS	Side-by-side
SBTVD	Sistema Brasileiro de TV Digital (Brazilian Digital TV System)
SDK	Software Development Kit
SMIL	Synchronized Multimedia Integration Language
SVG	Scalable Vector Graphics
TaB	Top-and-bottom
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VR	Virtual Reality
VRML	Virtual Reality Markup Language
X3D	eXtensible 3D
XML	eXtensible Markup Language
XSL	eXtensible Stylesheet Language
XSLT	XSL Transformations

Mathematical Notation and Symbols

f	Focal length
$\boldsymbol{m}$	Original camera pixels coordinate
$\boldsymbol{m_w}$	3D world coordinate
$\boldsymbol{m_v}$	Virtual camera pixels coordinate
P	Screen parallax
V	Distance of the observer to the screen
E	Interocular or interpupillary distance
Z_i	Observed depth of a point projected (with parallax P) in the screen
Z_m	Offset term to depth perception (with must be summed to V)
Z_{near}	Minimum allowed depth value
Z_{far}	Maximum allowed depth value
$\delta(x)$	Screen parallax value of a media object x

1
Introduction

1.1.
Motivating Scenario

Two-dimensional (2D) video technologies have advanced to increase spatial resolution and color quality. Nevertheless, they do not achieve a realistic and immersive view of the world since they do not offer important depth cues to the human vision system (HVS). Three-dimensional video (3DV) technologies try to fulfill this gap through video representations that enable 3D displays to provide those additional depth cues. 3DV and 3D display technologies have a huge range of applications, which include 3D Cinema, 3D Television (3DTV), Free-viewpoint Television (FTV) (TANIMOTO, 2012), and Telepresence (FUCHS; STATE; BAZIN, 2014).

To date, the most common and more developed approach for 3DV representation is the so-called conventional stereoscopic video (CSV). CSV-based systems are those that use a scene representation composed of two slightly different views (a *stereo-pair*) to be independently presented for each eye. Whereas CSV-based technologies have been the most used approach for 3DV, they have their own well-known problems. For instance, CSV-based technologies require the use of glasses (with the aim of filtering each eye's view), they lack depth cues based on movement (motion parallax), and they are hard to adapt to different viewing conditions (FEHN, 2004).

To solve some of the aforementioned issues, and to support new 3D media applications, other 3DV representations have been emerging. *Video plus Depth (2D+Z)* (FEHN, 2003b), *Multi-view Video (MVV)* (HO; OH, 2007), *Multi-view Video plus Depth (MVD)* (CHEN, YING *et al.*, 2014), and *Layered-depth-video (LDV)* (SHADE *et al.*, 1998) are among the most important ones. Figure 1 shows examples of those representations. 2D+Z format represents a unique view (conventional 2D video) of the scene with additional depth information. The depth information is commonly represented as a gray-scale image, in which 0 (or

black) means far from the viewer (inside the display), whereas 255 (or white) means close to the viewer. MVV format represents multiple viewpoints of the scene (not only a stereo-pair as in CSV). MVD is an extension of MVV, in which every viewpoint is associated with depth information. Finally, LDV is an extension of the 2D+Z in which besides the 2D view and depth information, background information is also represented. In particular, 3DV formats based on explicit depth information (2D+Z, MVD, and LDV) allow for detaching the 3DV format from the display and viewing conditions. In addition, they allow for transmission bandwidth optimizations and for generating additional views at the client side. A common approach for generating those additional views is using depth-image-based rendering (DIBR) (FEHN, 2003a).

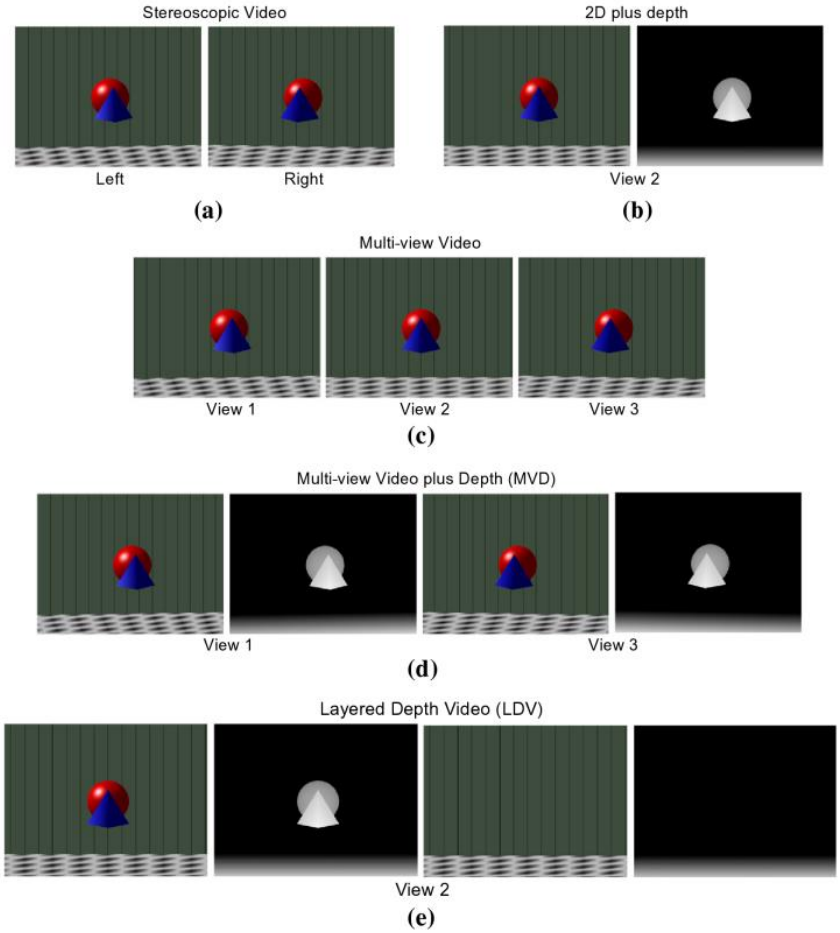


Figure 1 Examples of 3DV representations: (a) Conventional stereo video (CSV); (b) 2D plus depth (2D+Z); (c) Multi-view video (MVV); (d Multi-view video plus depth (MVD); and (e) Layered Depth Video (LDV). Adapted from (PICKERING, 2014).

End-to-end 3DV delivery chains based on 3DV formats have also been studied during the last years. CSV-based delivery chains are the most developed ones. Some approaches have already been standardized by the major digital TV (DTV) standards (ATSC, 2014; DVB, 2015), and some commercial trials have been performed. Concerning depth-based formats, two European projects deserve mention: ATTEST (REDERT *et al.*, 2002) and 3D4YOU (BARTCZAK *et al.*, 2011). ATTEST ("Advanced Three-dimensional Television System Technologies") proposes an end-to-end broadcasted 3DTV system based on the 2D+Z format. On the other hand, 3D4YOU project has studied the usage of the LDV format. Those (and similar) projects have achieved important contributions, showing the viability of using the different 3DV formats in end-to-end delivery chains, and discussing their advantages and disadvantages.

The fast development of the new 3DV representations mentioned above in all their phases (content creation, coding, transmission, and display) has enabled multimedia applications to take advantage of them. *Multimedia applications*—sometimes referred as richmedia (LIM *et al.*, 2012)(DUFOURD; AVARO; CONCOLATO, 2005)—are those digital applications composed of multiple (at least two) synchronized modalities—e.g., text, images, animations, video, audio (including speech), synthetic graphics (both 2D and 3D)—and that can support user interaction (Li, Drew, and Liu 2014). Examples of multimedia applications include those for the Web, Digital TV, and Interactive Cinema. Multimedia applications can be specified through using declarative or imperative programming languages.

Declarative languages are those in which authors are required to specify only the properties and the expected solutions. Examples of declarative languages include HTML ("HTML5", 2014, p. 5), SMIL (BULTERMAN *et al.*, 2008), SVG (MCCORMACK *et al.*, 2011), NCL (SOARES; LIMA, 2013), X3D (BRUTZMAN, 2007), and XMT (SIGNES; FISHER; ELEFTHERIADIS, 2000). *Imperative languages*, on the other hand, are those in which authors describe how to reach a solution as a sequence of steps. Examples of imperative languages include C, C++, Java, Javascript, and Lua. The textual specification of a multimedia application following declarative constructions is named a *multimedia document*. On the one hand, when compared to imperative languages, declarative languages provide a higher level of abstraction, which results in more reliable and easier to maintain software. On the other hand, declarative languages are usually domain

specific, which means they are not adequate for every task. As a consequence, declarative and imperative paradigms are commonly used together, with the latter being useful as a scripting language (for controlling complex behaviors) of the former.

The integration of the new 3D media representation formats with multimedia applications has the potential of allowing new rich content, new user experiences and new business models. In this scenario, it is possible to have multimedia applications using depth information provided by 3D displays not only as an additional technology, but also as a way to spatially structure user interfaces—e.g. occupying the space between the user and the display—providing genuine user interfaces (WOO; SUH; CHEON, 2012) (JUMISKO-PYYKKÖ; WEITZEL; STROHMEIER, 2008). It is possible to envision 3D display users not only watching videos in 3D, but also browsing the Web, participating in interactive games, consulting additional information on a TV program, and so on; all using real 3D depth information (CHISTYAKOV; GONZÁLEZ-ZÚÑIGA; CARRABINA, 2013).

Aware of those possibilities, some efforts have already followed the path of integrating interactive multimedia services into end-to-end 3DV delivery chains (LEE, BONGHO; YUN; LEE; *et al.*, 2009; LE FEUVRE, 2010). Some of them have been proposed to CSV-based end-to-end delivery chains. However, those systems usually require native support by the language player and they are not backward compatible with older 2D-only language players. Other works have included depth-based 3D media support into complete 3D multimedia scene environments. In such a case, the 3D media are included as objects in a full 3D world scene model. That approach is useful in many application scenarios. However, it cannot support real-time rendering of scenes containing high-quality (Full-HD or higher) 3DV-based services using current available graphics hardware.

A more recent approach is to rely on a 2D+Z multimedia scene representation and uses a DIBR approach for rendering the final views of the entire multimedia scene. In this case, it is the possible to support high-quality (full-HD or higher) 3DV media objects while allowing real-time performance. The main problem with this method is the introduction of annoying artifacts (such as holes and cracks)—which is an inherent issue of DIBR approaches—in the final generated views. The work

presented in this thesis tries to solve or minimize these problems, as stated in its objectives.

1.2. Objective

Motivated by the context above, the overall aim of this work is *to explore the integration of new 3D media formats into multimedia applications*. In broad terms, this work is interested in allowing for the composition of interactive multimedia applications containing both synthetic and natural (2D and 3D) media objects. In the scope of the thesis, synthetic media objects are those that are rendered at the client side, whereas natural media objects are those pre-rendered or captured from real world (e.g. ordinary video and images). In particular, we focus on the integration of the previously mentioned 3D media representation (CSV, 2D+Z, MVV, MVD, and LDV) into multimedia applications specified using declarative languages.

More specifically, the main goal of the work is to provide a framework, based on declarative multimedia languages, for easing the development of spatially structured multimedia applications in the context of 3DV end-to-end delivery chains. Those applications must be able to handle the new 3D media formats, and must allow a seamless integration of the different (2D and 3D) media objects with the main video in 3DV-based environments. Having the aim of reusing the already existing declarative languages for the specification of multimedia presentations, this work focuses on the integration of 3D media into current multimedia languages. When needed, extensions for those languages are proposed.

Interactive multimedia applications based on (3D) videos—e.g. those for interactive 3DTV and 3D Cinema—are the main use cases of the proposals herein.

The general research question addressed by this work is:

RQ.1. How to support the design and presentation (on 3D displays) of interactive multimedia applications containing (natural and synthetic) 2D/3D media objects in 3DV delivery chains?

However, this work is more specific when providing particular answers to the general question. The work focuses on two main scenarios.

The first one is based on a 3DV delivery chain using the CSV format. In this scenario, the aim is to extend the CSV-based 3DV with interactive multimedia content, addressing the following research questions:

RQ.2. How to extend current 2D-only declarative multimedia languages to allow the design of multimedia applications relating 2D and CSV-based 3D objects?

RQ.3. How to support the execution of multimedia applications, developed using the extended declarative languages of RQ.2, to be presented on 3D displays keeping unchanged the original language player?

In other words, RQ.3 is equivalent to answering: "*how to support the backward-compatibility of multimedia applications containing CSV-based media objects with current 2D-only multimedia language players?"*

The second scenario is a 3DV delivery chain based on 2D+Z or LDV formats. In that scenario, the following research questions are addressed:

RQ.4 How to extend current declarative multimedia languages to allow for the design of multimedia applications composed of 2D, 2D+Z, LDV, and 3D media objects?

RQ.5 How to support the real-time execution of multimedia applications addressed by RQ.4 to be presented on 3D displays?

In this second scenario, 2D, 2D+Z, LDV and 3D media objects can be both synthetic or natural.

1.3. Overview of the thesis proposals

To answer the questions related to the two mentioned scenarios, this work is divided into two main parts, namely *Stereoscopic Multimedia Applications,* and *Layered-depth-aware Multimedia Applications*.

1.3.1. Stereoscopic Multimedia Applications

The first part of the thesis focuses on answering RQ.2 and RQ.3. For that, the concept of *stereoscopic multimedia applications* is proposed. A stereoscopic multimedia application is one in which the left and right side views required by a stereoscopic 3D (S3D) display are explicitly codified. Standard input formats for S3D-based 3DTV sets are targeted as output, such as side-by-side and bottom-up formats. One of the main advantages of the proposed approach for supporting 3DV in multimedia application is that it is backward-compatible with current 2D-only language players.

The proposed approach takes as input 2D multimedia applications annotated with depth information, and converts them into stereoscopic multimedia applications, at the *application-level* (see Figure 2). What we mean by application-level is that the converter can translate (at client or server side) the original application to one that uses only the low-level 2D primitives supported by the original language player. The depth annotation in the input document allows for creating a stereo-pair as output. Therefore, individual 2D multimedia objects can float in or out of the screen when those applications are presented in S3D displays. Native CSV-based media objects, such as CSV-based videos, images, and dynamic generated animations (through an extended script API), can also be integrated and related with other media objects into the CSV-based multimedia scene.

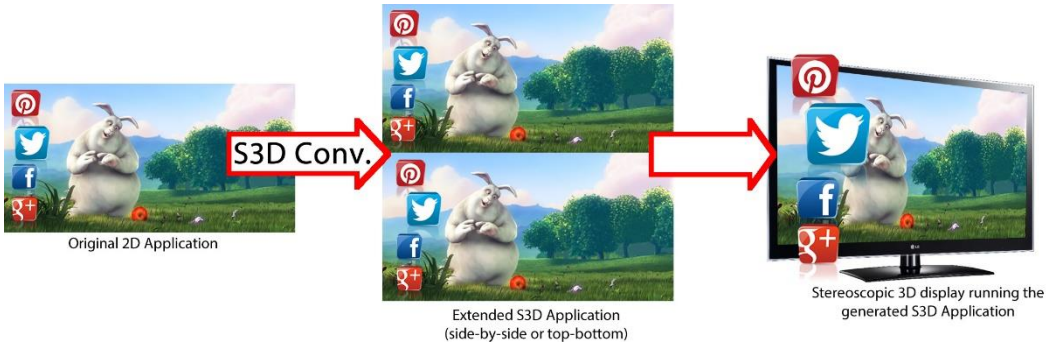


Figure 2 The proposed approach to support stereoscopic multimedia applications.

Issues related to the quality of experience (QoE) on stereoscopic-multimedia applications, such as asynchrony between the left and right views, and depth perception customization are handled in the proposed conversion process. The overall conversion process is proposed in an abstract way and can be performed both at the client side and at the server side, depending on the available transmission bandwidth and client receiver resources. In addition, an implementation of the proposed process for the NCL language is presented.

The approach employed in this first part allows for augmenting CSV-based delivery chain with rich interactive stereoscopic 3D services. It also provides a framework for creating applications which can spatially organize graphical elements using the real depth information, e.g., filling the space between the user and the display. The integration of stereoscopic multimedia applications in CSV-based 3DTV delivery chains and in 3D Cinema are discussed.

1.3.2. Layered-depth-aware Multimedia Applications

On the one hand, the approach employed in the first part of the thesis has the evident advantage of not requiring any change in the language player (since the proposed converter works at the application). On the other hand, it has the disadvantage of only supporting 2D and CSV-based media objects. For 2D media objects, it allows authors to associate only a uniform depth for each media object, which means that every pixel of the object will have the same depth.

Aiming at improving that issue, and focusing on RQ.4 and RQ.5, the second part of the thesis supports the integration of depth-based 3D media in multimedia

applications. Extensions to multimedia languages that allow for controlling the composition of the depth-based 3D media with other media objects in the application are proposed. The extensions also allow for adding depth and occlusion layers information to media objects that do not have those information natively. In order to achieve sufficient performance, the proposed approach needs to provide support at the language player level.

A graphics architecture supporting a seamless integration among 2D, 2D+Z, LDV, and 3D (synthetic) media is discussed (Figure 3). The graphics architecture of a multimedia platform is the part of the system software that handles the graphics composition among the different individual media objects and renders the system output on the screen (CESAR, 2005). In our case, the graphics architecture must support the rendering of multiple views, required by 3D displays. Aiming at achieving real-time performance, the proposed architecture is implemented using GPU (Graphics Processing Unit), through the OpenCL API (MUNSHI, 2012). The innovative concept of the proposed graphics architecture is that it is based on the concept of LDV format. When compared to other related works the proposed graphics architecture allows for better rendering results, while still keeping real-time performance.

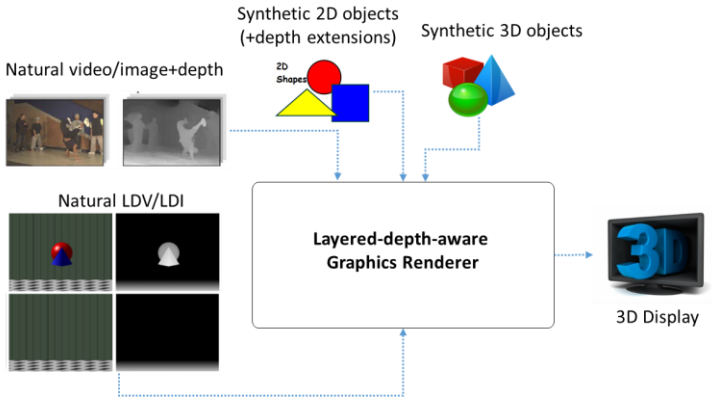


Figure 3 Abstract view of the Layered-depth-aware Graphics Renderer.

1.4. Contributions

In summary, the thesis aims to bring the following contributions:

- Extensions to 2D-only multimedia applications that simplify the development of interactive multimedia applications using CSV-based media for being presented on S3D displays;
- A framework for generating stereoscopic-multimedia applications, while keeping the language player unchanged;
- Extensions to multimedia languages that ease the development of interactive multimedia applications for 3DV systems based on 2D+Z or LDV format.
- A software architecture for the rendering of multimedia applications composed of 2D, 2D+Z, LDV, and 3D media objects (and that support the extensions above).

1.5. Outline

The remainder of the thesis is organized as follows.

Chapter 2 discusses some related works and compares them with the proposals herein.

Chapter 3 details the proposal for supporting stereoscopic multimedia applications at the application-level, and discusses QoE issues related to them.

Chapter 4 describes the proposal for integrating depth-based media objects into multimedia applications, and details the proposed graphics architecture and implementation that support such integration.

Chapter 5 concludes this work and presents some future work possibilities.

In addition, Appendix A discusses introductory concepts related to 3DV, such as how the human vision system is able to reach depth perception, and the currently available 3D displaying technologies. Mathematical background behind depth perception in S3D displays is also discussed. As an additional contribution of this thesis, Appendix B introduces some layout templates that ease application authoring using the proposals of Chapter 3.

2
Related Work

Low-level imperative graphics APIs, such as OpenGL (SHREINER *et al.*, 2013) and DirectX (GRAY, 2003), provide the necessary means for expert programmers to render multiple media objects, compose multimedia scenes, and generate the different views required by stereoscopic or multi-view 3D displays. Indeed, the graphics architecture proposed in Chapter 4 takes advantage of such lower-level APIs. Some higher-level imperative APIs allow for structuring 3D synthetic scenes through the use of the Scene Graph (CLARK, 1976) concept—e.g. OpenSceneGraph (OSFIELD; BURNS; OTHERS, 2004). Those APIs can be extended to support the generation of the multiple views, required by 3D displays (DANE; BHASKARAN, 2013). Although those APIs allow some abstractions over the lower-level OpenGL/DirectX, they are still imperative approaches and require advanced programming skills. Unlike those approaches, the main goal of this work is to support the design of multimedia applications through *declarative* multimedia languages. Minor imperative support is also part of the proposed approaches, but they are based on the extension of scripting languages that are attached to declarative multimedia languages. The related work discussion in this chapter is then limited to the scope of proposals supporting 3DV and 3D displays in declarative multimedia applications.

Section 2.1 presents some of the declarative languages currently in use for the development of multimedia presentations based on monoscopic 3D—i.e., traditional computer graphics that generates images based on a 3D coordinate system and presents them on 2D displays. Section 2.2 discusses previous proposals for extending multimedia languages with the aim of allowing them to be presented on 3D displays. Those proposals are relatively new, and they have been mainly driven by the renewed interest in S3D technologies and the affordable prices of currently available S3D displays. Finally, Section 2.3 discusses work on multimedia languages to be presented on multi-view displays.

2.1. Monoscopic 3D Multimedia Applications

Many efforts have aimed at bringing synthetic 3D objects to the Web. Technologies such as VRML (ISO, 1997), X3D (BRUTZMAN, 2007), and O3D ("O3D Project's page", [S.d.]) are examples of such efforts. At first, those technologies were mainly implemented in web browsers through the use of plug-ins APIs (ORTIZ JR., 2010). Recently, driven by the introduction of the WebGL API (KRONOS GROUP, 2013), it is also possible to find the implementation of players for those languages at the application-level. In the scope of the Web, that technique is usually called *Polyfill*, and examples of players following that approach include X3DOM (BEHR, JOHANNES *et al.*, 2009) (BEHR, J. *et al.*, 2010) (BEHR, JOHANNES *et al.*, 2011) and XML3D.js (SONS *et al.*, 2013).

In the scope of interactive digital television (iDTV), some proposals also follow this direction. Based on the NCL multimedia language—the Brazilian standard for interactive Digital TV—(AZEVEDO, 2010) embeds 3D objects (both simple, such as OBJ, and composite, such as VRML/X3D) into the language and extends it to define anchors and to control the behavior of 3D objects. (SOUZA *et al.*, 2010) also discusses the integration of X3D objects into the Brazilian Digital TV system and includes the support for imperative APIs based on Java/OpenGL ES. (CESAR; VUORIMAA; VIERINEN, 2006) presents a software graphics architecture for high-end interactive television terminals and discusses new interactive scenarios that can emerge when native rendering of 3D objects is supported in iDTV middlewares.

As argued by (CHEN, QINSHUI *et al.*, 2014), the above technologies can be broadly classified as monoscopic 3D, since they are: "traditional computer graphics that generates images based on a 3D coordinate system, and then presents those images on 2D display". The integration of 2D graphical elements which are part of the original declarative document (HTML or NCL) with embedded 3D scenes remains a research issue in the literature (JANKOWSKI *et al.*, 2013; LE FEUVRE, 2012). Other requirements for better integration of Web applications and 3D documents are presented in (LE FEUVRE, 2012). The support for 3D displays is among the requirements highlighted by Le Feuvre.

In the scope of this thesis, we are interested in multimedia applications based on a main 3DV, such as 3DTV and 3D Cinema applications. Furthermore, we are interested in using natural 3D media (CSV, 2D+Z or LDV) in the multimedia scene, which is not completely supported in current multimedia languages. Besides the support for depth-based 3D media, we are also interested in providing abstractions to multimedia authors to allow them to control the depth information of every graphical element in a multimedia application, even those that do not have native depth information.

Unlike VR environments, in which a complete 3D environment is necessary, the kind of applications we are interested in do not necessarily use stereoscopic effects to improve the user's immersion in a virtual environment. The effects can also be used to provide additional interactive effects, e.g. highlighting active elements in the user interface or spatially positioning the user interface.

2.2. Multimedia Applications and Stereoscopic 3D Media

Since the beginning of the Web, there have been many examples of websites specifically developed to reach stereoscopic effects, mainly through anaglyph techniques ("Awwwards Team: 10 stereoscopic 3d websites", 2011). More recently, driven by the growing availability at affordable prices of 3D displays, it is now possible to see some efforts that, as in this thesis, aim at extending current multimedia languages to allow their presentation on 3D displays. Since most of the 3D displays available on the market are stereoscopic-based, most of those proposals are based on stereoscopic-only content.

CSS "Extensions for Stereoscopic 3D support" (HANG; LEE, 2012) is a draft proposal that has been developed by W3C. This specification proposes a set of CSS properties that can be associated to graphical elements in HTML, which state how they should be presented on a stereoscopic 3D display.

(WANG *et al.*, 2014) also discusses similar extensions to the support of stereoscopic applications in HTML. Besides the CSS properties, Wang et al. propose a new HTML element (<*frame*>) and a new JavaScript object ('*format'*). The extensions proposed by Wang et al. are implemented in the Webkit (APPLE, 2015) rendering motor. Other publications of the same group that detail their

proposal include (ZHANG, JIANLONG *et al.*, 2014) (CHEN, QINSHUI; WANG; WANG, 2014) (CHEN, QINSHUI *et al.*, 2014).

(LIU; WANG; WANG, 2014) extends the Webkit rendering motor in order to allow SVG documents to be presented on S3D displays. Liu *et al.* propose two attributes to the SVG language: *thickness* and *depth*. Those attributes allow authors to control how SVG's graphical elements are disposed "inside" or "outside" the screen.

(LEE, HYUN *et al.*, 2007) (LEE, BONGHO; YUN; HUR; *et al.*, 2009) (LEE, BONGHO; YUN; LEE; *et al.*, 2009) and (JUNG *et al.*, 2008) allow the integration of interactive content in a 3DTV services based on video plus depth, in the context of T-DMB (*Terrestrial–Digital Multimedia Broadcasting*) (SABIRIN *et al.*, 2012). As the main transmitted video is based on video plus depth, the Lee et al.'s system could theoretically support not only stereoscopic-only 3D displays, but also multi-view displays. However, the interactive content itself is based on stereoscopic image pairs. A "homemade" XML document controls when additional media objects must start or stop. The lack of explicit depth information also incurs other drawbacks, such as the lack of support to movement-parallax and the "impossibility" of adapting the multimedia presentation to different viewing conditions. Moreover, Lee *et al.* and Jung *et al.* define a new multimedia language, which has only very simple features when compared to the current standard multimedia languages. In this thesis, we focus on extending the currently available multimedia language standards, and reuse their powerful behavior specification.

The aforementioned proposals are intended to be natively supported by the language player. Different from them, the first part of this thesis aims at supporting the stereoscopic effects at the application-level, which means we do not need any native support, but we can take advantage of this support when it exists.

Among the related work, the proposals of Zhang et al. (ZHANG, SHAOBO; ZHOU; SUN, 2012) and Chistyakov et al. (CHISTYAKOV; GONZÁLEZ-ZÚÑIGA; CARRABINA, 2013) are those closest related to the first part of this thesis. Both proposals implement a JavaScript library that generates stereoscopic webpages. Zhang et al. rely on the HTML5 Canvas API (CABANIER *et al.*, 2015, p. 3) to generate the left and right views of the webpage. Chistyakov et al. use the DOM API (HÉGARET *et al.*, 2004) to duplicate the HTML elements and produce the final stereoscopic application. Similarly, the first part of this thesis also

produces a new declarative stereoscopic version of the application. However, using the declarative constructions of the multimedia language, our proposed process can ensure a frame-based synchronization between the two instances (left and right) of dynamic media objects (such as videos and script-based animations). The synchronization problems of left and right view have not been handled or discussed by the approach of Chistyakov et al.. Moreover, other differentials of the first part of this thesis include: the adaptation of the output application to processing parameters (e.g., display size, viewer distance, and parallax scale factor); the support for the development of script-based animations; and the integration of CSV-based and 2D-only media objects with depth extensions.

2.3. Multimedia Applications and Depth-based 3D Media

Besides the aforementioned work that focus only on binocular 3D displays, there have been also some efforts considering multi-view display.

Tsai (TSAI; YOUNG; KU, 2012) presents a Flash[1] player that supports depth map configurations. Tsai's proposal allows the conversion of 2D multimedia scenes into 3D scenes based on DIBR. The author of the multimedia scene is able to control a per-pixel depth map for the entire scene. The proposals of this thesis, on the other hand, allow multimedia authors to control the depth information for each media object individually. The multimedia language player is responsible for the graphics composition of the entire scene based on the information of the each object.

An example of the integration of webpages with multi-view displays is presented by Nocent et al. (NOCENT *et al.*, 2012). Their work, however, does not integrate any functionality to the declarative multimedia language (HTML, in this case). It uses the WebGL (KRONOS GROUP, 2013) imperative API to generate one or more views that can be used as inputs for 3D displays. The depth effect is only perceived inside the WebGL canvas. Therefore, the integration with other elements in the webpage is not supported. In contrast, the approaches proposed in this thesis allow multimedia application authors to specify and dynamically control the depth information of any media object that is part of the application.

[1] Available at http://www.adobe.com/products/flash.html

BIFs AFX (Animation Framework eXtension) provides depth-nodes (LEVKOVICH-MASLYUK *et al.*, 2004) that can be integrated into a 3D BIFs scene. The AFX approach is interesting to support the integration of depth-based media into a complete 3D environment, such as VR environments. However, as that proposal integrates the depth-objects into a traditional polygon-based rendering pipeline, they cannot take full advantage of the performance allowed by image-based rendering (MCMILLAN JR., 1997). An image-based rendering approach, as the one used in the second part of this thesis, has the potential to provide real-time performance even for high quality 3DV (e.g. Full HD or higher).

Another work, mainly related to the second part of the thesis, is the one developed by Jean Le Feuvre (LE FEUVRE, 2010)(LE FEUVRE; MATHIEU, 2013). Depth-based extensions to the SVG language allow its execution on multi-view displays as suggested in (LE FEUVRE, 2010), whereas a graphics architecture for supporting those applications is presented in (LE FEUVRE; MATHIEU, 2013). Two solutions are presented in (LE FEUVRE; MATHIEU, 2013): one based on GPU, mainly developed for testing purposes; and another one based on FPGA (Field Programming Gate Array) to be included in low-cost platforms. Both solutions are integrated with the GPAC (LE FEUVRE *et al.*, 2011; LE FEUVRE; CONCOLATO; MOISSINAC, 2007) multimedia framework. The second solution is the one closest to this work: it produces a 2D+Z representation of the entire scene and then sends it to FPGA. The FPGA is then responsible for generating the views required by 3D displays using DIBR.

In comparison with Le Feuvre's work, the extensions to multimedia languages and the graphics composition solution presented in the second part of this thesis are based on the LDI/LDV concept, which allow us to reach better rendering results. That is mainly explained by the fact that our proposal keeps as much occlusion information as possible while graphically compositing the different media objects. We also support the proposed extensions in script-based canvas-like APIs, commonly integrated with the main declarative multimedia language. In addition, besides the native support for depth extensions (the second part of the thesis), we also discusses the application-level support for stereoscopic multimedia presentations (the first part of the thesis).

Compared to all the aforementioned related work, the work on this thesis is the only one that provides a multimedia scene model based on LDI and to support the rendering of multiple views for 3D displays based on such concept.

3
Stereoscopic Multimedia Applications

This chapter proposes a set of extensions to multimedia languages and a converter aiming at the development of stereoscopic multimedia application. A stereoscopic multimedia application is a multimedia application that codifies both the left and right views of a multimedia application following one of the frame-compatible approaches for CSV (side-by-side, top-bottom, etc.). As a result, the generated stereoscopic multimedia presentation is ready-to-run on current S3D displays (see Figure 2).

More precisely, the proposal encompasses: (i) extensions to multimedia documents that allow authors to add CSV-based media objects and control how 2D-only media object can be presented stereoscopically; and (ii) a conversion process that generates stereoscopic multimedia applications based on input documents using the proposed extensions.

As an example of the conversion process, an implementation for the NCL multimedia language is presented.

Unlike mechanisms for converting 2D-to-S3D video—a built-in feature of most 3DTV sets available today, and an active area of research in Computer Vision research community—the proposal in this chapter does not intend to convert elementary media (e.g. video or image) content to S3D. Instead, the focus here is on converting 2D interactive multimedia presentations to their stereoscopic versions in the multimedia document level. Automatic 2D-to-S3D conversion mechanisms for video-only are typically based on fallible heuristics to infer the depth information. In contrast, the conversion process proposed in this chapter uses depth information supplied by document authors. As a consequence, it does not suffer from those issues.

The remainder of the chapter is organized as follows. Section 3.1 details the motivating scenario (which was briefly introduced in Chapter 1). Based on this scenario, Section 3.2 discusses the main requirements the proposals in this chapter must support. Sections 3.3 to 3.7 detail the proposed solution. Section 3.8 discusses

some experimental results. Finally, Section 3.9 is reserved for the analysis of the complexity and a discussion on issues related to the quality of experience (QoE) of the generated stereoscopic applications.

3.1. Motivating Scenario

The motivating scenario for the proposals in this chapter is an interactive 3DV-based distribution chain based on the conventional stereoscopic representation for 3DV—which, as aforementioned, is the most commonly used approach by current commercial systems. 3DV representation using CSV formats is also a key factor to guarantee compatibility with existing 2D video networking technology and equipment (KONDOZ; DAGIUKLAS, 2014). Figure 4 depicts an example of such a distribution chain in broadcasted terrestrial 3DTV services.

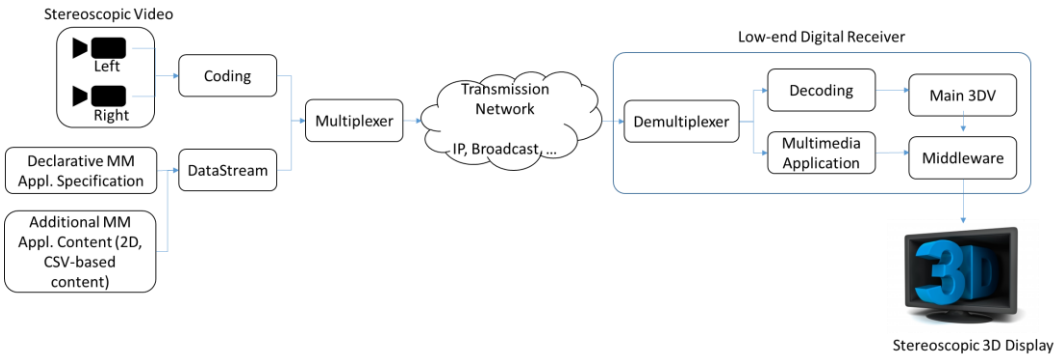


Figure 4 Example of an interactive terrestrial 3DTV delivery chain based on CSV.

In this scenario, the multimedia application (and its individual media components) is multiplexed together with the main audio and video streams into a transport stream (TS). The resulted bitstream is then modulated (in the case of a terrestrial digital TV system) or encapsulated and transmitted into IP packets (in the case of an IPTV system). The client is responsible for receiving the multiplexed bitstream, de-multiplexing the main audio/video and the multimedia application, and delivering them to the components responsible for decoding each of those data. The language player (usually part of a digital TV middleware) is the component responsible for interpreting the multimedia document and for creating a multimedia

presentation accordingly. The main audio and video can also be part of (and be controlled by) the multimedia application.

Hybrid delivery approaches are equally possible in the scenario. For instance, the main 3DV can be broadcasted while the multimedia application can be accessed through another transport network (e.g. Internet). It is also possible to have the basic 2D content (video-only or video+data) broadcasted and to have the augmented additional layer (e.g. the right-view of a stereoscopic video) coming from another transport network.

A *low-end* digital receiver is considered in this scenario. The kinds of low-end digital receiver that are part of the scenario are those that Pablo *et al.* (CESAR; VUORIMAA; VIERINEN, 2006) have defined as *Interactive Basic* (in the case it does not have a high speed return channel) or *Interactive Internet Access* (in which case it has a faster return channel). On the one hand, those digital receivers support declarative 2D multimedia language environments such as those provided by HTML, SMIL, or NCL. On the other hand, they do not necessarily have enough graphical power to render 3D objects natively—i.e. they lack a powerful GPU (Graphical Processing Unit) that supports OpenGL or DirectX 3D-like APIs.

3.2. Requirements Analysis

Even with a lot of efforts aiming at supporting synthetic (client-side rendered) 3D objects into iDTV middlewares (AZEVEDO, 2010; CESAR; VUORIMAA; VIERINEN, 2006; SOUZA *et al.*, 2010) none of the currently in use iDTV middlewares for terrestrial digital TV has such native support. That situation can be mainly explained due to cost and hardware restrictions. Indeed, we have now a situation in which there are many TV sets equipped with 3D displays, but those TV sets are not able to natively render synthetic 3D objects. Since we want to allow the development of stereoscopic multimedia presentation to execute in current CSV-based delivery chains, one important requirement of our proposal is:

Req. 1. The solution must be backward-compatible with 2D-only language players.

Even if it is possible to render 3D objects natively by the language player, rendering high-quality photorealistic 3D media content at interactive rates is still a challenge today. Videos and images, however, are photorealistic and simple to handle at the client side. Moreover, CSV-based video and images also support photorealistic stereoscopic content. Therefore, another requirement is:

Req. 2. The solution must support the use of CSV-based media objects, which can provide photorealistic stereoscopic content, in composing multimedia presentations.

Current available S3D displays support the presentation of the main 3DV codified in one of the frame-compatible approaches for CSV. Thus, another requirement is:

Req. 3. The solution must produce CSV-based multimedia presentations that can be displayed in current available S3D displays.

Whereas the backward-compatibility requirement does not allow for supporting the full-range of functionality provided by natively supported 3D APIs, some interesting stereoscopic effects should be allowed in 2D-only media objects. For instance, it must be possible to spatially organize 2D-only content in a layered-like user interface (see Figure 5). Thus, still another requirement is:

Req. 4. The solution must allow multimedia authors to associate depth information to 2D-only media objects.

Figure 5 Example of a 2.5D user interface. Source: SeeSpace (http://inair.tv).

Most multimedia languages embed a scripting language to extend its domain specific scope. In addition, some of them also provide canvas-like drawing APIs that allow authors to create complex animations at the client-side—that is the case of HTML5 Canvas (CABANIER *et al.*, 2015) and of NCLua (SOARES; MORENO; SANT'ANNA, 2009) APIs. The drawing results are treated in the multimedia application as a media component. Thus, another requirement is:

Req. 5. The solution must support CSV-based script (animations) media content.

Different viewing conditions induce different depth perceptions (see Section A.4 for details). For instance, stereoscopic content optimized for a standard 30-foot cinema screen will look completely different on a TV or on a handheld 3D display. Hence, controlling and adapting depth perception to the viewing conditions is of central importance to the widespread adoption of S3D (SUN; HOLLIMAN, 2009). Depth perception and side effects of stereoscopic displays also vary vastly among individual users. Individual viewers may also have different viewing preferences (LANG *et al.*, 2010). Therefore, it is important to allow users to adapt the depth information based on their own preferences, leading to a new requirement:

Req. 6. The solution must support adaptation to different viewing conditions and user preferences.

As previously shown in subjective experiments—e.g. (GOLDMANN; LEE; EBRAHIMI, 2010)—it is crucial that both the left and right views of a stereoscopic 3D application are always synchronized. Even small synchronization skews can deteriorate the quality of the resulting stereoscopic presentation, annoying users, and eventually making the presentation impractical (GOLDMANN; LEE; EBRAHIMI, 2010). Finally, another important requirement is:

Req. 7. The solution must guarantee a fine synchronization between the left and right views of the resulting stereoscopic multimedia presentation.

With the aim of easing the process of creating stereoscopic multimedia presentations and supporting the requirements of this section, the remainder of this chapter discusses a set of proposals that can be integrated into current 2D-only declarative multimedia languages.

3.3. Overview of the Proposed Solution

This thesis focuses on declarative multimedia applications. However, as previously mentioned some scripting supporting is also envisioned. In the scope of the thesis, scripting languages may be embedded into the declarative multimedia language to allow features that are not in the scope of the declarative multimedia model. Therefore, providing abstractions to create CSV-based media content through scripting languages is also in the scope of the proposals in this chapter.

Although the proposals can be used together (as it will become clear during the remainder of the text), for the following discussion they are divided into declarative and imperative support.

The declarative support is achieved by using:

- *extensions to 2D-only declarative multimedia languages* that allow authors to control the depth (and parallax[2]) of 2D-only media objects, and to integrate CSV-based media objects into the presentation. The discussion of those extensions is the subject of Section 3.4; and

[2] Parallax concepts and the geometry of depth perception on S3D displays are discussed in Section A.4.

- a *document converter* that handles the generation of stereoscopic multimedia documents. The document converter translates a document with the proposed extensions to a new document that uses only the primitives supported by the original language player. Section 3.5 discusses the document converter.

The imperative support is achieved by:

- *extensions to scripting languages* that allow authors to create CSV-based media content. Those extensions are the subject of Section 3.6; and
- a *library* that can be used by scripting authors to support the previous imperative extensions. That library is presented in Section 3.7.

3.4. Declarative Support for Stereoscopic Multimedia Presentations

Declarative support to develop stereoscopic multimedia presentations is achieved by properties that can be associated with media objects, or globally controlled in the multimedia application.

The following properties can be attached to 2D-only media objects to allow them to be rendered stereoscopically:

depth: The depth of the media object. This property is a double value in the interval $[-1.0,1.0]$. The depth value must be adapted to the final screen parallax based on the maximum and minimum allowed parallaxes of the scene. Negative values of depth mean inside of the screen (i.e. positive parallaxes), whereas positive values mean outside of the screen (i.e. negative parallaxes). This property allows for adapting the final parallaxes values to different viewing conditions, and should be preferred concerning the **parallax** property.

parallax: The hardcoded amount of screen parallax of a media object. The value of this property must be in pixels or percentages of the screen. Its value can be either positive (in which case the object will be presented inside the screen) or negative (in which case the object will be presented outside of the screen). The parallax property is only available with the aim of supporting cases in which the author

has enough information about the viewing environment, and wants to hardcode the value of the final parallax. The **depth** property, which allows the adaptation to different viewing conditions, should be used when possible to replace the **parallax** property.

The following properties may be associated with CSV-based media objects to inform how the content must be rendered:

stereoMode: specify that the media object uses one of the supported CSV-based formats. Possible values are '*none*', '*side-by-side*', and '*top-bottom*'. The default value is '*none'*.

stereoSwappedLR: specify whether the content in the frame is inverted, i.e., whether the right frame comes before the left frame. Possible values are '*true'* and '*false'*. The default value is '*false'*.

Whereas the properties above can be individually attached to each media object, some constructions in multimedia languages usually allow grouping media properties together (aiming at being reused by more than one media object). This is the case of CSS properties or NCL <descriptor> elements. Therefore, the properties above may also be associated through these constructions.

The following properties may be globally controlled to inform how the stereoscopic multimedia presentation must be rendered:

s3d.stereoMode: specifies how the final stereoscopic multimedia presentation must be rendered. Possible values are 'none', 'side-by-side', and 'top-bottom'. The default value is 'none'.

s3d.stereoSwapLR: specifies whether the outputted stereoscopic multimedia presentation must swap left and right views. Possible values are 'true' and 'false'. The default value is 'false'.

s3d.maxNegativeParallax: specifies, in pixels or percentage of the screen, the maximum allowed negative screen parallax. The default value is "3%"[3].

[3] This default value is based on the percentage rule (MENDIBURU, 2012), commonly used in cinematography, which states that for a comfortable stereoscopic viewing the *negative parallax* should not exceed 2%– 3% of the screen width.

s3d.maxPositiveParallax: specifies, in pixels or percentage of the screen, the maximum allowed positive screen parallax. The default value is "2%"[4].

s3d.resizeContent: specifies whether the media objects must be resized based on their depth. Possible values are 'true' and 'false'. The default value is 'false'.

s3d.minResizeFactor: if **s3d.resizeContent** property is 'true', this property is used as the minimum resize factor of a media object. This is equivalent to specify that elements with $depth = -1.0$ will be resized by a factor of **s3d.minResizeFactor**. The default value is 1.0.

s3d.maxResizeFactor: if **s3d.resizeContent** property is 'true', this property specifies the maximum resize factor of a media object. This is equivalent to specify that elements with $depth = 1.0$ will be resized by a factor of **s3d.maxResizeFactor**. The default value is '1.0'.

The following code block shows examples of how the extensions can be included in different multimedia languages.

[4] As for the positive parallax, this default value is also based on the rule commonly used in cinematography, which states that for comfortable stereoscopic viewing the *positive parallax* should not exceed 1%– 2% of the screen width.

```
...
<div style="depth: 0.5;">
  This is a text "floating out of the screen".
</div>
...
```

(a)

```
<smil ...>
  <head>
     ...
    <layout>
        <root-layout width="1920" height="1080"
                     s3d:stereoMode="side-by-side"
                     s3d:maxPositiveParallax="20%"
                     s3d:maxNegativeParallax="30%"/>
        <region xml:id="rgFloating" width="200" height="200
                depth="0.5"/>
        ...
    </layout>
  </head>
  <body>
    ...
    <text region="rgFloating" src="text.html" dur="10s" />
    ...
  </body>
</smil>
```

(b)

```
<ncl ...>
  <body>
    ...
    <media id="video1" src="text.html">
      <property name="width" value="50%"/>
      <property name="height" value="50%"/>
      <property name="depth" value="1.0"/>
    </media>
    ...
  </body>
</ncl>
```

(c)

Listing 1 Depth-property association in different multimedia languages: (a) HTML/CSS; (b) SMIL; (c) NCL.

3.5. Stereoscopic Multimedia Document Converter

The language player can natively implement the support to the extensions defined in Section 3.4. This is interesting, since it can provide various sorts of optimizations. However, to satisfy *Req. 1*, an additional approach is assumed. In it, a converter is responsible for translating the multimedia document, using the extensions proposed, to a stereoscopic version using only the original 2D language

primitives. The translation process can run at the client-side or at the server-side. In the latter, the server must convert the multimedia application to its stereoscopic version before sending it to the client.

The process to convert a multimedia document to its stereoscopic version, depicted in Figure 6, consists of three main steps: *media cloning*, *layout adjustment*, and *behavior cloning*.

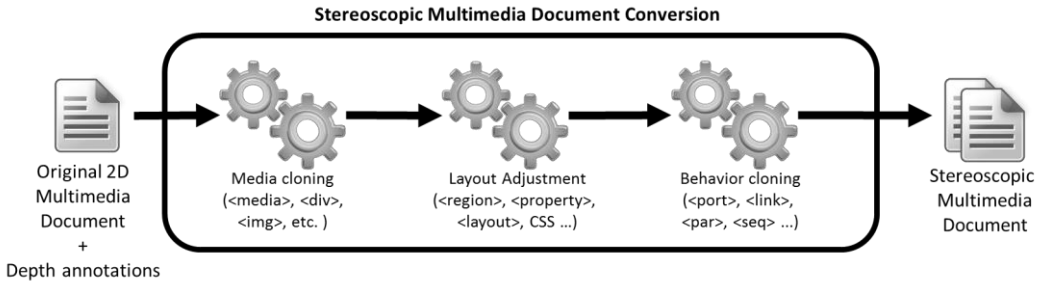


Figure 6 The three steps of the Stereoscopic Multimedia Document Converter.

The *media cloning* step focuses on *what* will be presented. In this phase, the processor duplicates 2D media objects that compose the original multimedia document and that have some graphical representation. Each new duplicated object will represent the left and right views of the original object in the resulting stereoscopic application. In addition, this step must also handle objects that have a native CSV-based codification and that can be part of the stereoscopic multimedia presentation.

The *layout adjustment* step determines, statically, *where* and *how* the media objects duplicated in the previous phase will be initially presented. In defining the initial presentation position of media objects, the processor can introduce parallaxes between the object's position in the left and right views. Such a parallax is what induces binocular disparity and allows the brain to interpret the 3D depth information during a stereoscopic multimedia presentation (see Section A.2.1). Multimedia authors can provide the *parallax* values or the high-level *depth* property for each media object, by using the extensions presented in Section 3.4.

The *behavior cloning* step focuses on the dynamic behavior of the multimedia presentation, and includes *when* the media objects appear, disappear, and when their properties change during presentation. In this step, the processor must correctly

duplicate the behavior of the original application into an equivalent behavior affecting both the left and right views of the stereoscopic application. Moreover, the processor must guarantee that dynamic (run-time) changes on any view are replicated on the other counterpart.

Next subsections detail each of those steps. As an example of the conversion process, a tool named NCLSC (NCL Stereo Converter) was implemented to convert NCL applications for S3D presentations. The details of the conversion process are discussed focused on the NCL language, but it can be extended to other multimedia document models as well. A brief discussion on how the conversion can be adapted to SMIL and HTML languages is also presented.

3.5.1. Media Cloning

The media cloning step must guarantee that each graphical element in the original multimedia document produces two representations in the final stereoscopic document: one for the left view and the other for the right view. In addition, media objects that are natively codified as one of the frame-compatible CSV-based approaches must be supported. Their left frame must be on the left side of the stereoscopic multimedia; and their right frame on the right side.

In NCL (SOARES; LIMA, 2013), media objects are defined via <media> elements. Thus, in the media cloning phase, the converter identifies and duplicates every <media> element of the input document that has a graphical representation (e.g. images, videos, animations etc.). Whenever duplicating a <media> element, the converter must also duplicate its child elements (<property> and <area> elements). To maintain the resulting document valid, the `id` attribute of each cloned <media> element is updated to be unique in the document.

In NCL, a <media id="A"> element refers to its media content via its *src* attribute, which defines the content URI (Uniform Resource Identifier). If the URI scheme is equal to "ncl-mirror" and the specific part of the scheme refers to an identifier of another media object, both objects will have the same content, that is, they present exactly the same content sample, independently of their starting time (SOARES NETO, CARLOS DE SALLES; SOARES; SOUZA, 2010). Since NCL players guarantee this behavior, NCLSC uses this feature to solve the problem

of synchronization skew between the left and right views of the same content. In other words, to guarantee a frame-based synchronization, for each pair of duplicated elements (left and right instances) one element uses the mirror source mechanism to refer to the other one, which keeps the same value defined in the original element in its *src* attribute. Notice, however, that, since they are different media objects, their properties may have different values.

To illustrate the process, consider the following NCL code chunk:

```
<media id="video" src="file:///video.mp4">
  <property name="explicitDur" value="50s"/>
  <area id="a1" begin="10s" end="20s"/>
</media>
```

After the media cloning step, the <media> element is replaced by the following pair of <media> elements:

```
<media id="video-L" src="file:///video.mp4">
  <property name="explicitDur" value="50s"/>
  <area id="a1" begin="10s" end="20s"/>
</media>

<media id="video-R" src="ncl-mirror://video-L">
  <property name="explicitDur" value="50s"/>
  <area id="a1" begin="10s" end="20s"/>
</media>
```

In the listing, each new media object represents a view—left, in case of `video-L`, and right, in case of `video-R`—of the original element in the resulting stereoscopic document.

Supporting CSV-based media objects

In addition to 2D-only media objects (annotated with depth information) some media objects codified as CSV-based formats—e.g. S3D videos, images, or script-based animations—may be part of the multimedia application. An example of such object in an NCL document is shown in what follows.

```
<media id="cube" src="cube.mp4">
  <property name="stereo-content" value="side-by-side"/>
  ...
</media>
```

In case of CSV-based objects, the media cloning phase must also clone the media object, but it should guarantee that each new media object will extract from the original element only the associated view. This is done through the element

<area> with the *coords* attribute. <area> elements allows for defining spatial and temporal segments of content, named anchors in NCL. In the previous case, the output of the media cloning step is:

```
<media id="cube-L" src="cube.mp4">
  <area name="cube-L-side" coords="0,0,50%,100%"/>
  ...
</media>

<media id="cube-R" src="ncl-mirror://cube-L">
  <area name="cube-R-side" coords="50%,0,50%,100%"/>
  ...
</media>
```

The order of values in the *coords* attribute is “left-x”, “top-y”, “right-x”, and “bottom-y”. By using the above approach, the links controlling the presentation of the original media, must now bind to each the specific <area>s, informing that only that spatial area must be presented.

An example of CSV-based media objects are those imperative objects developed using scripting languages (e.g. NCLua media objects using the imperative extensions of Section 3.6). When being used to compose a multimedia application, these objects work as any other CSV-based object. The only difference is that the script code itself is responsible for generating the left and right views that compose the graphical result of the CSV-based media object.

In case the *stereo-inverted* property is set to “true”, the previous definitions of <area> elements should be swapped between the left and right element views.

3.5.2. Layout Adjustment

In the layout adjustment step, the conversion process replaces the properties of media objects duplicated in the media cloning step that specify their initial positioning and dimension.

In an NCL document, initial placement and dimension of objects are defined either by <region>, <descriptor> or <property> elements. For simplicity, in the remainder of the layout adjustment discussion, the side-by-side output format is assumed. However, the derived equations are easily adaptable to other similar formats (the top-bottom is also supported by NCLSC).

Assuming the side-by-side output format, for each cloned media object, the converter updates the values of three properties: `left`, `right`, and `width`. The `left` property specifies the distance between the left margin of the screen and the left margin of the object being presented. In NCL, this value can be either absolute (e.g., 10px), or relative (e.g., 10%) to the screen width.

As an example, let $left(x)$ be the `left` property value of a media object x, defined as percentages (%) of the screen width. The corresponding left and right-view instance values in the resulting document, $left(x_l)$ and $left(x_r)$ are given, respectively, by (see Figure 7):

$$left(x_l) = \frac{left(x)}{2} \quad \text{and} \quad left(x_r) = \frac{left(x)}{2} + 50\% + \delta(x) \qquad (1)$$

where $\delta(x)$ is the parallax (also given as a percentage of the screen width) between the left and right-view instances of the resulting elements in the output document. As previously mentioned, the parallax value can be explicitly provided by authors through the namesake property or evaluated from the *depth* property. The calculation of the final parallax based on the depth property is discussed in Section 3.5.2.1.

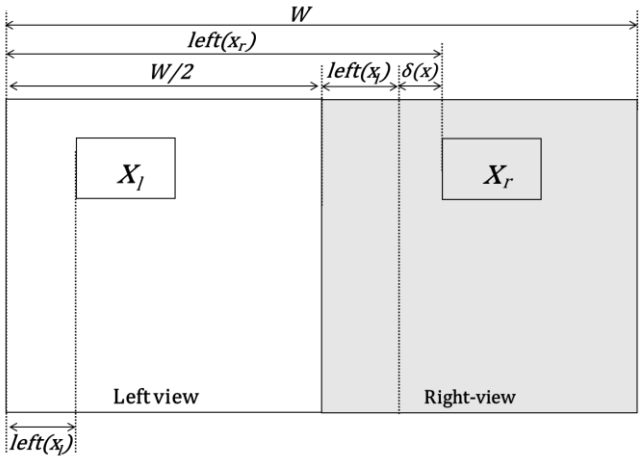


Figure 7 Spatial properties in a stereoscopic multimedia application for the side-by-side output format.

The equations to compute the resulting left and right-view values for the `right` property and for the cases in which the property values are given in pixels

are similar. The equations are similar to the ones used by the top-bottom output format, as well. However, the three properties that must be updated are: `top`, `bottom`, and `height`.

To illustrate the process performed by NCLSC in this phase, consider the <media> element in the code chunk of Section 3.1 with the additional spatial properties and the object's depth parameter defined by NCL <property> elements:

```
<media id="video" src="file:///video.mp4">
  <area id="a1" begin="10s" end="20s"/>
  <property name="explicitDur" value="50s"/>

  <property name="depth" value="1"/>
  <property name="left" value="10%"/>
  <property name="width" value="80%"/>
</media>
```

Assuming that the maximum allowed negative screen parallax is 3%, after processing the media cloning and layout adjustment steps, the following pair of <media> elements replaces the original one:

```
<media id="video-L" src="file:///video.mp4">
  <area id="a1-L" begin="10s" end="20s"/>
  <property name="explicitDur" value="50s"/>

  <property name="depth" value="1"/>
  <property name="left" value="5%"/>
  <property name="width" value="40%"/>
</media>

<media id="video-R" src="mirror://video-L">
  <area id="a1-R" begin="10s" end="20s"/>
  <property name="explicitDur" value="50s"/>

  <property name="depth" value="1"/>
  <property name="left" value="52%"/>
  <property name="width" value="40%"/>
</media>
```

The anchor defined by the <area> element is not redefined, except its *id* attribute value, because it is not a spatial anchor. The *depth* property is still present in the final stereoscopic document because its value can be changed during the document presentation, as discussed in Subsection 3.5.3. The final parallax value calculation is discussed in what follows.

3.5.2.1. From depth property to screen parallax

The final screen parallax value $\delta(x)$ for a media object x is obtained from the depth parameter $depth(x)$, as follows:

$$\delta(x) = \begin{cases} -(s \times depth(x) \times max_{\rho+}), & if\ depth \geq 0 \\ -(s \times depth(x) \times max_{\rho-}), & otherwise \end{cases} \quad (2)$$

in which *depth(x)* is the current value of the *x*'s depth property, a double value in the interval $[-1.0,1.0]$; s is a viewer-supplied scale factor in the interval $[0.0,1.0]$; $max_{\delta+}$ is the maximum positive disparity allowed by the display; and $max_{\delta-}$ is its maximum negative disparity (both $max_{\delta+}$ and $max_{\delta-}$ are given as a percentage of the screen width). The negative sign at the front of the terms is required because positive depths must produce negative parallaxes (resulting in objects in front of the screen), whereas negative depths must produce positive parallaxes (objects inside of the screen). The final depth perception based on the parallaxes presented to the user can be obtained as discussed in Section A.4.

The $max_{\delta+}$ and $max_{\delta-}$ variables denote the maximum positive and negative disparities that still allow for a good QoE using a specific display, which is usually called the *stereoscopic-box* for a S3D display.

The previous calculation allows NCLSC to adapt the resulting stereoscopic application to different display geometries, viewer distances, and user preferences. The s scale factor is a user-supplied parameter that allows end users, in line with their preferences, to adjust the final parallax. The factor can be modified at any time during the application execution. This parameter allows end users to have some degree of control over their depth perception, which also contributes to the overall QoE.

If authors need (or want) to use a fixed parallax settings, they still can do it by hardcoding the parallax property. In the previous example, it is sufficient to include the <property> element with attribute name equals to "parallax" and its corresponding value (in pixels or percentages of screen). In such a case, it is recommended to choose a fixed parallax value that works reasonably well across a

large range of display sizes and viewing distances. An example of such rule-of-thumb is the percentage rule (MENDIBURU, 2012), commonly used in cinematography, which states that: (i) negative parallax should not exceed 2%– 3% of the screen width; and (ii) positive parallax should not exceed 1%– 2% of the screen width.

3.5.3. Behavior Cloning

The behavior cloning step is responsible for replicating the behavior of each original media object onto that of the resulting left and right instances. Such a replication is directly related to the temporal model used by the multimedia language.

In NCL, the application behavior is defined by causal relationships (mainly specified by <link> elements). NCL is an event-oriented language. Occurrences of events trigger actions that cause the occurrences of other events. NCL has three types of events: *presentation* of a set of information units (an anchor) of a media object; *selection* of an anchor being presented; and *attribution* of a value to an object's property. More precisely, the `onBegin`, `onEnd`, `onPause`, `onResume`, or `onAbort` conditions for any event type may cause the `start`, `stop`, `pause`, `resume`, or `abort` actions for other (or the same) events. Link conditions and actions are associated with media objects through <bind> elements, defined within a parent <link>.

As an example of an NCL link, consider the following code block:

```
<link id="orig-link" xconnector="onBeginStop">
  <bind role="onBegin" component="x"/>
  <bind role="stop" component="y"/>
</link>
```

The above link establishes that when the media object x starts its presentation, the presentation of media object y must be stopped.

NCL behavior is also guided by content and presentation adaptations, represented by <switch> and <descriptorSwitch> elements, respectively. Because these elements are "syntactic sugars" to more complex structures of <link> elements (LIMA; SOARES, 2013), the remainder of this section details only how NCLSC handles <link> conversions.

The <link> conversion can be done in at least two ways:

(i) each original <link> element generates two links with the same conditions, but with actions targeting the new <media> elements representing the left views and the right views, respectively; or

(ii) the original link generates only one <link> in which each action is duplicated to operate on both views.

In both cases, if any condition of the original link refers to a <media> element that has been duplicated in the media cloning phase, in the produced link (or links), the condition must refer to the new left view instance.

As an example, take the link `orig-link`, depicted in the previous code block. Assuming that a media x has been duplicated in the media cloning phase, if the link update method (i) is applied, the following two links replace the `orig-link` element:

```
<link id="orig-link-L" xconnector="onBeginStop">
  <bind role="onBegin" component="x-L"/>
  <bind role="stop" component="y-L"/>
</link>

<link id="orig-link-R" xconnector="onBeginStop">
  <bind role="onBegin" component="x-L"/>
  <bind role="stop" component="y-R"/>
</link>
```

Alternatively, if the link update method (ii) is applied to `orig-link`, the following is produced as output:

```
<link id="orig-link" xconnector="onBeginStop">
  <bind role="onBegin" component="x-L"/>
  <bind role="stop" component="y-L"/>
  <bind role="stop" component="y-R"/>
</link>
```

The synchronous hypothesis assumed by some NCL players (SOARES *et al.*, 2013) states that the time between the application of an action and the onset of its effect is considered zero. If that hypothesis can be assumed, the two ways for converting a link produce the same effect on the QoE of produced stereoscopic applications. However, if the synchronous hypothesis cannot be assumed, which alternative leads to a greater delay between the action's effects on the two view instances depends on the language player implementation. Although the media cloning step guarantees a synchronized presentation of the two media views when

they are both on display, it cannot ensure that the two media views start at the same time.

Currently, NCLSC is able to use both link conversion approaches, (i) and (ii), and the authors may choose which one guarantees that a particular NCL player implementation will execute both actions (in the right and left-instances) as close as possible to each other. By default, NCLSC employs the first approach, since this is the best alternative when using the ITU-T NCL player reference implementation[5] (Subsection 3.8.2 brings experimental results supporting that decision).

In addition, if the original <bind> element specifies the assignment of a value of a spatial property of the original <media> element, such change must be replicated to the left and right view instances. As aforementioned, properties are specified through the <property> element, and spatial properties include `left`, `right`, `top`, `bottom`, `width`, `height`, and `depth`. The changes in the values of those properties must be in agreement with equations in Subsection 3.5.2.1. However, sometimes, the spatial property values cannot be computed statically. To calculate the values at run-time, NCLSC redirects any attempt to change the positioning properties to an NCLua script node. This script computes the final positioning property for the left and right view instances and redirects these changes to the corresponding target view instances.

Figure 8 shows the conversion process for dynamic changes in spatial properties and how the original <link> is changed to use the NCLua script as a proxy to change the positioning properties. In Figure 8, the original <link> (`orig_link`) changes the spatial properties of a media object (`main_video`) when this media object begins its presentation. After the behavior cloning step, the original link (`orig_link`) is converted into three links: one (`link_a`) that sets the desired values to the NCLua script proxy, instead of directly changing the properties of the `main_video` object; and two others (`link_b` and `link_c`) that redirect the new values calculated by the NCLua to the left and right views.

[5] Available at http://www.ginga.org.br

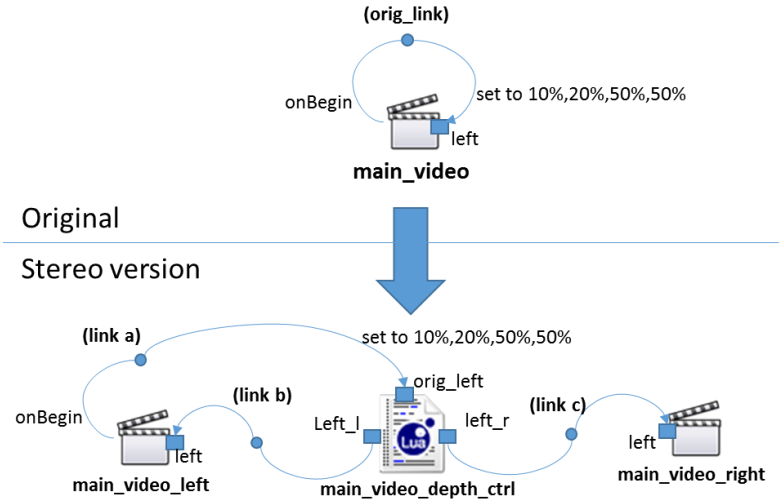


Figure 8 NCLua script controlling a positioning property.

3.5.4. Implementation and Usage

NCLSC is implemented as a Lua script that can be used as both a standalone application and an embedded NCLua media object that can modify the application in which it is inserted.

The standalone version of NCLSC can be used at the server-side or the client-side. For example, in IPTV or hybrid broadcast/broadband DTV systems, the conversion process can be performed at the server side before sending the requested application to the client. In this case, when requesting an application, the client must inform the display screen size, the optimal viewer distance, and the scale factor, so that the proper parallax parameter could be calculated.

Applications transmitted by broadcast (e.g. those transmitted in terrestrial digital TV systems), cannot be converted at the server side, because they target clients with different characteristics and needs. In this case, there are two possibilities. First, NCLSC can be embedded into the language player; the language player is then responsible for recognizing the depth/parallax properties of the received application and converting the application to a new stereoscopic version, if an S3D display is available. Second, the application transmitted by broadcast can be a wrapper application embedding NCLSC, implemented as an NCLua media object, as shown in the following code chunk:

```
<body id="wrapper">
  <port id="entry" component="nclsc"/>
   <property name="url" value="original-2d.ncl"/>
  </media>
</body>
```

In this case, the original document is passed as a parameter to NCLSC using the <property> element. When the application starts, NCLSC starts (see <port> element), it produces the stereoscopic multimedia document, based on the "original-2d.ncl" document and adds this new application into the wrapper document. Finally, NCLSC starts the new stereoscopic multimedia presentation.

3.5.5. Other declarative multimedia languages

Up to now, the stereoscopic conversion has been mainly presented in the scope of the NCL language. However, the proposals can be adapted to other 2D-only multimedia language as well. Clearly, some adaptation to the model of those languages may be needed. A brief discussion on how the conversion process can be adapted to SMIL and HTML is presented in what follows.

In SMIL, the media cloning step should focus mainly on duplicating the <ref>, <animation>, <img>, <text>, <textstream>, and <video> elements. As the temporal specification of SMIL is defined in the document structure, a simple way to specify that both the left- and right-side media objects will play during the same time is replacing the original media element by a <par> container with the two new media objects (representing the left and right side of the original media) inside that container. Unfortunately, that approach does not necessarily guarantee a frame-based synchronization between the elements synchronized. The frame synchronization skew is dependent on the SMIL player itself. Some control on the synchronization between two cloned objects can be achieved using the *syncBehavior*, *syncTolerance*, and *syncMaster* attributes. Concerning the support for CSV-based media elements in the stereoscopic multimedia presentation, the *panZoom* property may be used. The layout adjustment step should focus mainly on the <layout> and <region> elements of the language, with features similar to the ones of NCL <region> element. Moreover, dynamic changes in properties and animations should also be duplicated.

In HTML, some of the elements that define graphical content are , <canvas>, <img>, <video>, etc. Therefore, when converting an HTML document to its stereoscopic version, an instantiation of the stereoscopic conversion process should duplicate those elements. Since the HTML structure itself also defines the positioning of the graphical elements, it may be interesting to clone the elements into two different trees that completely emulate the two views of the document. Moreover, the layout adjustment step should focus also on updating the CSS elements and rules in order to correctly position the elements.

HTML itself does not provide behavior (temporal) specification. Thus, authors need to use imperative JavaScript code (or CSS animations) to control the behavior of a multimedia application written in HTML. In order to keep the same behavior on both sides of an HTML stereoscopic application, the process of converting an HTML/JavaScript code must also guarantee that such changes are replicated. Different from NCL, there is no mirroring scheme natively supported by HTML language features. Thus, avoiding the synchronization skews between two dynamic media elements (e.g. two <video> elements) is not straightforward. Still, it is possible to implement such a feature using JavaScript and the <canvas> element, which is provided by the HTML5 specification. In such approach, the conversion process should replace the original dynamic media element (e.g. <video>) with two <canvas> elements in the output document. A JavaScript code should then be responsible for getting the original frames cropping them and drawing them synchronously in each <canvas> element. CSV-based media support could work similarly, with the JavaScript code cropping and drawing the frames in their specific <canvas> elements.

3.6. Imperative Support for Stereoscopic Media Content

Imperative scripting languages can be used to control—e.g. using a DOM-like API (HÉGARET *et al.*, 2004)—the properties described in Section 3.4. In addition, some script languages provide a canvas-like API that allows scripting languages to draw on a 2D surface. The drawing results can then be integrated into the overall multimedia presentations. This is the case of HTML5/JavaScript (CABANIER *et al.*, 2015) and NCLua (SOARES; MORENO;

SANT'ANNA, 2009) canvas APIs. HTML provides a specific graphical element (<canvas>) in which JavaScript code can draw. The integration of NCLua canvas into NCL applications is done like any other media object through the <media> element.

Canvas APIs usually provide objects, methods, and properties to draw and manipulate graphics on a canvas drawing surface. They are commonly based on a drawing context that has a state (a set of properties) and methods. The methods can be used to change the context state (change the values of the properties) or for drawing on the surface. The state usually has influence on how the objects are drawn. To allow authors to control how the stereoscopic views are drawn on 2D canvas surfaces, the following method are proposed (which are added to the canvas context and control its states):

ctx:attrStereoMode(stereoMode: string) gets or sets the output stereo mode of the canvas element. Currently, the two supported stereoscopic modes are 'side-by-side' (or 'SbS') and 'top-and-bottom' (or 'TaB'). The default value is 'side-by-side'.

ctx:attrStereoSwap(swap: boolean) gets or sets a parameter to specify whether the left and right pair should be swapped. The default value is '*false*'.

ctx:attrParallax(parallax: number) gets or sets the current parallax (offset) used by the drawing methods. The default value is '0'.

ctx:attrDepth(depth: number) gets or sets the depth property. Depth is a flexible way to specify the **ctx:attrParallax**, which allows the final offset to be adapted to different display sizes. Depth must be a double value in the $[-1.0,1.0]$ interval. The default value is '0'.

ctx:attrMaxParallax(maxNegParallax: number, maxPositiveParalax: number) gets or sets the maximum negative and positive parallaxes properties of a canvas context. Those properties are used to calculate the final offset based on the depth property. Maximum negative parallax must be in pixels or percentage of the canvas width. The default value is "3%". Maximum positive parallax must be a value in pixels or percentage of the canvas width. The default value is "2%".

ctx.attrStereoResizeContent: specifies whether the objects must be resized based on their depth or not. Possible values are ‘true’ and ‘false’. The default value is ‘false’.

ctx.attrStereoResizeFactor(minResizeFactor: number, maxResizeFactor: number): if **ctx.attrStereoResizeContent** property is ‘true’, this compound property is used as the minimum and maximum resize factor of a media object. This is equivalent to specify that elements with $depth = -1.0$ will be resized by a factor of **minResizeFactor** and elements with $depth = 1.0$ will be resized by a factor of **maxResizeFactor**. The other depths will be linearly resized between those values. The default value of **minResizeFactor** and **maxResizeFactor** is 1.0.

3.7. Imperative Support Implementation

Aiming at providing a unified framework to support declarative and imperative abstractions, the above extensions were integrated into NCLua in a library named NCLuaCanvaS3D. “NCLua API” (SANT’ANNA; CERQUEIRA; SOARES, 2008) is the set of Lua extensions that provides integration between Lua scripts and NCL documents. NCLua objects are media objects developed using the NCLua APIs that are embedded into NCL documents. The NCLua API is composed of the following modules: *event*, *canvas*, *persistent*, *settings*, and *security*. From those modules, NCLua canvas is the one of special interest in developing CSV-based media content. “NCLua canvas” is the API that allows NCLua objects to perform graphics operations. They include drawing primitives such as lines, circles, rectangles, etc.

NCLuaCanvaS3D is a wrapper on the NCLua canvas module that eases the development of CSV-based media content. NCLuaCanvaS3D allows NCLua programmers to use the default NCLua canvas API and to control how those primitives are rendered into a CSV-based canvas. The graphical result of the canvas rendering is codified as one of the frame-compatible CSV formats (side-by-side and top-bottom are currently supported).

An NCLua script that wants to generate CSV-based media content just need to include the NCLuaCanvaS3D wrapper as follows:

```
local canvas = require "canvas_s3d"
```

The code overwrites the local canvas module with the NCLuaCanvaS3D library. From now on, NCLua authors uses the drawing primitives as usual (following the standard NCLua canvas API). However, the methods called by the script are redirected to NCLuaCanvaS3D, which is responsible for redirecting the drawing primitives to the two underlying canvas: the left and right canvas.

In addition to the standard NCLua APIs, NCLuaCanvaS3D also implements the methods discussed in Section 3.6. The authors can control the depth (`canvas:attrDepth(…)`) or the parallax (`canvas:attrParallax(…)`) of individual objects that are drawn on the stereoscopic canvas. It is also possible to control how the canvas is drawn using the (`canvas:attrStereoMode(..)`). The next code example shows a very simple case of drawing a black circle that pops out of the screen using NCLuaCanvaS3D API.

```
local canvas = require 'canvas_s3d'
...
canvas:attrStereoMode('side-by-side')
...
canvas:attrDepth(1.0)
canvas:attrColor(255, 0, 0, 255)
canvas:drawEllipse('fill', xc, yc, len, len)
...
```

Figure 9 shows an example of a dynamically generated content using NCLuaCanvaS3D.

Figure 9 Dynamically generated animation (using imperative stereoscopic 3D canvas API) that presents moving circles inside and outside of the screen.

The following code shows how to integrate the NCLua script content of Figure 9 into a parent NCL document.

```
<media id="cube" src="circles3d.lua">
  <property name="stereo-content" value="side-by-side"/>
  ...
</media>
```

In embedding CSV-based content based on the NCLuaCanvaS3D library, the *stereo-content* property can be omitted. The NCLuaCanvaS3D library can automatically notifies changes on this property to the NCL parent, without the user being aware of it.

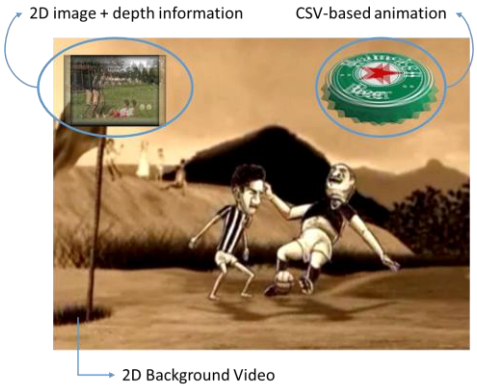
3.8. Experimental Results

As previously mentioned, the proposals of this chapter led to two practical implementations: NCLSC and NCLuaCanvaS3D. This section presents experimental results based on those implementations. First, it presents some applications developed using NCLSC and NCLuaCanvaS3D. Second, their performances are analyzed.

3.8.1. Applications

NCLSC and NCLuaCanvaS3D can be used either individually or together, to develop stereoscopic multimedia applications. Applications developed using those implementations are ready to run in current NCL players (without any change on the player). Since NCL is the declarative standard for developing interactive services for ISDB-T (Integrated Services Digital Broadcasting—Terrestrial), applications using the provided implementations can be seamlessly integrated into ISDB-T delivery chains. This section presents three simple stereoscopic multimedia applications developed using the proposed approach. The aim is to show examples of how the real depth information can be used to catch user attentions, spatially structure user interfaces, or provide realistic representation of media objects. The applications discussed can run, without any change, in current NCL players (and be transmitted ISDB-T chains).

Figure 10 shows an advertisement application composed of a 2D-only video (the main broadcasted video) and two additional media objects annotated with depth, popping-out of the screen. The one on the left of the figure is a 2D-only media object annotated with depth. The one on the right side is a CSV-based animation (based on NCLuaCanvaS3D) of a beer brand. In the application, the depth information is used as a way to call the users' attention. The use of 2D-only video with advertisement media popping out of the screen can help to reduce the user's eye fatigue while maximizing the advertisement impact (JUNG *et al.*, 2008).

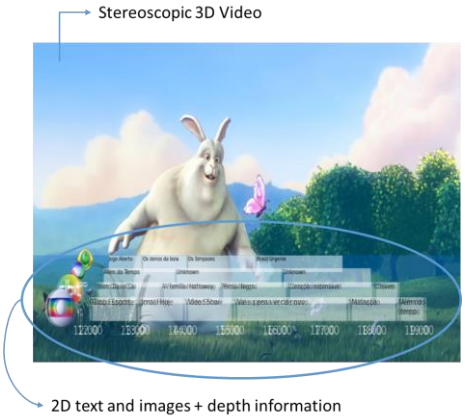


(a)

(b)

Figure 10 Example of an advertisement application using stereoscopic effects: (a) schematic view; (b) side-by-side format.

The application shown in Figure 11 is an *Electronic Programming Guide* (EPG) using real depth as a metaphor to organize the TV program schedule in space. In the figure, the real depth information is used to organize the schedule of the different channels. Each channel is presented on a different distance from the user, providing a layered-like interface. The user can navigate between the channels, bringing them closer or farther away.

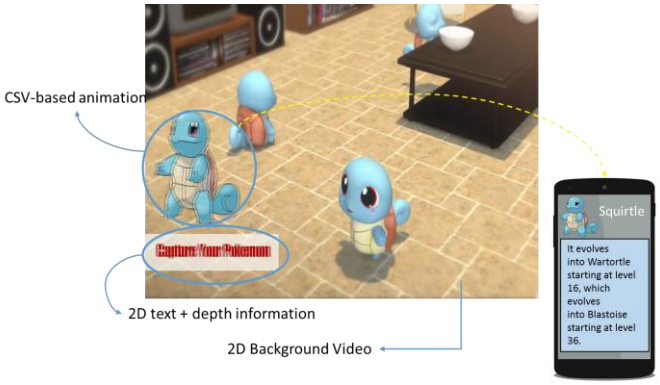


(a)

(b)

Figure 11 Example of an EPG application using the real depth information to organize elements in the user interface: (a) schematic view; (b) side-by-side.

Figure 12 is a distributed multimedia application that *integrates companion screens* (a feature supported by NCL) with the main stereoscopic multimedia application. Secondary devices may capture stereoscopic objects (CSV-based) popping out of the screen. When capturing those objects, additional information is displayed on the secondary device.



(a)

(b)

Figure 12 Example of a stereoscopic multimedia application integrated with a secondary device: (a) schematic view; (b) side-by-side format.

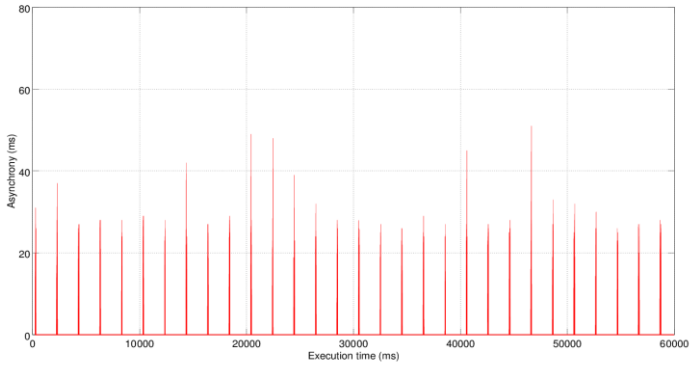
3.8.2. Performance Analysis

This section analyzes the performance of NCLSC and NCLuaCanvaS3D. First, NCLSC is analyzed concerning the asynchrony that can arise when running stereoscopic multimedia applications. To analyze NCLuaCanvaS3D, this section focuses on its rendering frame rates. Measurements are performed using the ITU-T reference implementation of the NCL player, running in a computer with an Intel Core i7 processor and 8GB of RAM. It is worth noting that whereas this is not a real world low-end platform, it might bring us some insights on the possibilities of using the proposed approach.

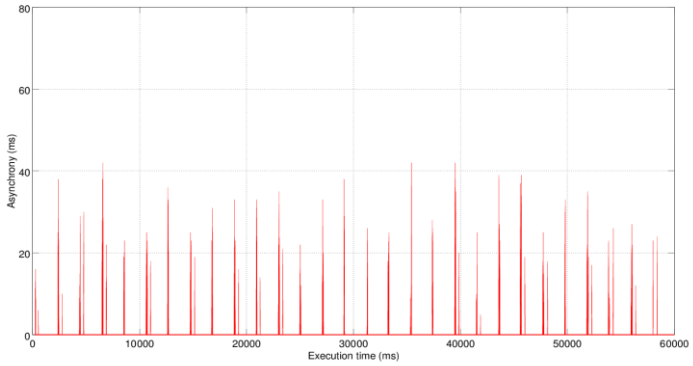
In stereoscopic multimedia applications generated by NCLSC, there is no skew between the left- and right-side media objects when they are both on the screen. Such a frame-by-frame synchronization is guaranteed due to the mirroring scheme of NCL. However, in particular NCL player implementations the results

coming from the two equivalent actions on the left- and right-side media objects may have some skew. For instance, the left-side media object may be on the screen while its right-side counterpart may not be shown yet. To test the skews among actions, an application that presents two alternating media objects was used. Each time a media object starts, it stays for 2 seconds on the screen. The total time of the application is 60 seconds.

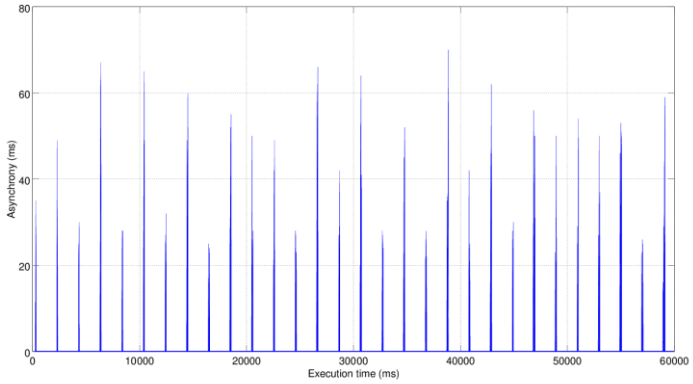
Figure 13 shows the measurements of the asynchrony between the left- and right-side media objects representing the same original media object. The *x-axis* is the total time of the application in milliseconds. The *y-axis* informs, also in milliseconds, for how long the application has been de-synchronized. For instance, a point (5000, 10) means that at the time 5s the application suffers 10ms of asynchrony. In other words, the measurements show for how long a media object representing one side of the application is on the screen while its counterpart is not being shown yet. Both link conversion methods, (i) and (ii), discussed in Section 3.5.3 were tested. As can be notice, method (i) has produced less asynchrony using the ITU-T NCL player. That is the reason why it was chosen as the default method.



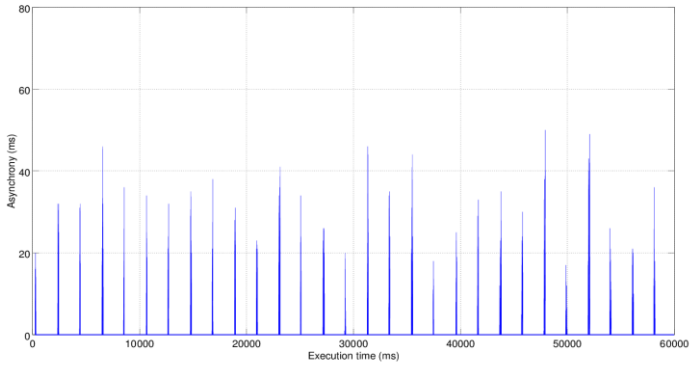
(a) Links conversion method (i) with images-only media objects.



(b) Links conversion method (i) with video-only media objects.



(c) Links conversion method (ii) with images-only media objects.



(d) Links conversion method (ii) with video-only media objects.

Figure 13 Asynchrony between left and right side of a stereoscopic multimedia application.

Whereas NCLSC may produce some asynchrony between the left- and right-views, NCLuaCanvaS3D operates synchronously—i.e., it draws both sides of a stereoscopic 3D media content at the same time—and does not suffer from the

aforementioned asynchrony issues. Therefore, to analyze NCLuaCanvaS3D, the focus is placed on its performance. Figure 14 shows the rendering frame rates of an NCLua applications similar to the one presented in Figure 9, but that draws rectangles on the canvas instead. The figure shows the frame rates of the "pure" NCLua canvas version of the application and of its stereoscopic version using NCLuaCanvaS3D. In the beginning of the curve (until approximately 1000 rectangles), the performance of the default NCLua canvas and of NCLuaCanvaS3D are similar. Such a similarity is explained by the fact that the NCLua implementation limits the rendering performance of NCLua objects up to 60fps. As expected, the performance of NCLuaCanvaS3D, which draws both the left and right views of the presentation, is lower than the performance of the pure NCLua canvas. Notice, however, that NCLuaCanvaS3D can provide enough performance to be used in real scenarios. According to the provided measurements, it is possible to achieve rendering performance above 30fps until approximately 2500 rectangles.

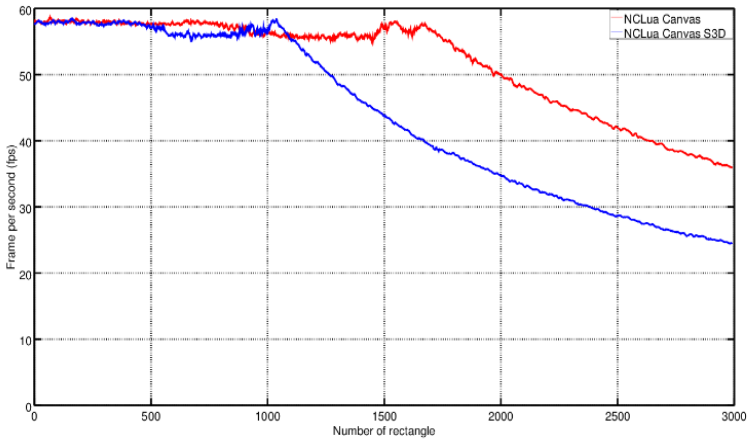


Figure 14 Frame rates of a pure NCLua application and its stereoscopic version using NCLuaCanvaS3D.

More interesting than knowing the frame rate of applications using NCLuaCanvaS3D, is to measure the overhead of using S3D compared with the pure NCLua. Figure 15 shows the ratio between the frame rate of NCLuaCanvaS3D and the pure NCLuaCanvas. As can be seen in the figure, in the worst case, the performance of NCLuaCanvaS3D is close to 70% of the pure NCLua canvas (which only draws one view and cannot achieve stereoscopic viewing). In

the beginning of the curve, the ratio between both cases varies close to 1 due to the 60fps frame rate limit imposed by Ginga-NCL reference implementation.

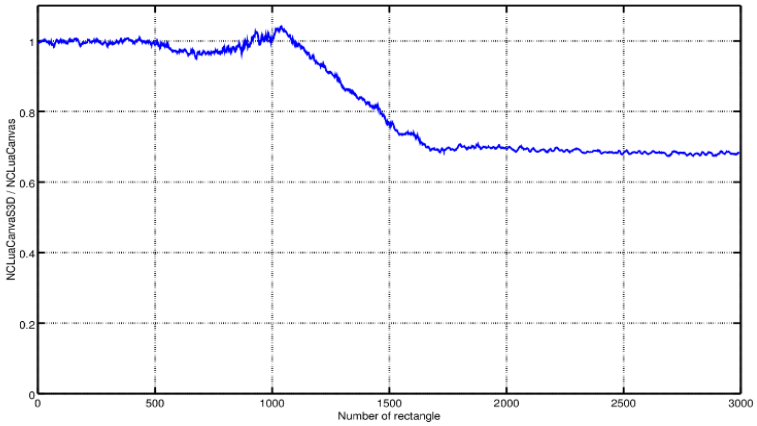


Figure 15 Relationship between the performance of NCLua and NCLuaCanvaS3D.

3.9. Discussion

At this point, some remarks should be made concerning the space and time complexity of the stereoscopic multimedia applications, the problems that can disturb the end-user QoE, and the limitations of the proposed solution.

3.9.1. Space and time complexity

The processing overhead of the produced stereoscopic applications, in comparison with the original one, heavily depends on how a particular language player is implemented and for which tasks it is optimized. In particular, the previous section has provided measurements related to the ITU-T NCL player. This section is more general and presents a complexity analysis of stereoscopic multimedia applications produced by NCLSC.

Beginning by the space complexity of the produced application, consider an original document with N media objects, and L links. After the media cloning step, the produced stereoscopic application has at most two times more media objects than the original one, that is, 2N media objects. This worst-case scenario occurs

when every media object has a graphical representation and needs to be duplicated. Moreover, for each duplicated object in this first step, NCLSC can add one NCLua script object to control the parallax between the left and right view instances at runtime. This processing is also done in the behavior cloning phase. Thus, in the worst case, the total number of media nodes in the output document is 3N.

For each link in the original document that changes a spatial property, the behavior cloning step outputs tree links. These links are responsible for redirecting changes on spatial properties to the associated NCLua script object and then to the left and right view instances (see Figure 8). Therefore, in the worst case, the total number of links in the output document is 3L.

As for the time complexity of running the final stereoscopic application, it can be estimated from the number of events generated at runtime that results in changes in the application state. In the worst case, the number of events generated by the produced stereoscopic application is twice that generated by the original application. That is, in the worst case we may end up running two instances of the same application.

3.9.2. Quality of Experience

Two main parameters affect the QoE of the stereoscopic multimedia applications produced by NCLSC: the parallax between the left and right view instances of a same media object; and the synchronization skew between the two views.

The parallaxes among objects is in complete control of the application author (who may also allow for some end user customization) who is tasked with the handling of depth (or parallax) of media objects. He can control such depth information to preserve or improve the QoE. NCLSC only gives him the tools, as discussed in Section 3.2.1. As previously mentioned, the authors can use a fixed parallax value between the left and right views. That approach is sufficient for achieving depth perception on S3D displays. However, some studies (SHIBATA *et al.*, 2011) (CHEN, YING *et al.*, 2014, p. 3) have reported that if the parallax value is kept fixed while the display size and viewer distance vary, there is a significant drop in the user quality of experience (QoE). By using the *depth* property, NCLSC

allows adaptation to different viewing conditions. Moreover, the end user can also customize his experience by controlling the final parallax by using the scale factor (discussed in Section 3.5.2.1).

To keep the views synchronized is more challenging. The left and right view instances must begin their presentations at the same time and keep their presentation as close as possible of a frame-by-frame synchronization. As previously mentioned, when the two representations (left and right) of the same media object are being shown on the screen, the frame synchronization is preserved taking advantage of the "ncl-mirror" scheme of NCL (see Section 3.1). However, keeping the same starting time for the left and right views depends on the NCL player implementation. This drawback is mainly due to the NCLSC approach of relying as much as possible on the declarative constructions of the declarative language. In particular, Section 3.8.2 has provided some measurements of the de-synchronization that can arise when running NCLSC applications on the ITU-T Ginga-NCL reference implementation. Although those experiments are application-specific, they provide evidence that it is possible to achieve good QoE for stereoscopic multimedia applications generated by NCLSC. According to (GOLDMANN; LEE; EBRAHIMI, 2010), a skew below 80ms leads to good stereoscopic 3D visual quality, while a skew larger than 200ms annoys the user.

3.9.3. Limitations

As aforementioned, one of the main advantages of using the NCLSC solution is that it does not need native S3D support from the language player. Despite that, some limitations should be mentioned.

The use of scripting languages instead of native software support has clearly some overhead that can influence the performance of the application.

Another issue is derived from the fact that NCL applications have no control over the pointing device, which is under complete control of the NCL player implementation. Thus, NCLSC is not able to duplicate pointers correctly to the left and right view instances, providing an S3D view. Even if it could be represented in an S3D view, the use of 2D pointers in S3D systems also has ssome major drawbacks, such as the position of the cursor in depth and handling of

occlusions (SCHEMALI; EISEMANN, 2014). Whereas this may be considered a big issue in PC environments, we consider it a minor one in interactive TV, in which the most common interactive device is still the remote control, i.e., no pointer is visible.

4
Layered-depth-aware Multimedia Applications

Some 3DV delivery chains based on conventional stereoscopic video representations are already in deployment (e.g., in 3DTV and 3D Cinema). The proposals of Chapter 3 allow for extending those chains by providing interactive content that is ready to run on currently available hardware and with support of 2D-only language players. Although CSV-based formats have been standardized and used for quite some time, they have some conceptual drawbacks. For instance, CSV-based multimedia services are restricted to only one viewpoint, do not support movement parallax, and cannot be easily customized to users' preferences or display characteristics (FEHN, 2005). The latter feature is usually required for good QoE when watching S3D content.

An alternative to CSV is to transmit video signals (or textures) and per frame depth-maps (FEHN, 2005). One or more stereoscopic videos can then be rendered at the client side. That approach allows for customizing the depth perception, supporting head motion parallax and multi-view 3D displays. In addition, the use of depth information is interesting from the compression point of view, saving bandwidth in transmission networks (SMOLIC *et al.*, 2007).

Following this direction, this chapter considers a depth-based delivery chain as a new scenario. Based on such a delivery chain, the chapter extends multimedia languages to allow for a seamless integration between synthetic (both 2D and 3D) and natural (real) scenes. A graphics architecture supporting the rendering of multimedia applications using those extensions is also in the scope of the chapter. The architecture is able to compose and generate multiple views for a multimedia scene and is implemented using OpenCL, aiming at achieving a real-time (at least 30 fps) performance.

Previous work has handled the integration of synthetic 3D media objects into multimedia scenes (CESAR; VUORIMAA; VIERINEN, 2006) (AZEVEDO, 2010) (SOUZA *et al.*, 2010). However, in most of those cases, the 3D media object is usually rendered on a 2D surface (window) that is used to compose the

entire (2D) multimedia scene. Stereoscopic and multi-view displays are not in the scope of those works. Partial occlusion (such as the one depicted in Figure 16) between different media objects is not supported either in those works.

Figure 16 Example of a partial occlusion between a 2D+depth video and a 3D model. [6]

Using a depth-only approach, such as the one proposed by (LE FEUVRE; MATHIEU, 2013), is interesting and can provide the graphical composition we aim to support. However, it may suffer from introducing artifacts when rendering multiple views for 3D displays. Such effect increases with the distance from the virtual views to the original camera position. Additional occlusion layers help to minimize those undesired artifacts. The video representation using multiple depth layers is the layered-depth video (LDV), which is an extension of the concept of layered-depth image (LDI)(SHADE *et al.*, 1998) (Section A.3 brings more details on LDI/LDV). The proposals in this chapter consider the possibility of using multiple LDV media objects. Moreover, the graphical composition of the various objects takes advantage of the layered-depth representation, which allows for storing as much occlusion information as possible.

The remainder of the chapter is organized as follows. Section 4.1 introduces the motivating scenario for the proposals in the chapter. Section 4.2 presents an overview of the proposed solution. Section 4.3 discusses extensions to multimedia languages that can be used in the context of depth-based 3DV delivery chain. Section 4.4 presents the graphics architecture based on the LDV concept to support the extensions of Section 4.3 and render multiple views of multimedia applications. The architecture supports the graphical composition of 2D, 2D+Z, LDV, and 3D,

[6] Extracted from “Augmented reality occlusion demo with X3D and Microsoft Kinect”. Available at: https://www.youtube.com/watch?v=mHhDUR06PfI.

natural and synthetic objects. Section 4.6 brings some experimental results. Section 4.7 is reserved for final considerations.

4.1. Motivating Scenario

Figure 17 depicts the 3DV delivery chain scenario. It is similar to the scenario defined in Chapter 3, but the main video is based on a depth format instead. In particular, 2D+Z or LDV are considered. Similar delivery chains have been studied, respectively, by ATTEST (2D+Z) and 3D4YOU projects. MPEG-C Part 3 (ISO, 2007), one of the results of ATTEST, can be used to synchronously multiplex the depth frame and the main texture frame, being backward compatible with current 2D-only delivery chains. Another alternative is using video coding algorithms that take advantage of the interdependency between the texture and depth data (CHEN, YING; VETRO, 2014).

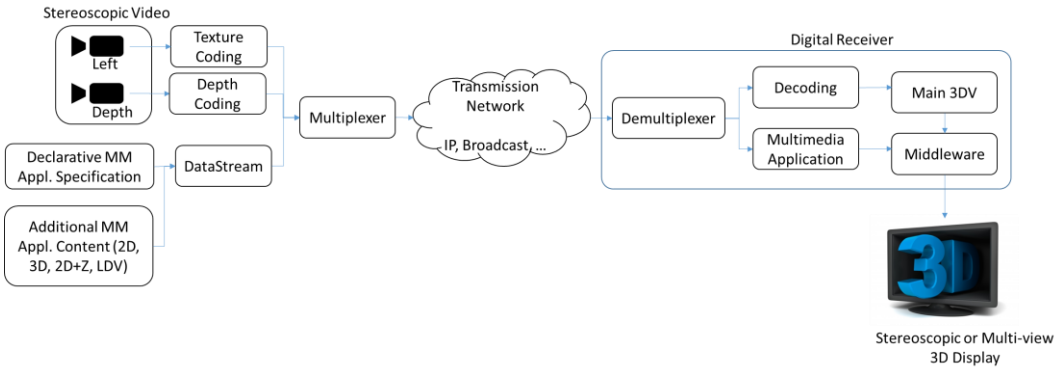


Figure 17 Example of an interactive 3DV delivery chain based on 2D+Z.

In Figure 17, similar to the scenario presented in Chapter 3, multimedia applications can be multiplexed in the main transport stream or retrieved from another transport mean. The language player is responsible for presenting the multimedia application in agreement with the multimedia document specification. In the scenario, natural media objects that are part of the multimedia application are not restricted to 2D-only objects, but may also be natively represented as 2D+Z or LDV. Indeed, the main depth-based 3DV may also be part of the multimedia application. Synthetic 3D objects may also be supported. In this case, it is possible to extract the texture and depth information after rendering them using standard APIs such as OpenGL or DirectX.

Unlike the receiver in the scenario of Chapter 3 (a low-end device that does not support rendering synthetic 3D objects), this chapter considers *high-end* digital receivers—or *interactive high-end* terminals as defined by (CESAR; VUORIMAA; VIERINEN, 2006). Such receivers can natively render 3D objects at the client side—e.g., using DirectX, OpenGL APIs—and may also support GPGPU (General Purpose for GPU) programming APIs, such as OpenCL and Cuda.

Moreover, the scenario discussed in Chapter 3 only targets receivers with stereoscopic displays. In addition, the proposal in this chapter must also support receivers equipped with multi-view displays.

4.2. Overview of the Proposed Solution

This chapter aims at providing a declarative scene environment to be implemented by high-end digital receiver to support the seamless integration of various types of media objects. More precisely, the different types of media content to be supported are:

- *Natural 2D:* the typical 2D-only videos and images.
- *Declarative 2D synthetic*: content developed using declarative languages and which render a 2D-only surface. Examples of declarative 2D synthetic objects are those specified using SVG, HTML, SMIL, or NCL.
- *Imperative 2D synthetic:* content developed using imperative languages and which render a 2D-only surface. Examples of imperative 2D synthetic objects are those developed using NCLua and JavaScript canvas APIs.
- *Natural 3D*: videos or images represented in 2D+depth or layered-depth representation.
- *Declarative 3D synthetic*: content developed through declarative languages and which render a layered-depth data (usually, a 2D+depth only). This is the case of scenes developed using X3D, VRML, or even those using a 2D-only scene plus the declarative extensions provided in Subsection 4.3.1.
- *Imperative 3D synthetic*: content developed using imperative languages and that renders layered-depth data. This is the case of NCLua scripting using the API proposed in Section 4.3.2.

In our proposal, each media object content type has an associated media player that is responsible for decoding (in the case of natural) or rendering (in the case of synthetic) the media content. The issues addressed in this chapter are how to compose (in a per-pixel mode) those media objects, allowing the authors to control such a composition, and how to generate multiple views to be presented on 3D displays. Different from related work, the considered depth information may contain multiple layers of textures and depth, i.e., it can be LDI.

In controlling the integration among the different media types, a "glue language" that supports a scene model based on LDI (see Figure 18) must be used. One solution would be to design a new language to do so. Instead, an existing 2D multimedia language is used as glue language, the NCL language, and extensions to support the specification of layered-depth properties of media objects are proposed (in Section 4.3). Indeed, any multimedia language (with the extensions proposed) could be used. By using NCL, the synchronization features of the language can also be used to control the spatial and temporal behavior of the media objects.

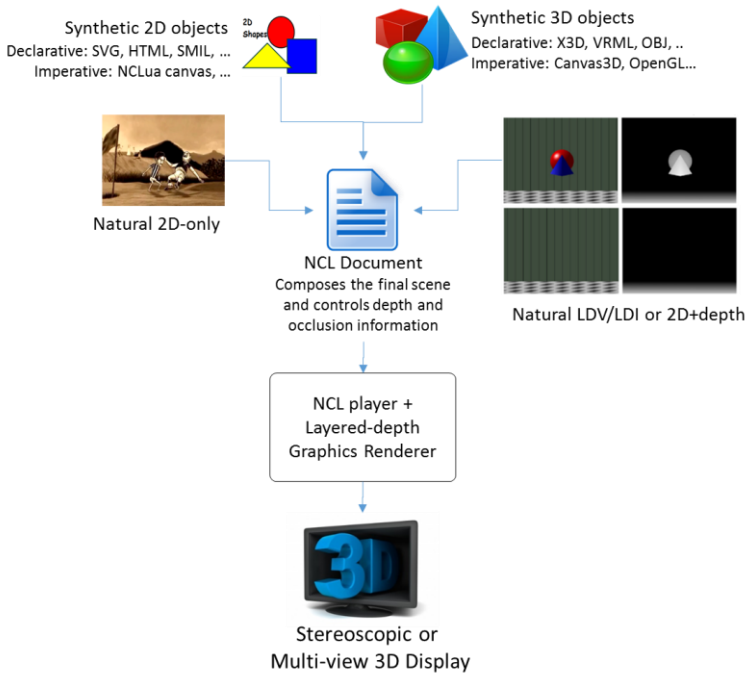


Figure 18 NCL Document as a glue language for 2D, 2D+Z, LDV, and 3D media objects.

4.3. Layered-depth Extensions

The multimedia language extensions provided in this chapter, as those defined in Chapter 3, can be divided into declarative and imperative ones. The declarative extensions are properties to be associated with media objects. Imperative extensions aim to be used by imperative scripting languages that provide a canvas-like API.

4.3.1. Declarative Extensions

The following properties are associated with media objects to allow programmers to define the depth and occlusion information of these objects. The layer parameter is an integer that indexes the different layers. The occlusion layer values are indexed starting from 1. For 2D-only media, Layer 0 is the content of the media itself.

depth(layer: number) associates a unique depth information to a media object layer. The property's value, in the interval $[-1.0, 1.0]$, informs how far the object must appear from viewers when rendering on a 3D display. It is like the depth property defined in Chapter 3, but multiple depth layers are supported.

depth is the equivalent short form for **depth(0)**.

depthMap(layer: number) associates a per-pixel depth information to media objects. The value of a property **depth(layer)** is an URI to a gray-scale image or video that must be used to generate the per-pixel depth information.

depthMap is the equivalent short-form for **depthMap(0)**.

occlusionLayerTexture(layer: number) specifies the occlusion layer texture data of media objects.

depthScale a scale factor to **depth** and **depthMap** properties' values.

All those properties are integrated into NCL through <property> elements, children of the <media> element. Since NCL allows to group media objects'

properties (to be (re)used in different media objects) into descriptors (<descriptor> element), the properties can also be associated with descriptor.

NCL constructions can dynamically control the values of these properties (e.g., by using <link> elements), allowing complex animation effects. NCL links define spatiotemporal relationships between media objects, thus defining the temporal behavior of applications. To exemplify the declarative manipulation of 3D effects using NCL, the following source code shows how the depth property is associated with a media object, and how to animate this property (during 5s), enabling an effect of popping-out an image from the screen:

```
<ncl>
...
<body id ="nclBody">
  ...
  <media id="button" src="media/button.png">
    <property name="depth" value="0.0"/>
  </media>
  ...
  <link ...>
    ...
    <bind role="set" component="button" interface="depth">
      <bindParam name="var" value="1.0"/>
      <bindParam name="dur" value="5s"/>
    </ bind>
  </link>
  ...
</body>
</ncl>
```

Although it is not assumed that authors of multimedia applications would change some system configuration-level properties, such as the number of views, it could be interesting to let authors to know system-level information. Thus, the following properties may be accessed considering the rendering system:

ldi.zFar farthest depth plane, which corresponds to the value 0 in **depthMap** or $depth = -1.0$.

ldi.zNear nearest depth plane, which corresponds to the value 255 in the **depthMap** or $depth = 1.0$.

ldi.numViews the number of views that the rendering system produces.

ldi.baselineDistance the baseline distance between two consecutive cameras (or the eye-distance for two views only).

ldi.viewerDistance the user viewing distance.

interleavingMethod informs which method should be used for interleaving the final generated views. Possible values include “side-by-side”, “top-bottom”, “row”, and “column”.

For keeping the system-level properties, NCL has a special node, named settings node, whose properties can be accessed by NCL <link> and <rule> elements.

4.3.2. Imperative Support to Layered-depth Content

Similar to the extensions proposed to the declarative languages, imperative extensions, which allow to control the depth information in canvas-like APIs, are also proposed. The following new methods have been incorporated into NCLua:

canvas:depth(layer: number)->depth: number: returns the depth associated to a layer of the NCLua canvas. The *layer* parameter is an integer. If it is omitted, the first layer is considered, i.e., it is equivalent to **depth(0)**. The returned value is a double in the interval $[-1.0, 1.0]$.

canvas:depth(layer: number, depth: number): sets the depth of a layer of the NCLua canvas. The *layer* parameter is an integer, and *depth* is a double interval $[-1.0, 1.0]$. If the *layer* parameter is not used, the first layer is considered, i.e., it is equivalent to **depth(0)**.

canvas:depthMap(layer: number)->depthMap: canvas: returns the **depthMap** canvas associated to a layer of the NCLua canvas.

canvas:depthMap(layer: number, depth_map: canvas): sets the **depthMap** of a layer of the NCLua canvas. It is possible to create a **depthMap** based on an image by creating a canvas element through “depthMap = canvas:new(image_url)”. That canvas may now be used as the depthMap information of another canvas.

canvas:occlusionLayerTexture(layer: number)->texture: canvas: returns the canvas object that is currently used as the **depthMap** of canvas. It is possible to change the individual pixels of the depthMap

through the native “canvas:pixel” method. If *layer* is not passed as parameter, the first layer is considered.

canvas:occlusionLayerTexture(layer: number, texture: canvas): returns the canvas object that is currently used as the **depthMap** of canvas. It is possible to change the individual pixels of the depthMap through the native “canvas:pixel” method. If *layer* is not passed as parameter, the first layer is considered.

The use of canvas objects for representing *depthMap*, *occlusionLayer*, and *occlusionLayerDepth* of other canvas elements is interesting because these additional data can be dynamically generated. It also enables authors to load external images that represent the depth or occlusion information.

4.4. Layered-depth-aware Graphics Architecture

This section discusses a graphics architecture (schematically shown in Figure 19) that supports the extensions proposed in Section 4.3 and allows for composing 2D and 3D media objects, both synthetic and natural.

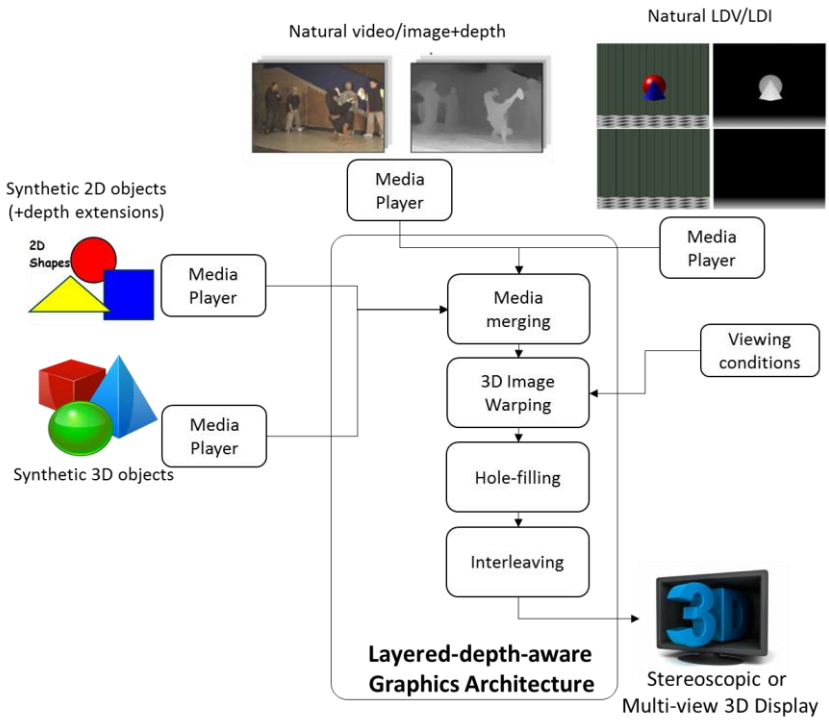


Figure 19 Layered-depth-aware graphics architecture.

The process of rendering multiple views of the multimedia scene works as a loop. In each step, N views, required by multi-view 3D displays, are rendered. The process for one rendering step is discussed in what follows.

Each media object has its own media player that is responsible for generating the texture and, possibly, depth and occlusion information. Synthetic objects must be rendered based on their primitives; natural media must be decoded. Each media player generates a 2D-only texture, which alternatively can have additional depth and occlusion information. In other words, each media player outputs an LDI, which is used as input to the media merging step.

In the *media merging* step, the layered-depth information of each media object that is currently active in the application is received. The component outputs a unique LDI structure for the entire scene. Based on the properties associated with each media object, the layered-depth composition can scale or add some offset to the original depth information of the media objects.

In the *3D image warping* step, the LDI generated by the media merging component is received, as well as, the viewing conditions and the camera configuration parameters. Based on this information, the component outputs two or more views, which can be used by 3D displays.

As will be discussed further, the 3D warping step may produce some holes in the generated views. In comparison to a 3D warping that uses only one video and depth, the use of a layered-depth structure solves most of those holes. Even so, some holes may still exist in the generated views, and the *hole filling* component is responsible for filling them.

Finally, the *interleaving* component is responsible for formatting the generated views for 3D displays. Examples of those formats include top-bottom, side-by-side, and column and row interleaving.

Section 4.5 details the implementation of each component of the layered-depth-aware renderer.

4.5. Implementation

The proposed architecture is implemented in GPGPU using OpenCL (MUNSHI, 2012). In particular, the media merging, 3D warping, hole

filling, and interleaving are instantiated as different OpenCL-coded routines (called kernels). As an example of the use of the architecture, the implementation is currently integrated to the Ginga-NCL reference implementation, allowing the use of NCL as the glue language for layered-depth media. The remainder of this section details the implementation of each component.

4.5.1. Media Merging

The media merging kernel is responsible for generating an LDI data structure that represents the entire scene. As input, it receives the texture(s) and depth(s) information of the media objects that are currently active in the scene. As previously mentioned, individual media players are tasked with the decoding or rendering tasks. The media merging receives that information as OpenGL textures.

In the case of synthetic 2D and 3D media objects, an interesting approach is to use the Framebuffer Object (FBO) to render the objects. FBO allows for the off-screen rendering of an OpenGL context into a color texture and depth buffer. Thus, they can be sent to the media merging step with minor overhead.

A layered-depth image (or LDI) is a bi-dimensional array of layered-depth pixel (LDP). Each layered-depth pixel is an array of *NumLayers* size that keeps depth-pixels (DP). A depth-pixel stores the RGBA color information and the depth information. The following details the LDI data structure (depicted in Figure 20).

```
LayeredDepthImage ={
  Pixels[0..xres-1,0..yres-1]: array of LayeredDepthPixel
}

LayeredDepthPixel = {
  NumLayers: integer
  Layers[0..NumLayers-1]: array of DepthPixel
}

DepthPixel = {
  ColorRGBA: integer
  Z: integer
}
```

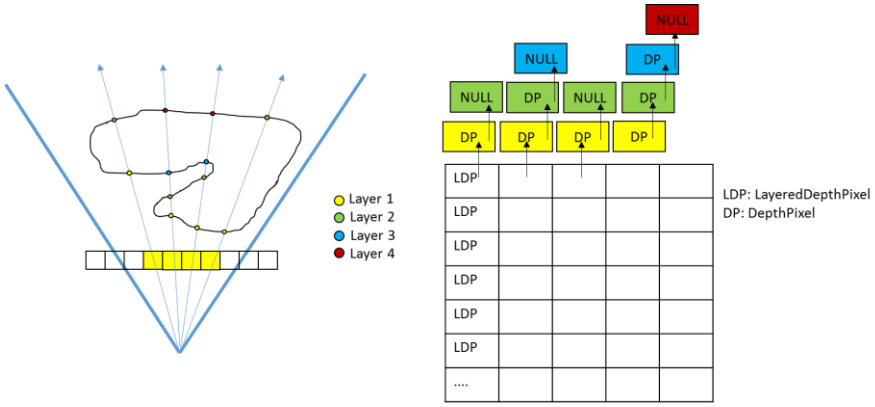


Figure 20 Structure of the LDI.

When composing two depth-pixels that rely on the same position, they must be inserted in a way the layered-pixel data structure remains ordered (from the closest pixel to the farther one).

In theory, the number of layers in an LDI structure may be infinite. In practice, however, it may be useful to limit the number of layers for optimization purposes, allowing a constant rendering time.

4.5.2. 3D Image Warping

The *3D image warping* kernel receives the LDI structure produced by the media merging step and the camera parameters for the output views. It produces the N views required by 3D displays (Figure 21).

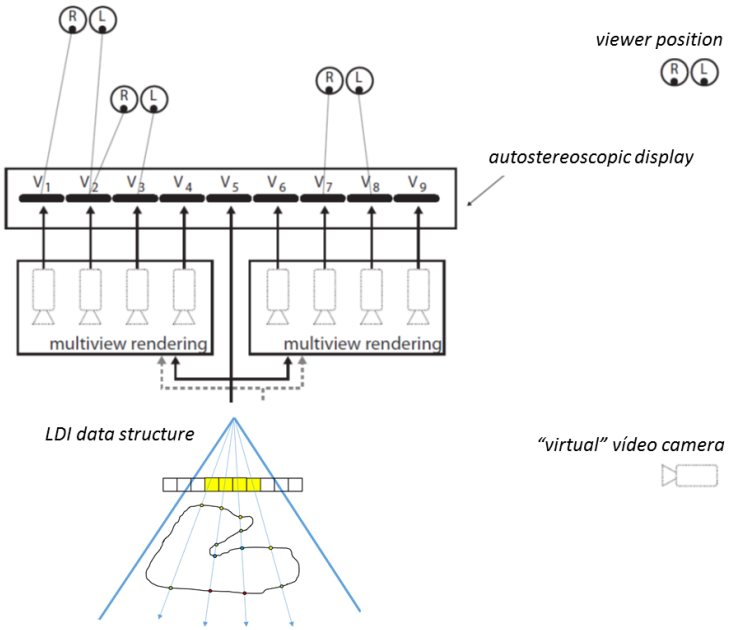


Figure 21 View synthesis for multi-view displays using LDI.

The general "3D image warping" formulation (HARTLEY; ZISSERMAN, 2003) (FEHN, 2004), illustrated in Figure 22, allows us to generate arbitrary views based on a reference color image with corresponding per-pixel depth map and the camera parameters. Conceptually, 3D image warping can be decomposed into two basic steps. First, it back projects the 2D pixels of the reference image plane onto 3D points on the world coordinate system ($\boldsymbol{m} \rightarrow \boldsymbol{m}_w$). Then, it re-projects each 3D point in the world coordinate system onto the target image plane ($\boldsymbol{m}_w \rightarrow \boldsymbol{m}_v$). Each (x, y) coordinate on the reference LDI may contain multiple colors and depths. Therefore, when generating virtual views from the layered-depth information, the 3D warping projection must be performed for every DepthPixel in the LDI to the corresponding target views.

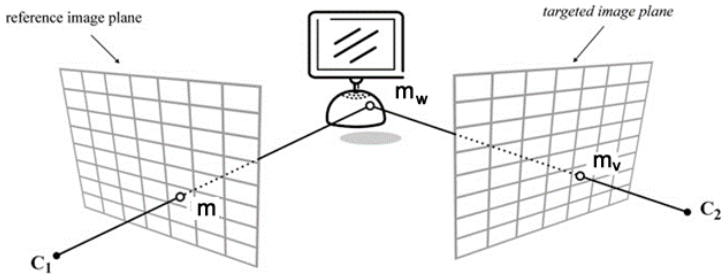


Figure 22 A 3D point $\boldsymbol{m}$ being projected in the reference and targeted image planes. Adapted from (DARIBO, ISMAEL *et al.*, 2013).

In this thesis, the multi-view synthesis is mainly based on a 1D parallel camera configuration. In such configuration, parallaxes only happen in the horizontal axis. More specifically, this thesis is interested in the views configuration of Figure 23.

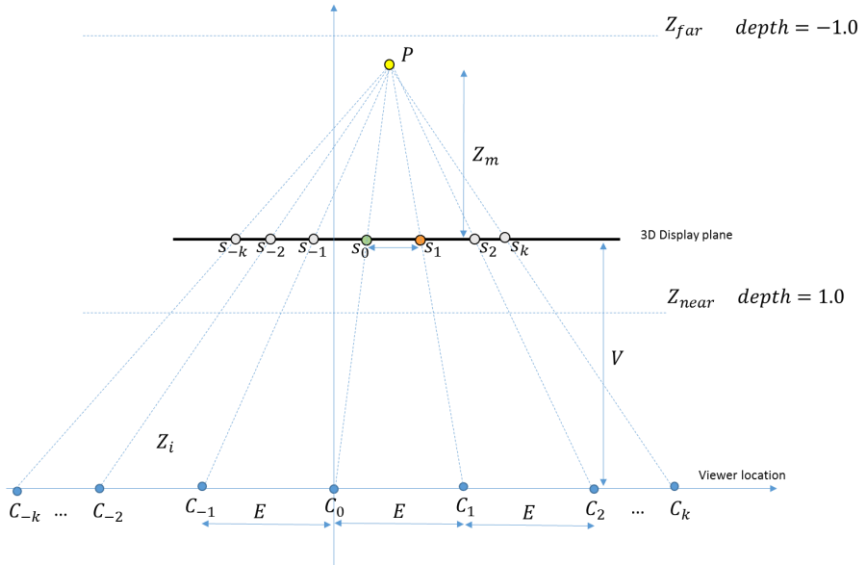


Figure 23 Geometry of a point P to be rendered in relation with viewer distance. Adapted from (HUANG *et al.*, 2010).

The parallax computation is based on the relationship between the depth and pixel location. From triangle similarities in Figure 23, it is possible to compute the final DepthPixel's position in a new view relative to the position of each corresponding DepthPixel in the central view as:

$$s_k = s_0 + k * E * \frac{Z_m}{V + Z_m} \rho \tag{3}$$

where

- s_0 is the pixel position in the center view (i.e. in the LayeredDepthImage);
- k is the view index (e.g. from -4 to 4 for nine views);
- Z_m is the distance from the point P to be rendered to the screen;
- E is the distance between two consecutive views (in a stereoscopic display it can be the interocular distance);
- V is the distance between the viewer and the screen; and

- ρ is the ratio of the screen definition in number of pixels to the width of the display.

To find the Z_m of a media objects (or pixel), the depth values used by multimedia authors (in the range $[-1.0,1.0]$) must be mapped to the $[Z_{min}, Z_{max}]$ range. Z_{min} and Z_{max} parameters can be controlled by the multimedia scene through the extensions defined in Section 4.3.1. The viewer may also control those values to provide customization on the depth perception.

When using gray images (**depthMap**) to represent depth information, each pixel value commonly ranges from 0 to 255. The original depth distance must then be quantized in this range. Instead of a linear quantization, a common approach is to use a non-linear quantization that considers human factors and improves the perceived depth (FAN; CHI, 2008), such as:

$$Z = \frac{1}{\frac{1}{Z_{min}}\left(\frac{Z}{255}\right) + \frac{1}{Z_{max}}\left(1 - \frac{z}{255}\right)}; Z \in [0, \dots, 255] \quad (4)$$

3D Warping in OpenCL

3D warping is pixel-based. Thus, it has the potential to be implemented in parallel architectures. The remainder of this subsection discusses how the 3D warping formulation above is implemented in GPU parallel architectures using OpenCL.

One of the main problems of running the warping algorithm in parallel is that multiple pixels can be mapped to the same final virtual view position. When this happens, it should be assured that only the pixel closest to the viewer will be drawn on the final virtual view. Therefore, the central problem to calculate the final pixel positions in 3D warping parallel implementations is to ensure a thread-safe depth test.

In the special case of parallel camera setups, the pixels will only shift horizontally. As a consequence, only pixels that are in the same line can be mapped to the same final position. Thus, thread-safe depth test may be avoided if only the computation of entire lines are parallelized. This means that each kernel thread execution must be tasked with the calculation of all the pixels in the same line of a

frame. This proposed *per-line parallel* approach has been implemented and tested against the more general *per-pixel parallel* kernel execution.

In the per-pixel parallel model, the 3D warping kernel runs for each LayeredDepthPixel. Each kernel thread execution is responsible for warping all the DepthPixel of a LayeredDepthPixel to their final positions. In this approach, the aforementioned concurrency problems (see Figure 24) may happen. Therefore, a global depth buffer, shared over all computing units of the GPU, is used to ensure a thread-safe depth test.

Figure 24 Example of warping multiple pixels to the same target position without a thread-safe access to the depth buffer (the frame is shown without hole filling).

To solve this problem, we have implemented an approach similar to the one in (GÜNTHER *et al.*, 2013). If the kernel had access and rendered the depth buffer only, atomic operations using *atomic_min* OpenCL function would be enough to guarantee a thread-safe access. However, after depth testing, the access to the color buffer is also needed (GÜNTHER *et al.*, 2013). Since there is no built-in function in OpenCL to guarantee such a thread-safe access to more than one memory position, a semaphore-like synchronization mechanism was implemented and an additional *lock* buffer was added to know if a certain pixel position is locked or not. Before reading or writing to depth or color buffers, the 3D warping kernel must first acquire lock of the corresponding pixel position. The *atomic_cmpxchg* operation was used to implement such a lock feature.

4.5.3.
Hole filling

Theoretically, the 3D warping formulation is able to generate infinitely desirable views, at any position in space. In practice, however, the lack of data or noise in the reference depth information generates a number of artifacts (including holes) in the target image, which annoys viewers (see Figure 25). Those artifacts increase with the distance between the target and original views.

Figure 25 Examples of holes and ghosting artifacts generated by 3D warping.

Compared with traditional 3D warping approaches based on only one texture and depth information, the LDI-based 3D warping solves most of the holes, since it keeps as much occlusion information as possible. Even so, some holes may still be present in the generated views. The *hole filling* kernel is responsible for filling those holes.

Various techniques for reducing the generation of artifacts in the generated views have been proposed. Some of them take place before the 3D warping and focus on pre-processing the depth-map aiming at reducing the large number of dis-occlusions. As an example, some work use Gaussian filters to pre-process the depth map to remove the holes generated by 3D warping (TAM *et al.*, 2004). Such approach, however, has some drawbacks, such as creating non-natural geometric distortions in the generated virtual views. Other proposed techniques include: asymmetric Gaussian filter (FEHN, 2003a), bilateral filters (DARIBO, ISMAËL;

SAITO, 2010); and adaptive filters (KOPPEL; BEN MAKHLOUF; NDJIKI-NYA, 2013).

Other approaches take place after 3D warping and focus on filling the generated holes by using neighboring pixel information. Many techniques have been proposed following this approach, such as nearest neighbor, simple interpolation, depth-aided interpolation, in-painting, and depth-aided in-painting (VÁZQUEZ; TAM; SPERANZA, 2006). Some of them are suitable to be implemented in parallel architectures, such as the nearest neighbor, and the simple interpolation. However, more complex approaches, such as those based on in-painting techniques (GUILLEMOT; LE MEUR, 2014), are not suitable for parallel or real-time implementations.

Concerning the final visual quality, in-painting techniques usually perform better. Subjective tests comparing hole filling techniques have been carried out by Vázquez *et al.* (VÁZQUEZ; TAM; SPERANZA, 2006) and Azzari *et al.* (AZZARI; BATTISTI; GOTCHEV, 2010). Vázquez *et al.* endorse that "hole filling using the background pixels rather than the foreground ones as the dis-occluded areas is more reasonable by the definition of dis-occlusion". Based on this assumption, the hole filling implemented in this thesis takes into account the depth of neighbor pixels for hole filling.

In each generated view, the hole filling kernel searches for colored pixels in the hole neighborhood but that are in the same line of the hole. It then selects the ones with the lowest depth values, *i.e.*, the background pixels, and fills the hole with a weighted average of those background neighborhood pixels. The weight is based on the distance from the pixel to the hole. This approach takes into account that pixels belonging to the background are probably the most correct ones to fill dis-occlusion holes. Figure 26 shows an example of a video-only frame before and after the hole filling kernel execution.

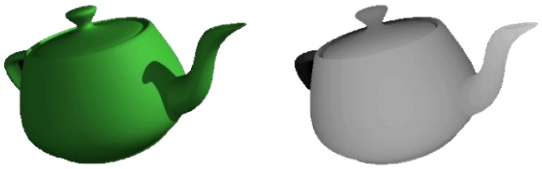
Figure 26 Example of a frame before (top) and after (bottom) hole filling kernel execution.

4.6. Experimental Results

Two approaches are used to evaluate the layered-depth graphics architecture implementation. First, an example of the rendering results when using the layered-depth for the graphical composition of a simple multimedia scene is presented. The scene is composed of an LDV-based video and a 3D model. The rendering result is compared with a graphical composition using a 2D+depth only systems. Second, the frame rates of rendering the simple multimedia scene with the layered-depth rendered are discussed.

The multimedia scene used in the remainder of this section is based on the media objects of Figure 27. Figure 27 shows two media objects: a natural 2D+Z media object, and a synthetic 3D media object (which, when rendered, generates the texture and depth map presented in Figure 27(b)).

(a)



(b)

Figure 27 Example a (a) video+depth and a (b) the 2D+depth data produced by rendering a synthetic 3D object.

Compared with a DIBR system based on a 2D+depth-only representation of the scene—such as the one proposed by (LE FEUVRE; MATHIEU, 2013)—the advantage of the layered-depth renderer is straightforward: it keeps as much occlusion information as possible. Thus, it helps the process of rendering multiples views of the scene. To illustrate such advantage, Figure 28 illustrates the two layers of the LDI-based composition between the two media objects of Figure 27.

Layer (1)

Layer (2)

Figure 28 The two layers (texture and depth) of the graphical composition between the media objects of Figure 27 using the LDI graphics architecture.

After creating the LDI structure (depicted in Figure 28), the 3D warping kernel is responsible for generating the views required by 3D displays. To compare the rendering results between a DIBR approach using only a 2D+Z representation (i.e., the first layer only) and the proposed LDI approach, Figure 29 shows the rendering result of a virtual view.

Figure 29(a) shows the rendering of a virtual view using only the first layer of the LDI and without hole filling.

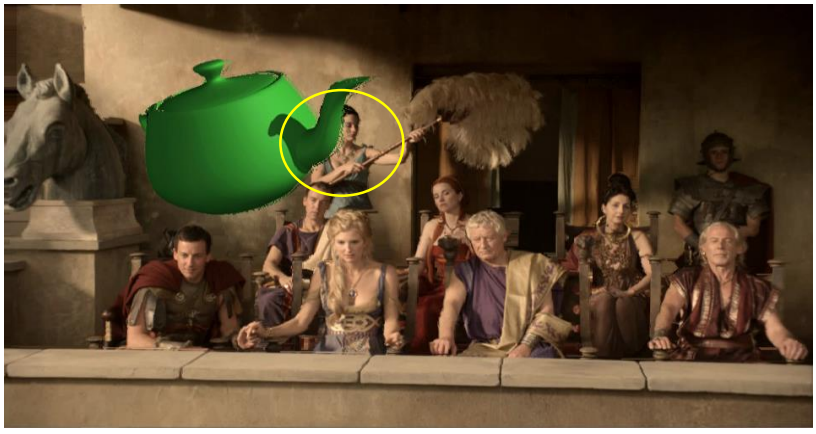
Figure 29(b) shows the rendering result after a simple hole filling that can be implemented in real-time (a horizontal depth-based extrapolation). More advanced hole filling approaches may achieve better result than the one of Figure 29(b), but it is hard to believe that they can recovery the girl's face based only on one 2D+Z data.

Figure 29(c) shows the rendering result of using the layered-depth rendered proposed in this chapter. In it, it is possible to see the girl's face in the background. In the images, some crack in the objects' border is still present. This is mainly because, in the time of this writing, the current implementation does not handle

those borders satisfactorily. Nevertheless, they can be removed by detecting (and not warping) the pixels on the objects' border or by a pre-filtering the depth-map.

(a)

(b)



(c)

Figure 29 Example of a composition between a video+depth and a synthetic 3D object.

Compared with the 2D+Z-only approach, however, the disadvantage of the LDI approach is that there are more pixels (now DepthPixels) to be warped to the final views. The rendering time then increases with the depth complexity of the scene. Nevertheless, constant timing virtual views rendering can still be supported by limiting the number of occlusion layers (which may interesting for many applications). Moreover, the LDI graphics architecture is implemented using OpenCL aiming at taking advantage of high-parallelism of GPUs and achieving real-time performance (at least 30fps).

Table 1 shows the results of the frame rates execution (the average value after 100 executions) of the scene in a laptop computer with a Core i7 CPU, 8 GB of RAM, and an Nvidia GeForce GT 740M GPU. For comparison purposes, the table shows the average frame per-second achieved by rendering the scene following the per-line parallel and the per-pixel parallel execution, as discussed in the previous section. The final image resolution is the composition of all the views of the scene (in the case of the measurements, there are five views horizontally multiplexed).

Media de-coding (for natural objects) or rendering (for synthetic media objects) is not taken into account mainly because our main goal is to measure the performance of the LDI compositor and the multi-view generator.

Resolution	**Per-line parallel (fps)**	**Per-pixel parallel (fps)**
1280x720 (HD)	35.50	66.73
2560x720 (Stereo HD)	18.35	37.42
1920x1080 (Full-HD)	**17.90**	**32.28**
3840x1080 (Stereo Full-HD)	9.66	17.72

Table 1 Layered-depth renderer frame rates when rendering the scene of Figure 28 using the two proposed parallel methods: per-line parallel and per-pixel parallel.

As can be noticed in Table 1, the OpenCL-based layered-depth renderer implementation is able to perform in real-time for full-HD video, and almost in real-time performance for side-by-side stereoscopic full-HD (2x horizontal resolution of full-HD), when running in the per-pixel parallel execution kernel. The per-line parallel execution kernel has the advantage of not explicitly requiring concurrent

synchronization mechanisms. However, it runs slower than the per-pixel parallel execution kernel. An explanation for the slower performance of the per-line parallel execution kernel is that the higher granularity of the data partition fails to fully exploit the parallel capabilities of the GPU configuration.

4.7. Concluding Remarks

This chapter proposed extensions to multimedia languages to support a layered-depth scene model and a graphics architecture, aimed at being implemented by high-end receivers, to support such extensions. The proposed extensions allow to develop multimedia applications merging synthetic and natural media objects. The proposed architecture for composing different media objects and rendering multiple views for 3D displays is implemented using OpenCL for achieving real-time rendering. Experimental results related to the rendering quality and performance have shown the viability of the proposal.

One of the advantages of using an image-based (2D+Z or LDV) representation of the multimedia scene, when compared to a polygon-based, is to support a constant-time rendering of the scene, which can be useful for supporting real-time systems based on high-quality video. In the LDI case, it is possible to support constant time multi-view rendering by limiting the number of layers.

The drawback of any DIBR approach (including the one presented here) is that there is a limitation on the distance between the original camera position and the virtual views in which the generated holes do not affect the QoE. The approach proposed in the chapter allows to improve the rendering results by taking advantage of the occlusion information provided by the LDI structure.

5
Conclusion

5.1.
Summary of the Contributions

In the Introduction chapter of this thesis (Section 1.2) we have defined the main research questions we have aimed to answer with this work. Based on those research questions, this thesis presents two approaches to support multimedia applications in 3DV-based delivery chain scenarios. The contributions of the work are related to those approaches.

Stereoscopic Multimedia Applications

The first proposed approach (Chapter 3) is related to and provide a solution that answer the RQ.2. and RQ.3. The proposed approach extends 3DV delivery chains based on CSV format through stereoscopic multimedia applications. The support to stereoscopic multimedia applications is introduced by depth annotations on media objects and a document converter. The document converter takes multimedia applications with depth annotations as input and outputs corresponding stereoscopic multimedia applications. The conversion process is backward compatible with current CSV-based delivery chains, low-end digital receivers, and 2D-only multimedia language players. Experimental results have shown that the generated applications have good enough QoE to deploy interactive-stereoscopic services in current CSV-based delivery chains.

In short, the main contributions of the first part of the thesis are the concept of stereoscopic multimedia applications and the process to supporting them at the application-level. The proposed stereoscopic multimedia applications have the potential to provide new user interfaces for interactive content in 3DV delivery chains, such as Interactive 3DTV, Interactive 3D Cinema, and 3D Teleconference. Those applications can engage users in different ways, providing realistic S3D media objects content or additional mechanism for reinforcing elements in the user interface.

As additional contributions, the proposed extensions and the conversion process have been instantiated to the NCL multimedia language, resulting in two open source tools (NCLSC and NCLuaCanvaS3D). NCLSC is the tool to convert an NCL document annotated with the proposed extensions into its stereoscopic version. A first version of NCLSC was reported in (AZEVEDO; LIMA; SOARES, 2015). That version has evolved to the one presented in Chapter 3. NCLuaCanvaS3D allows developing CSV-based content based on Lua, the scripting language of NCL. NCL is the declarative multimedia language standardized by the Brazilian Digital Terrestrial TV System, which has also been adopted by most of the Latin America countries. Therefore, the discussed tools constitute a framework for broadcasters to provide interactive stereoscopic services in today NCL-based digital TV systems.

Appendix B discusses another additional contribution, presenting depth-based layout templates, which can simplify the author's work when defining complex stereoscopic multimedia applications.

Although it is possible to envision that in the future most of the digital receivers will be high-end ones, with native support for 3D synthetic objects and stereoscopic effects, until now most of them are still low-end devices (even those with S3D displays). For instance, TV devices with support for OpenGL have only appeared in the last year (PERAKAKIS; GHINEA, 2015). Therefore, the stereoscopic multimedia application proposed solution allows developers to reach 3DTV users today. They can also support high-quality CSV-based media content, which even by using high-end devices may not be feasible to render at the client side in real time.

Layered-depth-aware Multimedia Applications

The second proposed approach (Chapter 4) is related to and provide answers to RQ.4 and RQ.5. The proposed solution considers a 3DV delivery chain based on depth information—a 3DV representation that allows advanced features such as client-side adaptation of depth perception and multi-view displays support. In this scenario, layered-depth-aware multimedia applications are proposed. Extensions to multimedia languages allowing authors to control the per-pixel graphical composition of media objects are discussed. A graphics architecture supporting the

rendering of multiple views for layered-depth-aware multimedia applications is also discussed.

In summary, the main contributions of the second part of the thesis are the proposed extensions and the graphics architecture for rendering high-quality virtual views while taking advantage of the good performance provided by DIBR methods. The use of a layered-depth graphical composition allows keeping as much occlusion information as possible during the graphics composition process, which simplifies the multi-view rendering and hole filling processes. When compared to related work, the layered-depth approach allows one to get better virtual views rendering quality.

Layered-depth multimedia applications can also be used to provide advanced user interfaces taking advantage of the real depth information and new 3D displays. An example of such applications is augmented reality applications that can seamlessly integrate natural and synthetic media objects.

As in the stereoscopic multimedia applications, the layered-depth multimedia application proposal has also been integrated and tested in 3DTV delivery chains based on ISDB-T and its declarative middleware Ginga-NCL. Future versions of ISDB-T/Ginga-NCL targeting advanced 3DV services can then take advantage of the proposals herein.

A first integration of 3D objects (simple and composite ones) in NCL documents was reported in (AZEVEDO; SOARES, 2012). Despite focusing mainly on controlling the behavior of the embedded 3D objects, the reported implementation was used as a starting pointing for the layered-depth renderer proposed in this thesis. Preliminary results on 2D+depth media extensions for NCL were discussed in (AZEVEDO; SOARES, 2013). The carried out implementation transformed the texture+depth data of media objects into a mesh of triangles to rendering stereoscopic views based on OpenGL. It did not provide a real-time rendering for high-definition images and video media objects. However, the experience of this first implementation has allowed for finding the main problems related with transforming the image+depth data into traditional meshes, and to further advance with a DIBR and a layered-depth solution. In addition, based on the implementation, a complete end-to-end interactive 3DTV delivery chain based on video+depth was presented in (AZEVEDO; SOARES, 2014). An instantiation of the proposed chain for ISDB-T and NCL was also discussed, in which interactive

content supporting the extensions defined are supported. The first implementation of the OpenCL-based DIBR process (AZEVEDO *et al.*, 2014) has evolved into the layered-depth architecture discussed in Chapter 4. It was a real-time (performance of at least 30fps) depth-image-based rendering for only video+depth representation, not ready to compose different media objects. The architecture of Chapter 4 allows for composing different media objects and the generation of multiple views.

5.2. Future Research

Viewing stereoscopic 3D content through 3D displays is different from real-world viewing. Notably, the natural synchronization between accommodation and vergence (see Section A.1 for details) is lost when seeing 3D content in S3D displays. Even with perfect S3D content, some QoE issues may arise due to different user's preferences and viewing conditions. In particular, for stereoscopic multimedia applications, the first part of the thesis provides means to customize depth perception and discusses experimental results evidencing that de-synchronization issues do not significantly affect the QoE perceived by users. However, assessing the final QoE through subjective experiments is an important future work. Moreover, the development of meaningful S3D applications content, which improve QoE when compared to 2D-only content, is still an important issue to be researched in stereoscopic 3D presentations.

Concerning the layered-depth multimedia applications, future work includes the use of GPU to decode color and depth frames, and the evolution of the system to support FTV applications. In this case, arbitrary (possibly with movement) camera configurations must be supported. When using arbitrary camera configurations, mainly the ones with wide baseline, more advanced hole filling approaches, possibly based on in-painting techniques, will be necessary. Therefore, how to efficiently map such algorithms, which are not easily parallelizable, to the GPGPU must also be handled. Moreover, the extension of the layered-depth approach for new super-multi-view displays, which support hundreds of views, is also another interesting future work. In such a case, the extension of the layered-depth architecture to run on multiple GPUs may be needed. As in the stereoscopic multimedia application, the QoE assessment by final users using layered-depth

multimedia application and multi-view 3D displays is also an important future work.

As a whole, the proposals of this work can be seen as part of an active and exciting 3D Media research field that includes all phases in a 3DV delivery chain: capturing, coding, transmission, decoding, rendering, and display. Even though all these phases have been researched and many important advancements have been achieved, there are still many open issues that should be fulfilled to achieve successful high-quality end-to-end 3DV systems.

6
References

AMORIM, Glauco Fiorott; DOS SANTOS, Joel André Ferreira; MUCHALUAT-SAADE, Débora Christina. Adaptive layouts for authoring NCL programs. In: 19TH BRAZILIAN SYMPOSIUM ON MULTIMEDIA AND THE WEB, WebMedia '13, 2013, [S.l.]: ACM Press, 2013. p. 205–208. Disponível em: <http://dl.acm.org/citation.cfm?doid=2526188.2526234>. Acesso em: 9 set. 2014.

APPLE. *The WebKit open source project*. Disponível em: <http://www.webkit.org>. Acesso em: 4 dez. 2015.

ATSC. *ATSC Standard: 3D-TV Terrestrial Broadcasting, Part 1.* , nº ATSC A/104 Part 1:2014. [S.l.]: Advanced Television Systems Committee, 4 ago. 2014.

Awwwards Team: 10 stereoscopic 3d websites. Disponível em: <http://www.awwwards.com/10-stereoscopic-3d-websites.html>. Acesso em: 4 dez. 2014.

AZEVEDO, Roberto Gerson de Albuquerque *et al*. Real-time Depth-Image-Based Rendering for 3DTV using OpenCL. In: INTERNATIONAL SYMPOSIUM ON VISUAL COMPUTING, dez. 2014, Las Vegas. *Anais*... Las Vegas: [s.n.], dez. 2014.

AZEVEDO, Roberto Gerson de Albuquerque. *Suporte ao controle e à apresentação de objetos de mídia tridimensionais em NCL*. 2010. Master Thesis – Rio de Janeiro, 2010. Disponível em: <http://www2.dbd.puc-rio.br/pergamum/biblioteca/php/mostrateses.php?open=1&arqtese=0821387_10_Indice.html>.

AZEVEDO, Roberto Gerson de Albuquerque; LIMA, Guilherme; SOARES, Luiz Fernando Gomes. An Approach to Convert NCL Applications into Stereoscopic 3D. In: DOCENG, 2015, Lausanne, Switzerland. *Anais*... Lausanne, Switzerland: [s.n.], 2015.

AZEVEDO, Roberto Gerson de Albuquerque; SOARES, Luiz Fernando Gomes. Embedding 3D Objects into NCL Multimedia Presentations. Web3D '12, 2012, New York, NY, USA. *Anais*... New York, NY, USA: ACM, 2012. p. 143–151. Disponível em: <http://doi.acm.org/10.1145/2338714.2338739>.

AZEVEDO, Roberto Gerson de Albuquerque; SOARES, Luiz Fernando Gomes. Ginga extensions to support depth-based 3D media. jul. 2014, [S.l: s.n.], jul. 2014. p. 1–4.

AZEVEDO, Roberto Gerson de Albuquerque; SOARES, Luiz Fernando Gomes. NCL+Depth: Extending NCL for Stereo/Autostereoscopic 3D Displays.

WebMedia '13, 2013, New York, NY, USA. *Anais*... New York, NY, USA: ACM, 2013. p. 185–192. Disponível em: <http://doi.acm.org/10.1145/2526188.2526203>.

AZZARI, Lucio; BATTISTI, Federica; GOTCHEV, Atanas. Comparative analysis of occlusion-filling techniques in depth image-based rendering for 3D videos. 2010, [S.l.]: ACM, 2010. p. 57–62. Disponível em: <http://dl.acm.org/citation.cfm?id=1878037>. Acesso em: 9 abr. 2014.

BADROS, Greg J. *et al.* A constraint extension to scalable vector graphics. In: 10TH INTERNATIONAL CONFERENCE ON WORLD WIDE WEB, WWW '01, 2001, [S.l.]: ACM Press, 2001. p. 489–498. Disponível em: <http://portal.acm.org/citation.cfm?doid=371920.372146>. Acesso em: 10 set. 2014.

BARENBRUG, B. Declipse 2: multi-layer image and depth with transparency made practical. 5 fev. 2009, [S.l: s.n.], 5 fev. 2009. p. 72371G. Disponível em: <http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.806819>. Acesso em: 29 maio 2015.

BARTCZAK, B *et al.* Display-Independent 3D-TV Production and Delivery Using the Layered Depth Video Format. *IEEE Transactions on Broadcasting*, v. 57, n. 2, p. 477–490, jun. 2011.

BEHR, J. *et al.* A Scalable Architecture for the HTML5/X3D Integration Model X3DOM. Web3D '10, 2010, New York, NY, USA. *Anais*... New York, NY, USA: ACM, 2010. p. 185–194. Disponível em: <http://doi.acm.org/10.1145/1836049.1836077>.

BEHR, Johannes *et al.* Dynamic and Interactive Aspects of X3DOM. Web3D '11, 2011, New York, NY, USA. *Anais*... New York, NY, USA: ACM, 2011. p. 81–87. Disponível em: <http://doi.acm.org/10.1145/2010425.2010440>.

BEHR, Johannes *et al.* X3DOM: A DOM-based HTML5/X3D Integration Model. Web3D '09, 2009, New York, NY, USA. *Anais*... New York, NY, USA: ACM, 2009. p. 127–135. Disponível em: <http://doi.acm.org/10.1145/1559764.1559784>.

BENZIE, P. *et al.* A Survey of 3DTV Displays: Techniques and Technologies. *IEEE Transactions on Circuits and Systems for Video Technology*, v. 17, n. 11, p. 1647–1658, nov. 2007.

BORNING, Alan; LIN, Richard Kuang-Hsu; MARRIOTT, Kim. Constraint-based document layout for the Web. *Multimedia Systems*, v. 8, n. 3, p. 177–189, 1 out. 2000.

BOURGE, Arnaud; GOBERT, Jean; BRULS, Fons. MPEG-C part 3: Enabling the introduction of video plus depth contents. 2006, [S.l: s.n.], 2006.

BRUTZMAN, Don. *X3D: extensible 3D graphics for Web authors*. Amsterdam ; Boston: Elsevier, 2007.

BULTERMAN, Dick *et al. Synchronized Multimedia Integration Language (SMIL 3.0)*. W3C Recommendation. [S.l.]: W3C, dez. 2008. Disponível em: <http://www.w3.org/TR/SMIL3/>. Acesso em: 30 jul. 2015.

CABANIER, Rik *et al. HTML Canvas 2D Context*. W3C Candidate Recommendation. [S.l.]: W3C, 2 jul. 2015.

CAGNAZZO, Marco; PESQUET-POPESCU, Béatrice; DUFAUX, Frédéric. 3D Video Representation and Formats. In: DUFAUX, FRÉDÉRIC; PESQUET-POPESCU, BÉATRICE; CAGNAZZO, MARCO (Org.). . *Emerging Technologies for 3D Video*. Chichester, UK: John Wiley & Sons, Ltd, 2013. p. 102–120. Disponível em: <http://doi.wiley.com/10.1002/9781118583593.ch6>. Acesso em: 8 jun. 2015.

CESAR, Pablo. *A graphics software architecture for high-end interactive TV terminals*. 2005. Helsinki University of Technology, Espoo, 2005.

CESAR, Pablo; VUORIMAA, Petri; VIERINEN, Juha. A graphics architecture for high-end interactive television terminals. *ACM Transactions on Multimedia Computing, Communications, and Applications*, v. 2, n. 4, p. 343–357, 1 nov. 2006.

CHENG, Xiaoyu; SUN, Lifeng; YANG, Shiqiang. Generation of Layered Depth Images from Multi-View Video. 2007, [S.l.]: IEEE, 2007. p. V – 225–V – 228. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4379806>. Acesso em: 25 maio 2015.

CHEN, Qinshui *et al.* An approach to support stereoscopic 3D web. In: ANNUAL ACM SYMPOSIUM ON APPLIED COMPUTING, SAC'13, 2014, [S.l.]: ACM Press, 2014. p. 981–984. Disponível em: <http://dl.acm.org/citation.cfm?doid=2554850.2555096>. Acesso em: 8 set. 2014.

CHEN, Qinshui; WANG, Wenmin; WANG, Ronggang. The rendering context for stereoscopic 3D web. 6 mar. 2014, [S.l: s.n.], 6 mar. 2014. p. 90111P. Disponível em: <http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2038828>. Acesso em: 29 jul. 2014.

CHEN, Ying *et al.* Overview of the MVC+D 3D video coding standard. *Journal of Visual Communication and Image Representation*, v. 25, n. 4, p. 679–688, maio 2014.

CHEN, Ying; VETRO, Anthony. Next-Generation 3D Formats with Depth Map Support. *MultiMedia, IEEE*, v. 21, n. 2, p. 90–94, abr. 2014.

CHISTYAKOV, Alexey; GONZÁLEZ-ZÚÑIGA, Diego; CARRABINA, Jordi. Bringing the Web Closer: Stereoscopic 3D Web Conversion. *Human Computer Interaction*. [S.l.]: Springer, 2013. p. 22–25. Disponível em: <http://link.springer.com/chapter/10.1007/978-3-319-03068-5_5>. Acesso em: 24 jul. 2014.

CLARK, James H. Hierarchical Geometric Models for Visible Surface Algorithms. *Commun. ACM*, v. 19, n. 10, p. 547–554, out. 1976.

COCKBURN, Andy; MCKENZIE, Bruce. Evaluating the effectiveness of spatial memory in 2D and 3D physical and virtual environments. In: SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, CHI '02, 2002, [S.l.]: ACM Press, 2002. p. 203–210. Disponível em: <http://portal.acm.org/citation.cfm?doid=503376.503413>. Acesso em: 4 ago. 2014.

CUTTING, James E.; VISHTON, Peter M. Perceiving Layout and Knowing Distances. *Perception of Space and Motion*. [S.l.]: Elsevier, 1995. p. 69–117. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/B9780122405303500055>. Acesso em: 17 jun. 2015.

DANE, Gökçe; BHASKARAN, Vasudev. Multiview synthesis for autostereoscopic displays. 26 set. 2013, [S.l: s.n.], 26 set. 2013. p. 885610. Disponível em: <http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2026554>. Acesso em: 20 jun. 2014.

DARIBO, Ismael *et al.* Hole Filling for View Synthesis. In: ZHU, CE *et al.* (Org.). . *3D-TV System with Depth-Image-Based Rendering*. [S.l.]: Springer New York, 2013. p. 169–189. Disponível em: <http://dx.doi.org/10.1007/978-1-4419-9964-1_6>.

DARIBO, Ismaël; SAITO, Hideo. Bilateral depth-discontinuity filter for novel view synthesis. 2010, [S.l.]: IEEE, 2010. p. 145–149. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5662009>. Acesso em: 23 maio 2014.

DODGSON, Neil A. Analysis of the viewing zone of multiview autostereoscopic displays. 24 maio 2002, [S.l: s.n.], 24 maio 2002. p. 254–265. Disponível em: <http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=875998>. Acesso em: 11 ago. 2015.

DODGSON, NEIL A. Autostereoscopic 3D Displays. 2005.

DOS SANTOS, JoelAndréFerreira; MUCHALUAT-SAADE, DéboraChristina. XTemplate 3.0: spatio-temporal semantics and structure reuse for hypermedia compositions. *Multimedia Tools and Applications*, v. 61, n. 3, p. 645–673, 2012.

DUFOURD, Jean-Claude; AVARO, Olivier; CONCOLATO, Cyril. An MPEG standard for rich media services. *IEEE MultiMedia*, v. 12, n. 4, p. 60–68, 2005.

DVB. *Plano-stereoscopic 3DTV; Digital Video Broadcasting (DVB); Plano-stereoscopic 3DTV; Part 1: Overview of the multipart*. , nº DVB Document A154-1. [S.l.]: Digital Video Broadcasting (DVB), mar. 2015. Disponível em: <http://www.etsi.org/deliver/etsi_ts/101500_101599/10154701/01.01.01_60/ts_10154701v010101p.pdf>.

ETEMAD, Elika; JR, Tab Atkins; ATANASSOV, Rossen. *CSS Grid Layout Module Level 1*. W3C Working Draft. [S.l.]: W3C, maio 2014. Disponível em: <http://www.w3.org/TR/2014/WD-css-grid-1-20140513/>.

FAN, Yu-Cheng; CHI, Tsung-Chen. The Novel Non-Hole-Filling Approach of Depth Image Based Rendering. maio 2008, [S.l.]: IEEE, maio 2008. p. 325–328. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4547874>. Acesso em: 18 ago. 2014.

FEHN, Christoph. 3D TV broadcasting. *Fraunhofer Institute for Telecommunications, Berlin, Germany, Information Society Technologies, Proposal No. IST-2001-34396*, 2005. Disponível em: <http://books.google.com/books?hl=en&lr=&id=fdrTq7cWxmwC&oi=fnd&pg=PA23&dq=%22concept+of+an+end-to-end+stereoscopic+video+chain.+Thereafter,+an%22+%22HISTORY+OF+3D+TV%22+%22the+advent+of+motion+pictures,+the+popularity+of+the+stereoscope+began+to%22+&ots=1lfaRSGnrF&sig=c8k-QYGBdCxzKMYVWQlon-xz5bY>. Acesso em: 19 mar. 2015.

FEHN, Christoph. A 3D-TV approach using depth-image-based rendering (DIBR). 2003a, [S.l: s.n.], 2003. p. 482–487. Disponível em: <http://www.actapress.com/PaperInfo.aspx?PaperID=14373>. Acesso em: 9 abr. 2014.

FEHN, Christoph. A 3D-TV system based on video plus depth information. 2003b, [S.l.]: IEEE, 2003. p. 1529–1533. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1292241>. Acesso em: 18 jul. 2014.

FEHN, Christoph. Depth-image-based rendering (DIBR), compression, and transmission for a new approach on 3D-TV. *Proc. SPIE*, v. 5291, p. 93–104, 21 maio 2004.

FUCHS, Henry; STATE, Andrei; BAZIN, Jean-Charles. Immersive 3D Telepresence. *Computer*, v. 47, n. 7, p. 46–52, jul. 2014.

GENG, Jason. Three-dimensional display technologies. *Advances in Optics and Photonics*, v. 5, n. 4, p. 456, 31 dez. 2013.

GOLDMANN, Lutz; LEE, Jong-Seok; EBRAHIMI, Touradj. Temporal synchronization in stereoscopic video: Influence on quality of experience and automatic asynchrony detection. set. 2010, [S.l.]: IEEE, set. 2010. p. 3241–3244. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5651142>. Acesso em: 20 jun. 2014.

GOLDSTEIN, E. Bruce. *Sensation and perception*. Ninth edition ed. Belmont, CA: Wadsworth, Cengage Learning, 2014.

GOTCHEV, A *et al.* Three-Dimensional Media for Mobile Devices. *Proceedings of the IEEE*, v. 99, n. 4, p. 708–741, abr. 2011.

GRAY, Kris. *Microsoft DirectX 9 programmable graphics pipeline*. [S.l.]: Microsoft Pr, 2003.

GUILLEMOT, Christine; LE MEUR, Olivier. Image Inpainting : Overview and Recent Advances. *IEEE Signal Processing Magazine*, v. 31, n. 1, p. 127–144, jan. 2014.

GÜNTHER, Christian *et al.* A GPGPU-based Pipeline for Accelerated Rendering of Point Clouds. *Journal of WSCG*, v. 21, n. 2, p. 153–162, 2013.

HANG, Soonbo; LEE, Dong-Yong. *Extensions for Stereoscopic 3D support*. . [S.l: s.n.], 30 nov. 2012. Disponível em: <http://www.w3.org/2011/webtv/3dweb/3dweb_proposal_121130.html>. Acesso em: 27 set. 2014.

HARTLEY, Richard; ZISSERMAN, Andrew. *Multiple view geometry in computer vision*. Cambridge, UK; New York: Cambridge University Press, 2003. Disponível em: <http://dx.doi.org/10.1017/CBO9780511811685>. Acesso em: 24 jan. 2015.

HÉGARET, Philippe Le *et al. Document Object Model (DOM) Level 3 Core Specification*. W3C Recommendation. [S.l.]: W3C, abr. 2004.

HONG, Jisoo *et al.* Three-dimensional display technologies of recent interest: principles, status, and issues [Invited]. *Applied Optics*, v. 50, n. 34, p. H87, 1 dez. 2011.

HOWARD, Ian P.; ROGERS, Brian J. *Binocular Vision and Stereopsis*. [S.l.]: Oxford University Press, 1996. Disponível em: <http://www.oxfordscholarship.com/view/10.1093/acprof:oso/9780195084764.001.0001/acprof-9780195084764>. Acesso em: 11 ago. 2014.

HO, Yo-Sung; OH, Kwan-Jung. Overview of multi-view video coding. 2007, [S.l.]: IEEE, 2007. p. 5–12. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4381085>. Acesso em: 9 abr. 2014.

HTML5. . [S.l.]: W3C, 28 out. 2014. Disponível em: <http://www.w3.org/TR/html5/>. Acesso em: 11 nov. 2014.

HUANG, Wei-Hao *et al.* Real-time novel rendering architecture for 3D display. *Proc. of Electronics and Optoelectronics Research Laboratories*, 2010. Disponível em: <http://cgit.nutn.edu.tw:8080/cgit/PaperDL/WSY_100930054957.PDF>. Acesso em: 17 abr. 2014.

IJSSELSTEIJN, Wijnand A.; SEUNTIENS, Pieter .J. H.; MEESTERS, Lydia M. J. Human Factors of 3D Displays. *3D Videocommunications*. [S.l: s.n.], 2001. .

ISO. *ISO IEC 23002-3 - Information technology — MPEG video technologies — Representation of auxiliary video and supplemental information*. , nº ISO IEC 23002-3. [S.l: s.n.], 2007.

ISO. *The Virtual Reality Modeling Language Specification. Version 2.0*. , nº ISO/IEC WD 14772-1:1997. [S.l.]: ISO/IEC, 1997.

JACOBS, Charles *et al.* Adaptive document layout. *Communications of the ACM*, v. 47, n. 8, p. 60, 1 ago. 2004.

JANKOWSKI, Jacek *et al.* Declarative integration of interactive 3D graphics into the world-wide web: principles, current approaches, and research agenda. 2013, [S.l.]: ACM, 2013. p. 39–45. Disponível em: <http://dl.acm.org/citation.cfm?id=2466547>. Acesso em: 16 abr. 2014.

JUMISKO-PYYKKÖ, Satu; WEITZEL, Mandy; STROHMEIER, Dominik. Designing for user experience: what to expect from mobile 3D TV and video? 2008, [S.l.]: ACM, 2008. p. 183–192. Disponível em: <http://dl.acm.org/citation.cfm?id=1453841>. Acesso em: 8 maio 2014.

JUNG, Kwanghee *et al.* 2D/3D Mixed Service in T-DMB System Using Depth Image Based Rendering. fev. 2008, [S.l: s.n.], fev. 2008. p. 1868–1871.

KIM, Nam *et al.* 3D Display Technology. *Display and Imaging*, v. 1, p. 73–95, 2013.

KONDOZ, Ahmet; DAGIUKLAS, Tasos (Org.). *3D Future Internet Media*. New York, NY: Springer New York, 2014. Disponível em: <http://link.springer.com/10.1007/978-1-4614-8373-1>. Acesso em: 23 abr. 2014.

KOPPEL, Martin; BEN MAKHLOUF, Mehdi; NDJIKI-NYA, Patrick. Optimized Adaptive Depth Map Filtering. 2013, [S.l.]: IEEE, 2013. p. 1356–1360. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6738279>. Acesso em: 5 maio 2014.

KRONOS GROUP. *WebGL Specification v1.0.2*. . [S.l: s.n.], mar. 2013. Disponível em: <https://www.khronos.org/registry/webgl/specs/1.0/>. Acesso em: 11 set. 2014.

LANG, Manuel *et al.* Nonlinear disparity mapping for stereoscopic 3D. *ACM Transactions on Graphics (TOG)*, v. 29, n. 4, p. 75, 2010.

LEBRETON, Pierre *et al.* 3D Video. In: MÃ¶LLER, SEBASTIAN; RAAKE, ALEXANDER (Org.). . *Quality of Experience*. T-Labs Series in Telecommunication Services. [S.l.]: Springer International Publishing, 2014. p. 299–313. Disponível em: <http://dx.doi.org/10.1007/978-3-319-02681-7_20>.

LEE, BongHo; YUN, Kugjin; LEE, Hyun; *et al.* Rich Media Services for T-DMB: 3-D Video and 3-D Data Applications. In: JAVIDI, BAHRAM; OKANO, FUMIO; SON, JUNG-YOUNG (Org.). . *Three-dimensional Imaging, Visualization, and Display*. [S.l.]: Springer US, 2009. p. 131–151. Disponível em: <http://dx.doi.org/10.1007/978-0-387-79335-1_8>.

LEE, BongHo; YUN, Kugjin; HUR, Namho; *et al.* Stereoscopic contents authoring system for 3D DMB data service. 5 fev. 2009, [S.l: s.n.], 5 fev. 2009. p. 72371D–72371D–12. Disponível em:

<http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=811327>. Acesso em: 17 dez. 2014.

LEE, Hyun *et al.* A structure for 2D/3D mixed service based on terrestrial DMB system. 2007, [S.l.]: IEEE, 2007. p. 1–4. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4379423>. Acesso em: 16 maio 2014.

LE FEUVRE, Jean *et al.* Experimenting with Multimedia Advances Using GPAC. MM '11, 2011, New York, NY, USA. *Anais*... New York, NY, USA: ACM, 2011. p. 715–718. Disponível em: <http://doi.acm.org/10.1145/2072298.2072427>.

LE FEUVRE, Jean. SVG Extensions for 3D displays Enabling SVG on auto-stereoscopic displays. 2010, [S.l: s.n.], 2010. Disponível em: <https://www.svgopen.org/2010/papers/54-SVG_Extensions_for_3D_displays/>.

LE FEUVRE, Jean. Towards Declarative 3D in Web Architecture. 2012.

LE FEUVRE, Jean; CONCOLATO, Cyril; MOISSINAC, Jean-Claude. GPAC: open source multimedia framework. 2007, [S.l.]: ACM Press, 2007. p. 1009. Disponível em: <http://portal.acm.org/citation.cfm?doid=1291233.1291452>. Acesso em: 31 jul. 2014.

LE FEUVRE, Jean; MATHIEU, Yves. Graphics Composition for Multiview Displays. *Emerging Technologies for 3D Video: Creation, Coding, Transmission and Rendering*, p. 450–467, 2013.

LEVELT, Willem JM. *On binocular rivalry*. 1965. Van Gorcum Assen, 1965.

LEVKOVICH-MASLYUK, L. *et al.* Depth Image-Based Representation and Compression for Static and Animated 3-D Objects. *IEEE Transactions on Circuits and Systems for Video Technology*, v. 14, n. 7, p. 1032–1045, jul. 2004.

LIMA, Guilherme Augusto Ferreira; SOARES, Luiz Fernando Gomes. Two normal forms for link-connector pairs in NCL 3.0. 2013, [S.l.]: ACM Press, 2013. p. 201–204. Disponível em: <http://dl.acm.org/citation.cfm?doid=2526188.2526238>. Acesso em: 17 mar. 2015.

LIM, Young-Kwon *et al.* MPEG Multimedia Scene Representation. In: CHIARIGLIONE, LEONARDO (Org.). . *The MPEG Representation of Digital Media*. New York, NY: Springer New York, 2012. p. 177–202. Disponível em: <http://link.springer.com/10.1007/978-1-4419-6184-6_10>. Acesso em: 10 jun. 2015.

LIU, Zhongxin; WANG, Wenmin; WANG, Ronggang. The design and implementation of stereoscopic 3D scalable vector graphics based on WebKit. 6 mar. 2014, [S.l: s.n.], 6 mar. 2014. p. 90111R. Disponível em: <http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2038681>. Acesso em: 29 jul. 2014.

LUMLEY, John; GIMSON, Roger; REES, Owen. A framework for structure, layout & function in documents. DocEng '05, 2005, [S.l.]: ACM Press, 2005. p. 32.

Disponível em: <http://portal.acm.org/citation.cfm?doid=1096601.1096615>. Acesso em: 10 set. 2014.

LUMLEY, John W. Functional, extensible, svg-based variable documents. DocEng '13, 2013, [S.l.]: ACM Press, 2013. p. 131–140. Disponível em: <http://dl.acm.org/citation.cfm?doid=2494266.2494274>. Acesso em: 10 set. 2014.

MCCORMACK, Cameron *et al. Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. W3C Recommendation. [S.l.]: W3C, ago. 2011.

MCMILLAN JR., Leonard. *An Image-Based Approach To Three-Dimensional Computer Graphics*. . [S.l: s.n.], 1997.

MENDIBURU, Bernard. *3D TV and 3D cinema tools and processes for creative stereoscopy*. Waltham, MA: Focal Press/Elsevier, 2012. Disponível em: <http://site.ebrary.com/id/10483435>. Acesso em: 18 jun. 2014.

MÜLLER, K; MERKLE, P; WIEGAND, T. 3-D Video Representation Using Depth Maps. *Proceedings of the IEEE*, v. 99, n. 4, p. 643–656, abr. 2011.

MUNSHI, Aaftab. *OpenCL programming guide*. Upper Saddle River, NJ: Addison-Wesley, 2012.

NOCENT, Olivier *et al.* Toward an Immersion Platform for the World Wide Web Using Autostereoscopic Displays and Tracking Devices. Web3D '12, 2012, New York, NY, USA. *Anais*... New York, NY, USA: ACM, 2012. p. 69–72. Disponível em: <http://doi.acm.org/10.1145/2338714.2338724>.

O3D Project's page. Disponível em: <https://code.google.com/p/o3d/>. Acesso em: 4 dez. 2014.

ORTIZ JR., Sixto. Is 3D Finally Ready for the Web? *Computer*, v. 43, n. 1, p. 14–16, jan. 2010.

OSFIELD, Robert; BURNS, Don; OTHERS. *Open scene graph*. [S.l: s.n.], 2004.

PERAKAKIS, Emmanouil; GHINEA, Gheorghita. A proposed model for cross-platform web 3D applications on smart TV systems. 2015, [S.l.]: ACM Press, 2015. p. 165–166. Disponível em: <http://dl.acm.org/citation.cfm?doid=2775292.2778303>. Acesso em: 17 ago. 2015.

PERKINS, M.G. Data compression of stereopairs. *IEEE Transactions on Communications*, v. 40, n. 4, p. 684–696, abr. 1992.

PICKERING, Mark R. Stereoscopic and Multi-View Video Coding. *Academic Press Library in Signal Processing: Image and Video Compression and Multimedia*, v. 5, p. 119, 2014.

REDERT, A. *et al.* Advanced three-dimensional television system technologies. 2002, [S.l.]: IEEE Comput. Soc, 2002. p. 313–319. Disponível em:

<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1024077>. Acesso em: 13 abr. 2014.

ROTARD, Martin. Layout Managers for Scalable Vector. In: SVG OPEN, ago. 2005, Enschede, The Netherlands. *Anais*... Enschede, The Netherlands: [s.n.], ago. 2005.

SABIRIN, Houari *et al.* DMB Application Format for Mobile Multimedia Services. *IEEE Multimedia*, v. 19, n. 2, p. 38–47, 2012.

SANT'ANNA, Francisco; CERQUEIRA, Renato; SOARES, Luiz Fernando Gomes. NCLua: Objetos Imperativos Lua Na Linguagem Declarativa NCL. WebMedia '08, 2008, New York, NY, USA. *Anais*... New York, NY, USA: ACM, 2008. p. 83–90. Disponível em: <http://doi.acm.org/10.1145/1666091.1666107>.

SCHEMALI, Leila; EISEMANN, Elmar. Design and evaluation of mouse cursors in a stereoscopic desktop environment. mar. 2014, [S.l.]: IEEE, mar. 2014. p. 67–70. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6798844>. Acesso em: 19 mar. 2015.

SEUNTIENS, Pieter; MEESTERS, Lydia; IJSSELSTEIJN, Wijnand. Perceived quality of compressed stereoscopic images: Effects of symmetric and asymmetric JPEG coding and camera separation. *ACM Transactions on Applied Perception*, v. 3, n. 2, p. 95–109, 1 abr. 2006.

SHADE, Jonathan *et al.* Layered depth images. 1998, [S.l.]: ACM, 1998. p. 231–242. Disponível em: <http://dl.acm.org/citation.cfm?id=280882>. Acesso em: 9 abr. 2014.

SHIBATA, T. *et al.* The zone of comfort: Predicting visual discomfort with stereo displays. *Journal of Vision*, v. 11, n. 8, p. 11–11, 21 jul. 2011.

SHREINER, Dave *et al. OpenGL programming guide: The Official guide to learning OpenGL, version 4.3*. [S.l.]: Addison-Wesley Professional, 2013.

SIGNES, Julien; FISHER, Yuval; ELEFTHERIADIS, Alexandros. MPEG-4's binary format for scene description. *Signal Processing: Image Communication*, v. 15, n. 4, p. 321–345, 2000.

SMOLIC, Aljoscha *et al.* Coding Algorithms for 3DTV; A Survey. *IEEE Transactions on Circuits and Systems for Video Technology*, v. 17, n. 11, p. 1606–1621, nov. 2007.

SOARES, Luiz Fernando Gomes *et al.* Revisiting the Inter and Intra-Media Synchronization Model of the NCL Player Architecture. In: MEDIASYNC, 2013, [S.l: s.n.], 2013.

SOARES, Luiz Fernando Gomes; LIMA, Guilherme Ferreira. *NCL Handbook*. , Monografias em Ciências da Computação., nº 18/13. [S.l.]: Pontifícia Universidade Católica do Rio de Janeiro, 2013. Disponível em: <http://handbook.ncl.org.br>.

SOARES, Luiz Fernando Gomes; MORENO, Marcelo Ferreira; SANT'ANNA, Francisco. Relating declarative hypermedia objects and imperative objects through the NCL glue language. 2009, [S.l.]: ACM Press, 2009. p. 222. Disponível em: <http://portal.acm.org/citation.cfm?doid=1600193.1600243>. Acesso em: 2 set. 2014.

SOARES NETO, CarlosdeSalles; SOARES, LuizFernandoGomes; DE SOUZA, ClarisseSieckenius. TAL—Template Authoring Language. *Journal of the Brazilian Computer Society*, v. 18, n. 3, p. 185–199, 2012.

SOARES NETO, Carlos de Salles; SOARES, Luiz Fernando Gomes; SOUZA, Clarisse Sieckenius. The Nested Context Language reuse features. *Journal of the Brazilian Computer Society*, v. 16, n. 4, p. 229–245, nov. 2010.

SONS, K. *et al.* xml3d.js: Architecture of a Polyfill implementation of XML3D. mar. 2013, [S.l: s.n.], mar. 2013. p. 17–24.

SOPIN, Ivan; HAMZA-LUP, Felix G. Extending the Web3D: design of conventional GUI libraries in X3D. Web3D '10, 2010, [S.l.]: ACM Press, 2010. p. 137. Disponível em: <http://portal.acm.org/citation.cfm?doid=1836049.1836070>. Acesso em: 4 ago. 2014.

SORENSEN, S.E.B.; HANSEN, P.S.; SORENSEN, N.L. *Method for recording and viewing stereoscopic images in color using multichrome filters*. . [S.l: s.n.]. Disponível em: <http://www.google.com/patents/US6687003>. , fev. 2004

SOUZA, Daniel FL *et al.* Incorporating 3D technologies to the Brazilian DTV standard: a study of integration strategies based on middleware ginga. 2010, [S.l.]: ACM, 2010. p. 251–258. Disponível em: <http://dl.acm.org/citation.cfm?id=1809828>. Acesso em: 14 jul. 2014.

SUN, Geng; HOLLIMAN, Nick. Evaluating methods for controlling depth perception in stereoscopic cinematography. 5 fev. 2009, [S.l: s.n.], 5 fev. 2009. p. 72370I. Disponível em: <http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.807136>. Acesso em: 21 jul. 2015.

SUTHERLAND, Ivan E. A Head-mounted Three Dimensional Display. AFIPS '68 (Fall, part I), 1968, New York, NY, USA. *Anais*... New York, NY, USA: ACM, 1968. p. 757–764. Disponível em: <http://doi.acm.org/10.1145/1476589.1476686>.

TAKAKI, Yasuhiro. Development of Super Multi-View Displays. *ITE Transactions on Media Technology and Applications*, v. 2, n. 1, p. 8–14, 2014.

TAM, Wa James *et al.* Smoothing depth maps for improved steroscopic image quality. 25 out. 2004, [S.l: s.n.], 25 out. 2004. p. 162–172. Disponível em: <http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=852441>. Acesso em: 9 abr. 2014.

TANIMOTO, Masayuki. FTV: Free-viewpoint Television. *Signal Processing: Image Communication*, v. 27, n. 6, p. 555–570, jul. 2012.

TSAI, Yi-Feng; YOUNG, Chung-Ping; KU, Chuan-Chun. Design and Implementation of a Flash Player Supporting Stereoscopic Image. jul. 2012, [S.l: s.n.], jul. 2012. p. 63–66.

VÁZQUEZ, Carlos; TAM, Wa James; SPERANZA, Filippo. Stereoscopic imaging: filling disoccluded areas in depth image-based rendering. 18 out. 2006, [S.l: s.n.], 18 out. 2006. p. 63920D–63920D–12. Disponível em: <http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=1294775>. Acesso em: 16 maio 2014.

VETRO, Anthony. Frame compatible formats for 3D video distribution. set. 2010, [S.l.]: IEEE, set. 2010. p. 2405–2408. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5651071>. Acesso em: 8 jul. 2015.

WANG, Wenmin *et al.* Stereoscopic 3D Web: From idea to implementation. abr. 2014, [S.l: s.n.], abr. 2014. p. 1440–1444.

WHEATSTONE, C. Contributions to the Physiology of Vision. Part the First. On Some Remarkable, and Hitherto Unobserved, Phenomena of Binocular Vision. *Philosophical Transactions of the Royal Society of London*, v. 128, n. 0, p. 371–394, 1 jan. 1838.

WOO, Seunghyun; SUH, Hyojin; CHEON, Hosang. Reinforcement of spatial perception for stereoscopic 3d on mobile handsets. 2012, [S.l.]: ACM, 2012. p. 2075–2080. Disponível em: <http://dl.acm.org/citation.cfm?id=2223755>. Acesso em: 8 abr. 2014.

X3D Architecture and base components V3. International Standard. [S.l.]: ISO/IEC, 4 nov. 2013.

ZHANG, Jianlong *et al.* A rendering approach for stereoscopic web pages. 6 mar. 2014, [S.l: s.n.], 6 mar. 2014. p. 90111O. Disponível em: <http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2038805>. Acesso em: 29 jul. 2014.

ZHANG, Shaobo; ZHOU, Jun; SUN, Jun. 3D Webpage Rendering by Canvas. In: ZHANG, WENJUN *et al.* (Org.). . *Advances on Digital Television and Wireless Multimedia Communications*. Communications in Computer and Information Science. [S.l.]: Springer Berlin Heidelberg, 2012. v. 331. p. 411–417. Disponível em: <http://dx.doi.org/10.1007/978-3-642-34595-1_56>.

Appendix A
Fundamentals of 3D Video

This appendix reviews the fundamentals of visual science for depth perception (Section A.1), their realization by 3D display technologies (Section A.2), and the main 3DV representations and formats (Section A.3). The goal is to summarize the main concepts that are useful to understand the remainder of the thesis; it does not intend to be exhaustive on the individual concepts presented here. A detailed discussion of these individual concepts is outside the scope of this thesis, and can be found in several other publications (DUFAUX; PESQUET-POPESCU; CAGNAZZO, 2013; MINOLI, 2011; ZHU et al., 2013). For each specific topic, the reader will be referred to relevant literature, in which more detailed descriptions can be found.

A.1.
Human Perception of Depth

Depth perception is our visual ability to perceive the distances of objects and the 3D world as a whole. What is fascinating about this ability is that our perception of the three-dimensionality is based on 2D images, projected onto our retinas (GOLDSTEIN, 2014).

To explain how we perceive three-dimensional space, researchers have tried to identify which information is contained in a 2D image that enable us to perceive depth in a scene. That is called the “cue approach to depth perception”, or “cue theory”. The “cue theory” claims that the human visual system (HVS) achieves depth perception based on a variety of “cues” and that we learn the connection between a cue and the real depth through our previous experience with the environment.

Over many years, researchers have identified the different cues we use to get depth information. A commonly used classification for depth cues (used in this thesis) is classifying them as *monocular* or *binocular* cues (see Figure 30). Another common way is to group them as *physiological*—those that we can feel—or

psychological—those related to how we interpret what we see. These depth-cues classifications are not independent. For example, linear perspective (detailed in Subsection A.1.1) is psychological and monocular; convergence (detailed in Subsection A.1.2) is physiological and binocular. Figure 30 summarizes the main depth cues discussed in the remainder of this section.

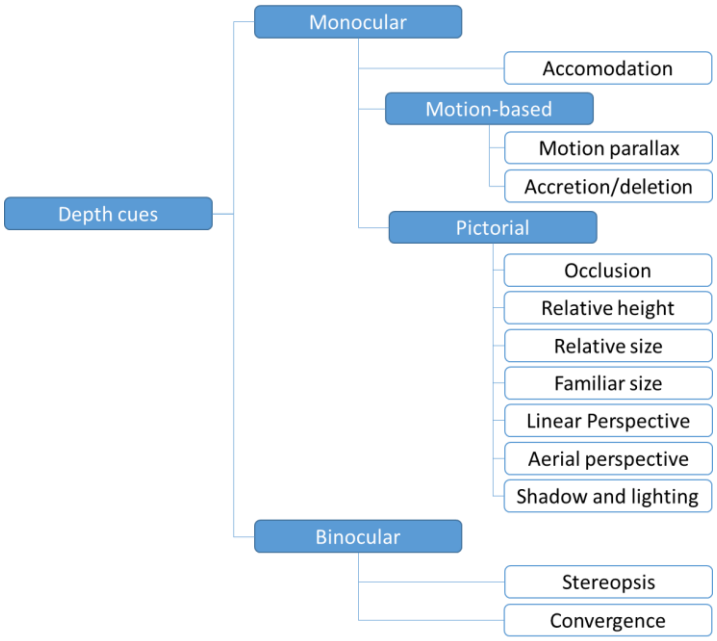


Figure 30 Depth cues classification according to (GOLDSTEIN, 2014).

A.1.1. Monocular cues

Monocular cues are those that we can perceive with only one eye. They can be further classified as *accommodation*, *pictorial* or *motion-based*. Accommodation is a physiological depth cue; pictorial and motion-based cues are psychological.

Accommodation is related to the change in the focal length of eyes as one focuses on objects at various distances. Objects close to us require the eye's lens to be more curved; we tighten muscles around our eyeballs to accomplish this effect. We subconsciously feel this and feed that information into our brain's interpretation of what we see.

Pictorial depth cues are depicted in flat 2D pictures, such as the figures printed in this thesis. Some of the most important pictorial cues are:

Occlusion (also referred to as interposition) occurs when a foreground object partially blocks the view of background objects. Partially occluded objects are perceived to be farther away.

Relative height According to this cue, objects that are below the horizon and have their base higher in the field of view are usually perceived to be more distant.

Relative size This cue is related to the fact that close objects cover a larger visual angle on our retina; they appear bigger than more distant objects. Thus, if two objects are of equal sizes, the closer one will appear bigger.

Familiar size is related to the fact that we already know some object sizes, e.g., trees, cars, and people. We compare the relative familiar sizes of objects in the same scene.

Linear perspective Parallel lines merge into a single vanishing point on the horizon. We perceive points closer to the vanishing point as more distant.

Atmospheric (or Aerial) perspective Objects at a greater distance have a lower contrast and saturation. Their color spectrum is also shifted towards blue due to light scattering in the atmosphere.

Shadows and lighting The way an object casts shadows and how light falls on its surface can also be used to derive depth information.

Motion-based cues are those that infer depth information created by relative movement between the observer and the observed object. Motion-based cues include *motion parallax*, *deletion*, and *accretion*. Motion parallax cues are some of the most important sources of depth information. They are related to the fact that objects closer to the observer move faster than farther ones. Moreover, as an observer moves sideways (and due to motion parallax and occlusion) some objects, or part of objects, become covered, and others become uncovered, what is known as deletion and accretion cues. Figure 31 illustrates some of the most important monocular cues.

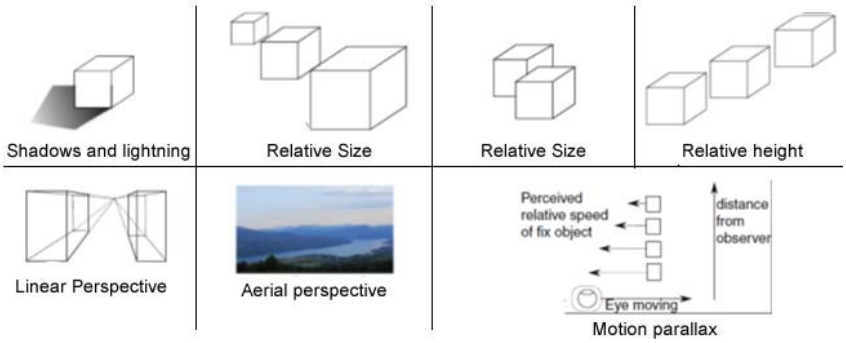


Figure 31 Monocular depth cues. Adapted from: (LEBRETON *et al.*, 2014).

A.1.2. Binocular cues

Binocular cues depend on both eyes working together. Two important binocular cues are *stereopsis* (or *binocular disparity*) and *convergence*.

Since our eyes are horizontally separated (by a distance of approximately 6.3cm in a human adult), each eye has its own perspective of the world. Therefore, the same point in the world is projected onto slightly different positions on each eye's retina; and each eye receives a slightly different image of the world. The distance between corresponding points in the images projected on each eye's retina is known as *retinal disparity*. Based on retinal disparity, the brain fuses the left and right images received by each eye in a process known as *stereopsis* and infers the relative depth information. Figure 32 schematically shows the stereopsis process, and how the binocular view is used to produce a single 3D image.

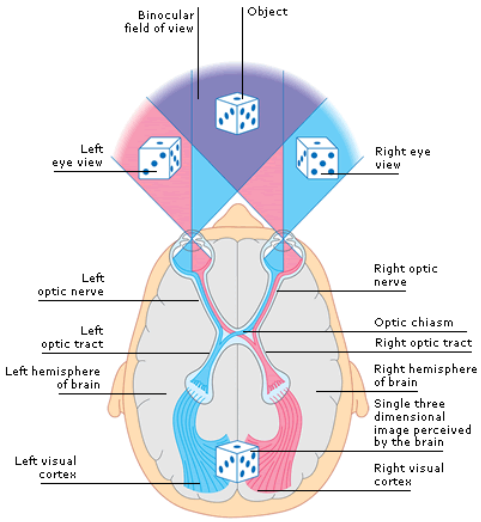


Figure 32 Binocular view and 3D image reconstruction.

When we look at nearby objects, our eyes move inward to converge on them. This is another important physiological source of information for depth perception, named *convergence*. HVS uses accommodation in combination with convergence, using the latter to correct the refraction power and to ensure clear image of the object being tracked (GOTCHEV *et al.*, 2011).

A.1.3. Depth cues and distance

According to "cue theory" the relationship between depth cues and the depth perception of a 3D scene is mainly learned by experience. Although those cues are also relatively independent of each other[7], they must be in harmony for a correct depth perception. When depth cues are not in agreement, the depth geometry of a scene will probably be unclear. Side effects, such as headaches and eyestrain, can also arise as a result of HVS's efforts to understand an unnatural scene.

It is also interesting to notice that depth cues have different importance for individual observers (HOWARD; ROGERS, 1996), and for varying distance between the viewer and the observed object (CUTTING; VISHTON, 1995). Figure

[7] A famous test, created by Bela Julesz (1971), that subjectively shows the independence between binocular and monocular depth cues is the so-called "random-dot stereograms" (RDS). In the RDS test, a stereo pair of images of random dots is viewed with the aid of a stereoscope. While the RDS has no pictorial cue (they are random dots without any shadows, linear perspective, etc.), they can produce a sensation of depth, with points appearing to be in front of or behind the screen level. This test "proves" that binocular disparity alone is a strong, and independent, depth cue for HVS.

33 depicts the relationship between the depth cues and the distance between the viewer and the observed object.

For close objects, binocular cues are often considered the most important ones. Stereopsis, for instance, is a strong depth cue for objects closer than one hundred meters (CUTTING; VISHTON, 1995). Convergence and accommodation are useful up to a distance of about an arm's length, with convergence being the most effective of the two (GOTCHEV *et al.*, 2011). Pictorial cues are relevant for images perceived at any distance. However, they tend to grow in importance for distant objects as other cues, such as accommodation and binocular, lose their importance.

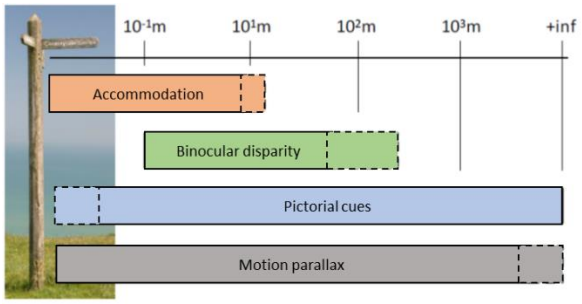


Figure 33 Depth perception as a set of separate visual "layers". Adapted from (GOTCHEV *et al.*, 2011).

A complete discussion and measurements of the importance of each depth cue based on the relative distance between the viewer and observed objects can be found in (CUTTING; VISHTON, 1995).

A.2. 3D Displaying Technologies

The 3D display is, perhaps, the most important component of a 3D media system. 3D displays directly affect the perceived QoE and ultimately provide the depth cues to the HVS. Moreover, the required 3D media representation technologies are also affected by display technologies. The remainder of this section reviews the most widely-used 3D display technologies (which are targeted by this thesis). Figure 34 shows a non-exhaustive classification of 3D displays.

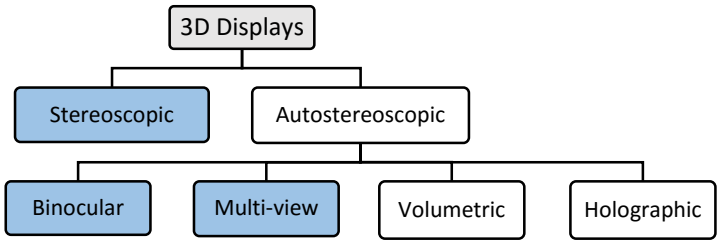


Figure 34 A non-exhaustive classification of 3D displays.

The proposals of Chapter 3 and 4 can work with both stereoscopic or binocular autostereoscopic displays. The proposals of Chapter 4 can also be used for composing multimedia applications for multi-view autostereoscopic 3D displays.

A.2.1. Stereoscopic Displays (Two views, eyeglasses based)

Stereoscopic 3D display technologies use special glasses, working as filters, to induce binocular disparity and convergence by providing different left-eye and right-eye images (HONG *et al.*, 2011). The first stereoscopic device was proposed by C. Wheatstone in the 1830s (WHEATSTONE, 1838), as illustrated in Figure 35. Stereoscopic technologies have been used in television and movies since 1920 (KIM *et al.*, 2013) and are still the base of current commercial 3D Cinema and 3DTVs.

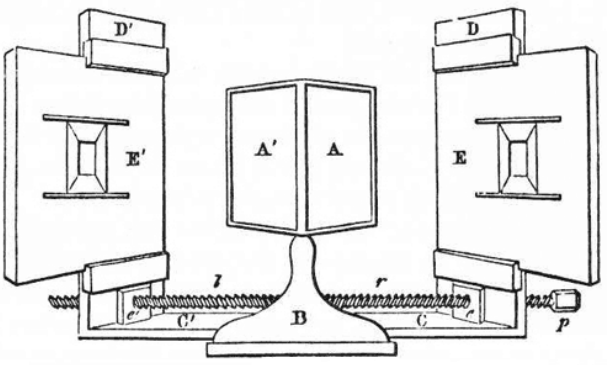


Figure 35 The Wheatstone stereoscope. It includes two mirrors (A' and A) placed at a specific angle to reflect the left and right eye drawings (E' and E) towards the viewer's eyes. The viewer must place his head in front of the mirrors to view images using the stereoscope. Adapted from (WHEATSTONE, 1838).

The most widely used techniques for filtering the left and right images (the so-called stereo-channel separation) are *color-interlaced* (or *anaglyph*), *time-multiplexed*, and *polarization-interlaced*. Another important type of stereoscopic display (which has been gaining commercial momentum in the last years) is the *head-mounted displays*. In the remainder of this subsection, a brief survey of those technologies (which are presented schematically in Figure 36) is presented.

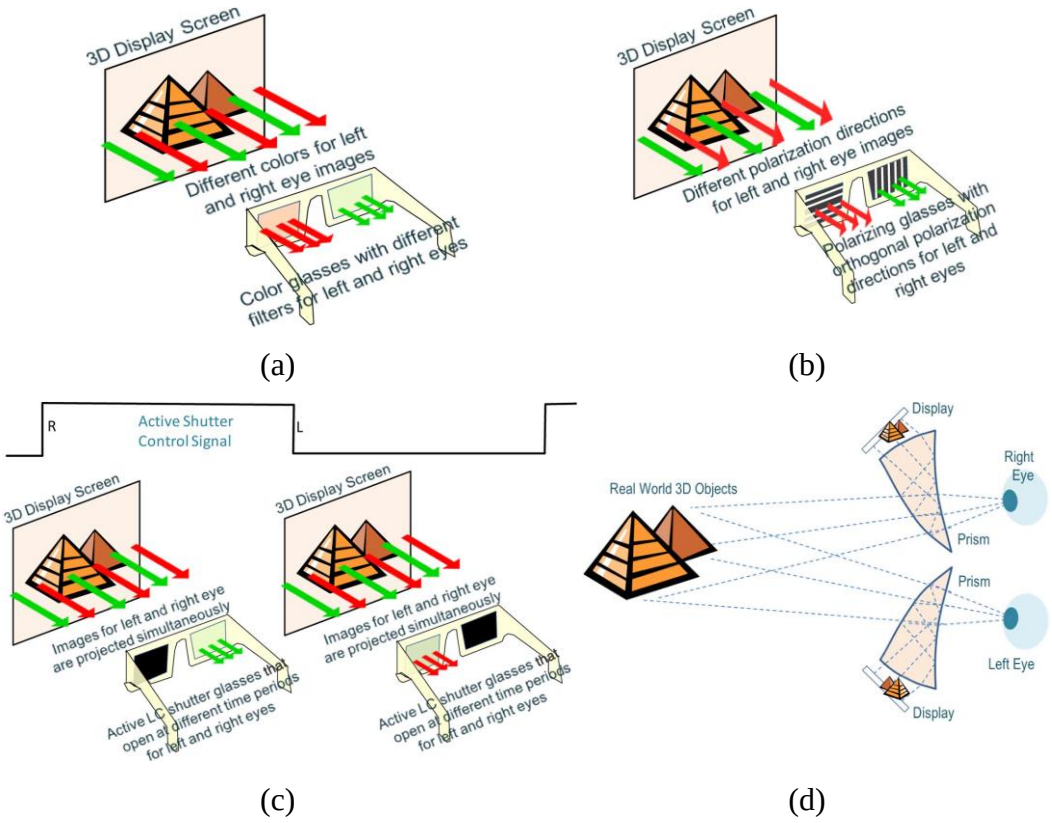


Figure 36 Different stereoscopic 3D techniques: (a) color-interlaced (anaglyph); (b) polarization-interlaced; (c) time-multiplexed; and (d) head-mounted display. Adapted from (GENG, 2013).

Color-interlaced (anaglyph)

In the *color-interlaced* (or *anaglyph*) technique (Figure 36(a)), left- and right-eye images are color-coded and combined into a single image, which can be printed or presented on a standard 2D display. When the combined image is viewed through glasses with lenses of corresponding colors, a 3D image is perceived. Various complementary color combinations have been used over the years. The most commonly used is the red/cyan combination.

Anaglyph methods have the advantage of not requiring any special projection system or screen display. On the other hand, color rivalry and unpleasant after-effects (transitory shifts in chromatic adaptation) can restrict its use (GENG, 2013). More recently, in the 2000s, Sorensen et al. (SORENSEN; HANSEN; SORENSEN, 2004) patented the ColorCode 3D stereo viewing system, which uses amber and blue filters. Unlike other anaglyph systems, ColorCode 3D is intended

to provide perceived nearly full-color viewing to existing television and printed media.

Polarization-interlaced Stereoscopic Display

A *polarization-interlaced* 3D system (Figure 36(b)) sends images to the corresponding eyes through polarized lights—light waves that vibrate in only one direction. The display presents both images simultaneously, usually with interlaced pixel columns or lines disposition to interlace the polarized image in the display.

Two polarization methods are commonly used: linear and circular. In the linear polarization method, one image is polarized so its vibration is vertical, whereas the other is polarized so its vibration is horizontal. In the circular polarization method, it is possible to use clockwise or counterclockwise polarization. Viewing both images simultaneous through corresponding (linear or circular) polarized glasses can filter the individual images for each corresponding-eye. The results of both polarization methods are similar, except that in the circular method the viewer can tilt his head and still maintain left/right separation (although stereoscopic image fusion will be lost due to the mismatch between the eye plane and the original camera plane).

Polarization-based 3D displays are commercially known as passive devices, since there are no electronics involved in the filter glasses. The images can be linear or circular polarized. Compared to other stereoscopic techniques, a polarized 3D system provides full-color, flicker-free images, and uses inexpensive glasses that do not require synchronization between devices. However, it cannot provide full resolution because the right- and left-view images are presented simultaneously (KIM *et al.*, 2013).

Time-multiplexed Stereoscopic Display

In the *time-multiplexed* stereoscopic displays (Figure 36(c)) the left and right views alternates on the display device while a blocking mechanism is required to prevent the left eye from seeing the right eye's view and vice versa. This kind of display exploit the so-called "persistence of vision", also known as the "memory effect". Such effect is related to the fact that the HVS is capable of merging the constituents of a stereo pair across a time lag of up to 50 ms (GENG, 2013). When

a system rapidly, and alternating, presents a left- and right- image to each human eye, the HVS merges those consecutive images into one 3D image.

A commonly used technology for time-multiplexed stereoscopic displays is the LC (Liquid Crystal) shutter. In this technology, the user wears LC shutter glasses that can block and unblock the observer's left and right eye viewing separately. The screen must show the left and right image as different frames multiplexed in time. The process requires a mechanism that synchronizes the screen and the LC shutter glasses. Such mechanism must ensure that, when the display presents the left frame, the left eye's shutter is open and the right eye's shutter is blocked, and vice-versa.

Time-multiplexed stereoscopic displays are commercially known as *active systems*. Since the left-eye and right-eye images are presented in different frames using all pixels on the display device, technologies based on LC shutter glasses have no resolution degradation when displaying 3D images.

Head-Mounted Displays

In a *Head-Mounted Display* (HMD) (SUTHERLAND, 1968) (Figure 36(d)), the user wears a helmet or glasses with two displays—e.g. small LCD or organic light-emitting devices (OLED)—with magnifying lenses, one for each eye (GENG, 2013). This technology can be used to show stereo films, images, or games, and can be used to create virtual displays. When coupled with head-tracking devices, it is also possible to provide users with a "look around" effect, which is very useful when immersing users in virtual worlds. Thanks to the continuing miniaturization of electronics, these devices are beginning to become available at more reasonable costs.

A.2.2. Binocular Autostereoscopic 3D displays (Two views, no glasses required)

There are some limitations with stereoscopic 3D displays, e.g., the use of glasses for filtering the left and right images is clearly an inconvenience. Autostereoscopic 3D displays provide 3D perception without requiring glasses or other headgear to provide 3D perception (DODGSON, NEIL A., 2005). They are usually built with additional optical elements aligned on the surface of the screen,

and operate by “casting” different images towards each eye of the observer to create binocular depth cues (GOTCHEV *et al.*, 2011). In its fundamental meaning, the term “autostereo” refers to a vast range of technologies, including holographic, volumetric, and integral imaging displays[8]. In the 3DTV industry, however, the “autostereoscopic displays” has been mainly used to refer to two technologies: *parallax barriers* and *lenticular sheets*.

In the parallax barrier method (Figure 37(left)), a sense of depth arises from the fact that narrow slits separated by opaque barriers in front of a screen enable each eye to see different sets of pixels. This is essentially a mask with openings and closings that blocks the light in certain directions (GOTCHEV *et al.*, 2011).

Lenticular sheets (Figure 37(right)) are composed of small lenses with special shapes, which refract the light in different directions. The lenses shapes are cylindrical or spherical in order to enable a proper light redirection.

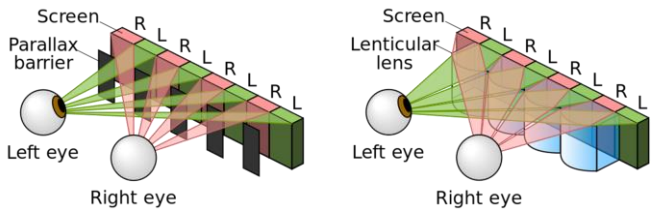


Figure 37 Light redirecting in binocular autostereoscopic displays: parallax barriers (left) and lenticular sheets (right). (Source: https://en.wikipedia.org/wiki/Autostereoscopy).

The obvious advantage of auto-stereoscopic displays is that they do not require the use of glasses. However, both aforementioned technologies have certain limitations. One of the main drawbacks is that the user’s eyes must be in a specific position, or “sweet spot”, so that the light beams can exactly reach the right and left human eyes. Moving outside this “sweet spot”, the user might catch the opposite view and experience the “pseudoscopy” effect. Non-ideal separation between the views creates inter-view crosstalk, which results in ghost-like images.

Because of the aforementioned restrictions, binocular autostereoscopic displays have been used mainly in mobile mono-user displays. Another possibility that can improve autostereoscopic viewing on mobile devices is the use of

[8] Those technologies are out of the scope of this thesis. Interested readers are referred to recent surveys on 3D technologies (KIM *et al.*, 2013) (GENG, 2013).

head/eye-tracking technologies, which can be used to dynamically adapt the light beams to the position of the eyes.

A.2.3.
Multi-view Displays (Many views, no glasses required)

An extension of binocular autostereoscopic displays is the so-called multi-view display. Multi-view displays are autostereoscopic displays that can provide more than one viewpoint (Figure 38). Each viewpoint allows for stereoscopic depth perception; at some level, multi-view displays are able to provide motion-based depth cues. The previously discussed technologies (parallax barrier and lenticular sheets) can be extended to provide those additional views. Commonly available multi-view displays provide 5, 9, or 28 views.

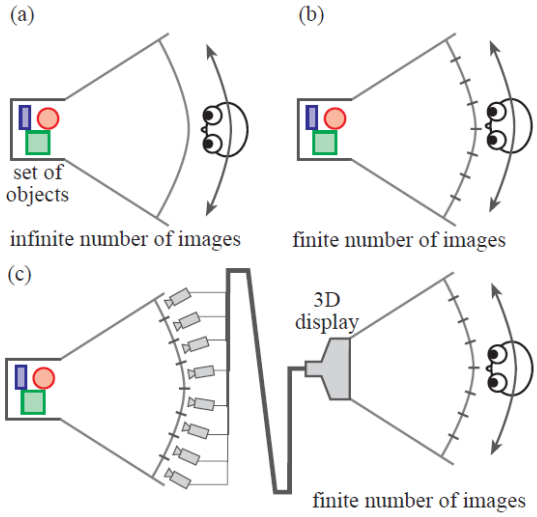


Figure 38 Principles of multi-view autostereoscopic 3D displays. As the user moves his head, different light beams, i.e., different viewpoint images, are perceived. (a) In the real world, the number of viewpoints is infinite; (b) Multi-view 3D displays provide a finite number of viewpoints of the world, which can be (c) captured from multi camera arrays. Adapted from (DODGSON, 2002).

More recently, super-multiview (SMV) displays (TAKAKI, 2014) have been proposed. In SMV displays, the interval between viewpoints is reduced, so it is smaller than the pupil diameter of the eye. This has the potential to solve the vergence-accommodation conflict, providing a continuous horizontal parallax.

A.3.
3D Media Formats

It is obvious that efficient representation is needed for successful 3DV applications. 3DV representation is closely intertwined with other components of 3DV systems: content acquisition, transmission, rendering, and display. For instance, stereoscopic 3D displays require only one stereo-pair, i.e., two views of the scene. Multi-view autostereoscopic displays, on the other hand, may require many stereo-pairs. Moreover, applications that use head-tracking techniques, even though presenting only a stereo-pair at each moment, may also require different views of the scene since user can dynamically change his/her viewpoint.

Some attributes are very important for a 3DV representation. First, it is important to decouple data representation from content acquisition and display. More specifically, 3DV formats should allow stereo devices to cope with varying display characteristics and viewing conditions by means of advanced stereoscopic display processing. For instance, controlling the baseline stereo distance to adjust depth perception can improve QoE and visual comfort. With the emergence of high-quality autostereoscopic multi-view displays, 3DV formats should enable the synthesis of many high-quality views. Moreover, the required bit rate should remain decoupled from the number of views, which calls for efficient compression.

This section details some of the most widely-used 3DV representations (some of them are briefly presented in Chapter 1) and discusses how they are related the proposals in this thesis. Figure 1 shows examples of 3D video representations.

Conventional Stereo Video (CSV)

CSV is the most well-developed and simplest 3D video representation. It is the de facto standard in current 3D Cinema and commercial 3DTVs. CSV codifies texture (color) information for the left and right eyes (Figure 1(a)), which is named a *stereo-pair*. Such stereo-pair is intended to be directly presented by stereoscopic 3D (S3D) displays. The easiest way to capture CSV images is to use two cameras, horizontally separated by approximately the distance between our two eyes.

A common way to codify a CSV video is using the so-called *frame-compatible* approach. In this approach, CSV videos are codified as a regular 2D videos that multiplexes the left and right views into a single frame or sequence of

frames (VETRO, 2010). Common frame-compatible approaches are side-by-side (SbS), top-and-bottom (TaB), checkerboard, and row and column interlaced (see Figure 39). Of these, top–bottom and side-by-side formats are the most common (CAGNAZZO; PESQUET-POPESCU; DUFAUX, 2013).

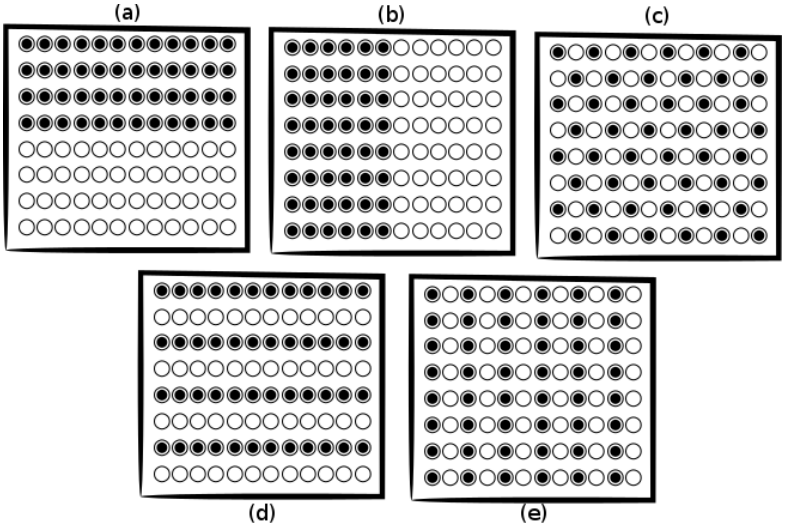


Figure 39 Frame-compatible approaches to codify CSV: (a) top-and-bottom; (b) side-by-side; (c) checkerboard; (d) row interlaced; and (e) column interlaced (◉ denotes samples from the first view; ○ denotes samples from the second view).

A derivative of CSV is the *Mixed Scene Resolution* (MSR) format (SEUNTIENS; MEESTERS; IJSSELSTEIJN, 2006). The concept of MSR was introduced by (PERKINS, 1992). In the MSR, one of the two sequences of the stereo-pair is sub-sampled. That representation is based on theories of binocular suppression, in which it is assumed that the binocular perception of a stereo image pair is dominated by the high-quality component (LEVELT, 1965).

The main drawback of CSV-based formats is that they double the amount of data to be stored and transmitted. Moreover, the lack of explicit geometry information of the scene (in both CSV and MSR) makes it hard to implement depth re-scaling features, which is a feature required for providing good QoE in different 3DV viewing conditions.

The proposals in Chapter 3 are related to the CSV frame-compatible approach for 3DV. They allow for extending 2D-only multimedia applications to support CSV-based media codified as side-by-side or top-bottom formats. The proposed

conversion process produces a multimedia scene specification that emulates one of the frame-compatible approaches to allow previous 2D-only multimedia applications to be presented on S3D displays taking advantage of the real depth information provided by these displays.

2D plus depth

As mentioned in Chapter 1, 2D plus depth (or 2D+Z) is a format in which the geometry of the scene is explicitly provided through a depth frame (or depth map) which is synchronized with the main (texture) video (Figure 1(b)).

The depth frame can be acquired through computer vision techniques (e.g. by using one or more original captured texture videos), directly captured with special cameras (such as time-of-flight), or artistically created. For videos based on synthetic content, the depth map can be synthetically acquired by rendering the scene with common computer graphics techniques (e.g., ray-tracing).

2D+Z format has the advantage of providing flexibility to various phases of a 3D media chain, allowing the adaptation of the final depth perception to different viewing conditions, keeping backward compatibility with 2D displays, and supporting the generation of additional views through Depth-Image-Based Rendering (DIBR) (FEHN, 2004) techniques. 2D+Z format has been standardized by MPEG in the MPEG-C part 3 (BOURGE; GOBERT; BRULS, 2006).

However, 2D+Z has its own drawbacks. The quality of the generated views is directed related to the quality of the provided depth maps. The process of acquiring high-quality depth maps is a challenge. Many research efforts have been done in recent years to improve these methods. In addition, by definition, 2D+Z lacks occlusion information.

Multi-view video (MVV)

MVV is a straightforward extension of CSV, in which several views can be synchronously acquired (Figure 1(c)).

It has the advantage of providing all information needed to be presented on a multi-view 3D display. However, just as in CSV, it lacks flexibility and cannot be easily adapted to different viewing conditions. Moreover, as the number of needed views increases, the amount of information to be transmitted will also

increase (linearly). For most applications, MVV is impractical to use in current network infrastructures.

Multi-view video plus depth (MVD)

MVD is an extension of the 2D+Z format, allowing a set of 2D plus depth view information, for different reference cameras (Figure 1(d)). The client is able to generate additional views between reference cameras. Compared to the 2D plus depth format, MVD offers higher field-of-view (still providing good rendering quality).

The most common approach for acquiring MVD is to use MVV as an input and to use computer vision techniques to extract depth information from them.

Layered-depth video (LDV)

LDV is an extension of the 2D plus depth format. It is based on the concept of Layered-Depth-Image (LDI) (SHADE *et al.*, 1998). LDV tries to avoid the redundancy information in MVD by transmitting only one central view together with additional layers. Those additional layers contain hidden texture and depth information values. In this format, a picture no longer consists of a single layer of pixels; it may contain several layers, with multiple associated depth data. Figure 40 provides an example of LDI (without depth layers).

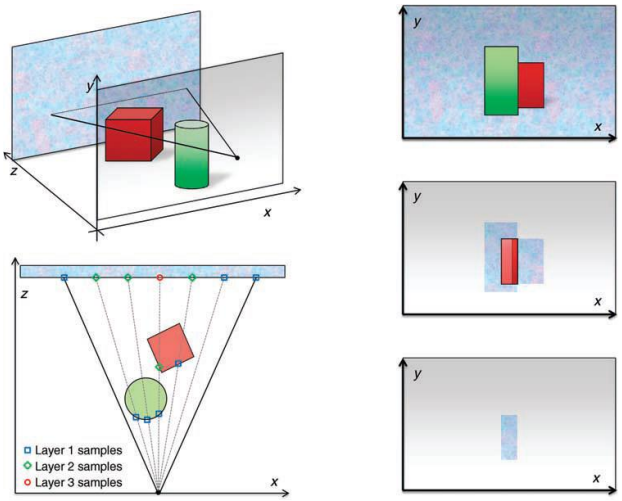


Figure 40 Representation of a layered-depth image. Left column, top: 3D scene; bottom: scheme of lines of sight. Right column, top: front layer; middle: second layer; bottom: last layer. Depth layers are not shown. (CAGNAZZO; PESQUET-POPESCU; DUFAUX, 2013).

Theoretically, each LDV/LDI frame can contain an infinite number of layers. In practice, however, they are usually restricted to a small number. A common approach has been to use only one occlusion layer[9](Figure 1(e)). A further optimization, which has the potential to improve coding performance, is to represent only the residual (difference) information between the main layer and the additional occlusion layer (see Figure 41).

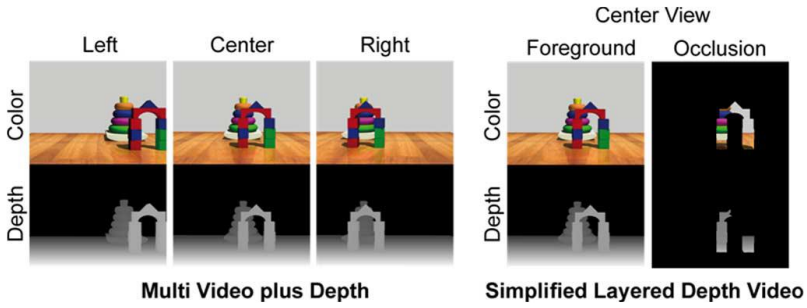


Figure 41 Differences between MVD and LDV. On the left side, MVD is shown for three cameras in a simple scene. On the right hand side, a simplified LDV representation (only one additional layer and represents only the residual information) is displayed. Adapted from (BARTCZAK *et al.*, 2011).

[9] This format has been successfully integrated into the Philips Wowvx® auto-stereoscopic 3D displays, in which it is named Declipse format (BARENBRUG, 2009).

Since the LDV format contains explicit depth and texture for occluded information, it allows for the generation of novel viewpoints for stereoscopic and multi-view displays, with better precision and higher performance than the 2D+Z format. This is because because the client may not need to infer the occluded regions (BARTCZAK *et al.*, 2011).

LDV has been used as an alternative to the MVD format, because the former is believed to achieve better compression performance. A common approach to generate the occlusion layers of LDV is to use the multiple views available in MVD (CHENG; SUN; YANG, 2007) (BARTCZAK *et al.*, 2011), a process that can be performed at the server side.

In this thesis, the concept of LDI/LDV is used to provide a software rendering architecture to multimedia players that supports depth-based 3D media. In addition, extensions to multimedia languages are provided to allow depth and occlusion information in LDV media. Compared to previous solutions, the use of LDI/LDV during the composition of scenes allows keeping occlusion information of all media objects while composing the multimedia scene. This is extremely useful in the process of filling the holes generated by 3D warping.

A.4. Geometry of Stereoscopic 3D Displays

Two-view 3D displays, such as stereoscopic or binocular autostereoscopic displays, provide the user with two images (a stereo-pair). Each image on a stereo-pair is almost the same, but may include some offset between the same point in the left and right views. This section derives the relationship between the perception of depth and a stereo-pair presented on a S3D display. To do that, Figure 42 illustrates the viewing parameters of stereoscopic images on S3D displays. From Figure 42, it is possible to see the disparities presented to the observer depend on:

(i) the screen parallax (P), i.e., the horizontal linear distance between common (homologous) points on the display surface;

(ii) the distance between the observer and the screen (V); and

(iii) the observer's eye separation (E), also known as *interocular* or *inter-pupillary distance* (IPD)[10].

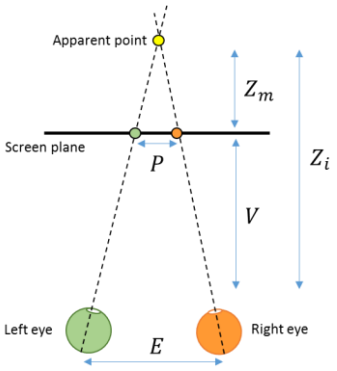


Figure 42 Stereoscopic viewing parameters, with an image point projected behind the screen, i.e., with positive parallax.

The observed depth Z_i of an image point projected with screen parallax P is equal to the sum of the distance between the observer and the screen, V, plus an offset term, Z_m. Using similar triangles, we obtain the following equation:

$$\frac{P}{Z_m} = \frac{E}{V + Z_m} \tag{5}$$

From which we can deduce Z_m as:

$$Z_m = \frac{PV}{E - P} \tag{6}$$

Since the observed depth is equal to $V + Z_m$, it is possible to express Z_i as:

$$Z_i = V + \frac{PV}{E - P} \tag{7}$$

which can be reduced to:

$$Z_i = V\frac{E}{E - P} \tag{8}$$

Based on Equation (8), some interesting conclusions follow:

[10] In adult males, IPD usually ranges from 5.77cm to 6.69cm, with a median of 6.32cm (IJSSELSTEIJIN; SEUNTIENS; MEESTERS, 2001). 6.5cm is usually taken as the average IPD. However, it is important to produce stereoscopic content that is comfortable to people with smaller IPD.

(i) as the observer moves closer to the screen (i.e., V decreases), the perceived depth for a same amount of parallax decreases;

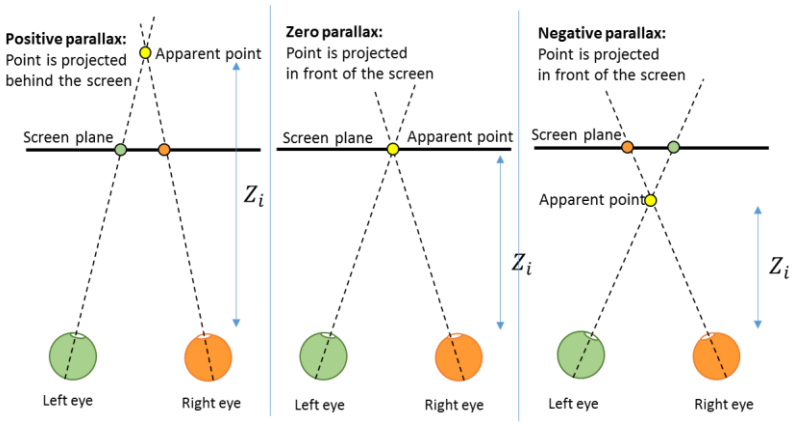
(ii) reducing the screen size (and thus of parallax) at a given viewing distance also decreases the perception of depth;

(iii) when the parallax P is equal to zero, the perceived depth is equal to the screen distance, V, that is, the object of interest will be seen at the screen plane. That configuration is named **Zero Parallax** (Figure 43(b)).

(iv) as parallax becomes **negative**, the image is seen in front of the screen plane (see Figure 43), and the viewer has to look cross-eyed, potentially making binocular fusion slightly uncomfortable, depending on the amount of negative parallax.

(v) when $P = E$, the observer's eyes are looking straight ahead, which corresponds to looking at an image infinitely far away.

(vi) there is no visible point in increasing the on-screen parallax (P) beyond eye separation (E). Moreover, if the eyes need to diverge ($P > E$) to fuse an image, this requires unusual muscular effort, which may cause discomfort.



(a) (b) (c)

Figure 43 Relationship between different types of parallaxes and depth perception. (a) positive parallax; (b) zero parallax; (c) negative parallax.

Appendix B
Depth-based Layouts for Multimedia Applications

B.1.
Introduction

Layout managers implement policies (or layout templates) defining how graphical components are positioned in containers (ROTARD, 2005), thus helping to provide good-looking software interface for different display sizes and viewing conditions. Layout managers are used in many graphical user interface (GUI) toolkits. In addition, they are useful when presenting multimedia (MM) applications, *e.g.*, for the Web or Digital TV (DTV). Extensions to support layout templates have been integrated into standard multimedia languages (ROTARD, 2005) (AMORIM; DOS SANTOS; MUCHALUAT-SAADE, 2013) (ETEMAD; JR; ATANASSOV, 2014). However, they are usually restricted to 2D-only layers, even when discussed in the scope of 3D multimedia languages. As argued by Sopin and Hamza-Lup (SOPIN; HAMZA-LUP, 2010), the reason probably comes from the fact that people are more acquainted with planar interface layouts. Moreover, the amount of different possibilities in organizing the components in a complete 3D coordinate system makes the definition of 3D layout managers non-trivial.

Nevertheless, between 2D-only and complete 3D spatial layouts there is the alternative of representing multimedia scenes with a 2D+depth (or 2.5D, or 2D with seeming depth) coordinate system (LE FEUVRE, 2010) (AZEVEDO; SOARES, 2013). As suggested by Cockburn and McKenzie's experiments (COCKBURN; MCKENZIE, 2002), 2D or 2D+depth formats are usually more effective to user interfaces than complete 3D formats. Furthermore, 2D+depth format is usual in 3DTV systems for encoding the main video content, known as video-plus-depth (MÜLLER; MERKLE; WIEGAND, 2011). For those videos, the depth information is encoded in a depth frame (synchronized with the texture frame) as a normalized value related to the true depth distance between each pixel and the viewer (Section A.3 details that format). Commonly, zero (or black) means far

from the viewer (inside the display), whereas 255 (or white) means close to the viewer.

Providing a 2D+depth coordinate system for multimedia applications can be particularly useful when presenting them on 3D displays, such as stereoscopic and multi-view auto-stereoscopic (BENZIE *et al.*, 2007). In doing this, additional presentation effects can be achieved. Per-component and/or per-pixel depth information could be associated with each individual component of a scene. Not only objects encoded with explicit per-pixel depth information, such as video-plus-depth or image-plus-depth (LE FEUVRE, 2010), could compose a scene, but also 2D media objects augmented with depth through the language constructions. Moreover, multimedia applications could be allowed to control depth information during execution, allowing for animations and additional artistic effects, such as objects popping in and out of the screen plane. Besides achieving a more natural perception, the use of a depth dimension could provide genuine user interfaces, for example, by reinforcing spatial perception, emphasizing elements of the interface, or improving the feedback from user actions (WOO; SUH; CHEON, 2012). Figure 44 shows an example of a 2D+depth multimedia application running on a stereoscopic TV.

Figure 44 Example of a stereoscopic 2D+depth multimedia application (in horizontal interlaced mode).

Current layout-manager and layout-template approaches that only focus on 2D coordinate systems are unsatisfactory to produce appealing multimedia applications to be presented on 3D displays. The current available options do not take advantage of depth information that 2D+depth multimedia applications are able to control. Aiming at easing the development of 2D+depth multimedia applications for 3D displays, this appendix proposes the concept of *depth-based*

layouts, *i.e.*, layout templates that organize the graphical components on a 2D+depth space. It identifies some common depth-based layout templates that can be useful for a broad range of 2D+depth multimedia applications, and discusses a *template processor* implementation to support these layouts in the scope of standard multimedia authoring languages and multimedia template languages extended with 2D+depth support. The layout template processor implementation proposed in this appendix can be extended through a well-defined API to support new layout templates besides the pre-defined ones identified here.

The remainder of the appendix is structured as follows. Section B.2 discusses work related to the proposal of depth-based templates. Section B.3 details the concept of 2D+depth multimedia applications and presents previous proposals intending the low-level specification of these applications in the scope of standard multimedia languages. Section B.4 introduces our proposal, discusses some depth-based layout templates, and presents an overview of the proposed depth-based layout template processor. Finally, Section B.5 is reserved for conclusions and future work.

B.2. Related Work

Most of current GUI toolkits provide basic layout managers for organizing graphical components in a 2D-only space, following a pre-defined layout template, such as *box* (horizontal or vertical), *flow*, and *grid* layouts. Figure 45 shows examples of these layout managers. Box layouts put their components into a single line (horizontal) or column (vertical). Flow layouts put their components into a single line, starting a new one if the width of the container is not enough to contain a new object. Grid layouts display components on a grid with a predefined number of columns and lines. Spacing between the objects can also be controlled through parameters. Since most of these layout managers are in the scope of general purpose programming languages, they can be extended, through a well-defined API. Graphic designers are also able to create complex GUIs by adding components that follow layout templates to containers, which also follow layout templates, thus creating a hierarchy of layout templates.

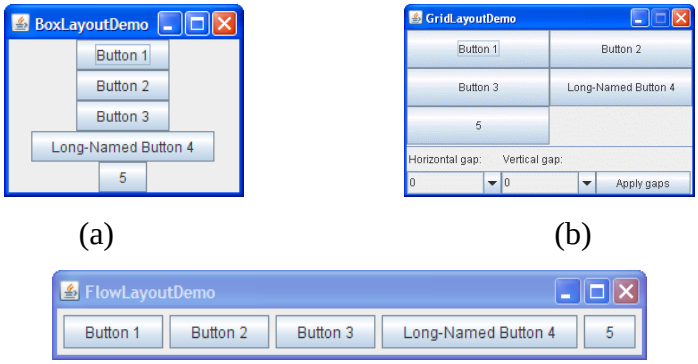


Figure 45 2D-only layout templates examples: (a) box layout; (b) grid layout; and (c) flow layout.

XML is commonly used as a declarative approach to describe the relationship between layout templates and graphical components. Some examples include Qt UI language[11] and Layout Android language[12]. The XML document is interpreted during run-time, or preprocessed, to generate the final source code. In the Qt UI case, XML files are used to generate C++ source code.

On the one hand, our proposal is similar to the one used in GUI toolkits, since it also proposes some basic layout templates. On the other hand, our layout templates are in the context of 2D+depth multimedia applications, targeting 3D displays. Moreover, it provides an XML notation used as input to generate the final multimedia application.

2D layout template approaches have also been integrated into standard multimedia authoring languages, such as SVG, NCL, and HTML. For instance, Rotard (ROTARD, 2005) proposes flow and grid layouts for SVG, whereas Amorim *et al.* (AMORIM; DOS SANTOS; MUCHALUAT-SAADE, 2013) propose the same template layouts in the context of NCL. As the HTML5 ecosystem evolves to become a platform for traditional GUIs, it is also possible to identify many works aiming at including more advanced grid-based layouts as both JavaScript toolkits and CSS extensions (ETEMAD; JR; ATANASSOV, 2014). However, none of the aforementioned work targets either 3D displays or systems with a 2D+depth coordinate support. In contrast, this appendix targets 2D+depth coordinate support that can be used to extend current authoring languages or to be

[11] http://qt-project.org/doc/qt-4.8/designer-ui-file-format.html
[12] http://developer.android.com/guide/topics/ui/declaring-layout.html

preprocessed generating new documents in these languages (such as proposed in Chapter 3 of this thesis), using their low-level depth support.

Instead of using pre-defined layout templates with minor parameters that are easily controllable, some approaches use lower-level constraints to define 2D-only layouts. The proposals of Borning *et al*. (BORNING; LIN; MARRIOTT, 2000), Jacobs *et al.* (JACOBS *et al.*, 2004), Brados *et al.* (BADROS *et al.*, 2001), and Lumley *et al.* (LUMLEY, JOHN; GIMSON; REES, 2005) (LUMLEY, JOHN W., 2013) are examples. Whereas very expressive, these constraint-based approaches are harder to use when compared to the layout templates. In this appendix, we focus mainly on providing higher-level definitions of depth-based layout templates, which are translated or interpreted. Indeed, constraint-based approaches could also be used to program the depth-based layouts proposed here. However, such discussion is out of the scope of this appendix.

The constraint-based layout approach of Jacobs et al. (JACOBS *et al.*, 2004) uses z-order data in graphical components with a different semantic from that used in our proposal. Since grid-based page design often includes overlapping elements, or regions that appear to be cut out from other elements, their system uses z-order information to produce effects like wrap text around figures, etc. Since we are targeting 3D displays, we use true depth information to arrange graphical components.

More similar to this work, Sopin and Hamza-Lup's proposal (SOPIN; HAMZA-LUP, 2010) use the concept of layouts in a 2D GUI with seeming depth information. They have proposed a library to create conventional GUI components in X3D ("X3D Architecture and base components V3", 2013), and have recognized the importance of 2D and 2D+depth coordinate systems in creating GUIs. Their approach allows different graphical components to have different depth (different layers). However, their layout templates (Border, Box, Grid, and Flow) do not support depth information for each individual component. Layout templates are restricted to only one layer, *i.e.*, the same depth for all components in the same layout container. In contrast, we propose layout templates that organize components in a container such that each component can have a different depth.

B.3.
Low-level 2D+Depth Support in Multimedia Languages

Low-level extensions to multimedia languages aiming at supporting 3D displays are not novelties. However, with the boom of 3D-stereoscopic technologies in recent years, there is a growing interest in them. Since most of 3D displays currently available are stereoscopic-only, some approaches focus on these displays. For instance, there has been an effort in developing a W3C draft aiming at extending CSS to support stereoscopic presentations (HANG; LEE, 2012). Also based on CSS, Chen *et al.* (CHEN, QINSHUI *et al.*, 2014) propose a basic approach to support stereoscopic images and a per-component depth for each component in an HTML page. Liu *et al.* (LIU; WANG; WANG, 2014) propose similar extensions for SVG. These three works focus on supporting stereoscopic-only displays, and do not mention multi-view autostereoscopic displays. Moreover, the proposal of Chapter 3 allow for supporting stereoscopic effects at the application-level for the NCL language (AZEVEDO; LIMA; SOARES, 2015).

By using the per-component depth association, it is possible to attach depth information to a media object, specifying how far it is from the viewer when rendered on a 3D display. Since most video-plus-depth codification schemas uses a quantized grayscale (0-255), a value in this range can be the one associated to a component depth, as proposed by Le Feuvre (LE FEUVRE, 2010). Double values can also be used to define the depth of media components. For instance, the value could be in the range [-1.0, 1.0], where 1.0 means closer to the viewer and -1.0 far from him/her (inside the display), or vice-versa. This is the solution provided by Chen *et al.* and Azevedo and Soares (AZEVEDO; SOARES, 2013) (and the Chapter 3 of this thesis), for CSS and NCL, respectively.

The high-level depth-based layouts proposed in this appendix can benefit the languages discussed in this section, or any other multimedia representation that uses 2D+depth coordinate system. Two solutions can be adopted to integrate the depth-based layouts into these languages. First, language players could be natively extended to support the new high-level layout primitives. Alternatively, a pre-processing step can be used to translate the high-level layout templates to the low-level depth primitives provided by the languages. The second alternative is used in this appendix, and detailed in Section B.4.1.

B.4.
High-level Depth-Based Layouts

The high-level layout templates proposed are defined using XML. To allow for a layout template instance to be reused in different multimedia applications, a solution based on style-content separation is adopted, as in XSLT, CSS, etc. Each layout template instance acts as a container (possibly containing other recursive containers), where graphical elements must be inserted. A heuristic algorithm resizes and disposes layout child components on the 2D and depth axes.

The language discussed in this section is an extension of the XML language previously defined by Amorim *et al.* (AMORIM; DOS SANTOS; MUCHALUAT-SAADE, 2013) to support 2D-only (Flow and Grid) layout types. The XML elements are in the *lay* XML namespace. The <layout> is the root element that defines a container instance. A specific layout template is given by the attribute *type* of the <layout> element. Optional spatial attributes, such as *top*, *left*, *width*, and *height* define the bounding-box of a <layout> element. If these attributes are not defined, they can be inferred based on child components' spatial attributes as discussed afterwards. Other attributes can also be supported and their name and possible values are dependent on the layout type (*type* attribute) in use.

The <layout> element can have <item> and other <layout> elements as children. The *id* attribute of an <item> element is unique in the document and can be used, by a host language, to associate graphical elements to the container, through *lay:layoutItem* attribute. For instance, if the template defines the <layout> element:

```
<lay:layout id="mediaContainer" type="flow"
            width="400px" height="400px">
  <lay:item id="VMC" …/>
  <lay:item id="XYZ" …/>
  ...
</lay:layout>
```

and this layout template is referred in the following NCL document:

```
<body ...>
  <media id="video" lay:layoutItem="VMC"/>
  <media id="image" lay:layoutItem="XYZ"/>
  <media id="text" lay:layoutItem="VMC"/>
  <media id="lua" />
  ...
</body>
```

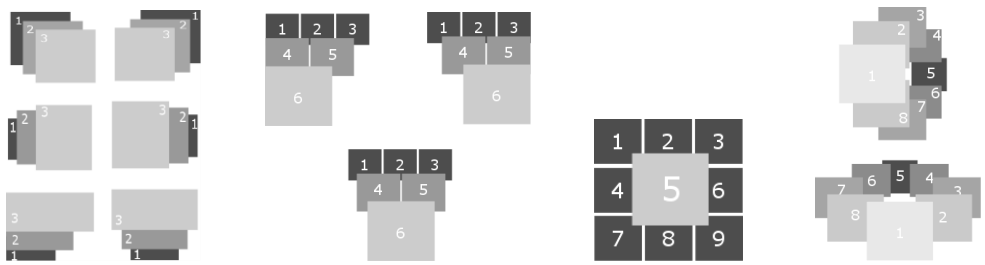
the “video”, “text”, and “image” <media> elements will be inserted into the “mediaContainer” layout container, in this order, whereas the “lua” <media> element will not be inserted.

An <item> element can also specify spatial attributes to be inherited by its associated graphical elements. The template heuristic can consider the spatial attributes of the container and the spatial information of its child components to determine the exact position and dimension of each child component. The spatial information of child graphical elements used in the heuristic estimates can be obtained from the <item> elements associated to the graphical elements, or from the host language that defines these elements. Since a <layout> element can contain another <layout> element, the spatial information of child layouts can be obtained from the child layout specification, if it is specified, otherwise from its child components’ spatial information, recursively. The exact position and dimension (in the 2D+depth coordinate system) of each child component is also evaluated recursively, but now from the outermost to the innermost layout container.

The basic depth-based layouts proposed here are: *Stack*, *DepthFlow*, *DepthGrid*, and *Carousel*. Unlike 2D-only layouts, the bounding-box that defines a container also includes a *min-depth*, and *max-depth* attributes. Table 2 summarizes the proposed depth-based layouts and presents their attributes. Figure 46 shows examples of the proposed depth-based layouts. In what follows, their heuristics are detailed.

Depth-based layout type	Attributes	Possible values
stack	align orientation order	“left”, “right”, “center” “left-to-right”, “right-to-left”, “bottom-to-top”, “top-to-bottom”, “diagonal” “front-to-back”, “back-to-front”
depthFlow	align orientation order	“left”, “right, “center”, “alternate-lr”, “alternate-rl” “vertical”, “horizontal” “front-to-back”, “back-to-front”
depthGrid	rows columns cell-depth(X, Y)	integer integer double between [-1.0, 1.0]
carousel	orientation rotation x-radius y-radius	“vertical”, “horizontal” “clockwise”, “counterclockwise” px or % px or %
	Common attributes	Possible values
	width height min-depth max-depth resize-components	px or % px or % double between [-1.0,1.0] double between [-1.0,1.0] “true”, “false”, “same”

Table 2 List of depth-based layouts currently integrated in our layout processor, their parameters, and possible values.



(a) (b) (c) (d)

Figure 46 Examples of depth-based layout templates: (a) Stack layout; (b) DepthFlow layout; (c) DepthGrid layout; and (d) Carousel layout.

Stack Layout defines that graphical components in the container must be displayed one over the other, with some displacement on their horizontal or vertical positions. The last inserted element is the top-most one (*i.e.* it has the greatest depth attribute). The size of the layout container (*width*, *height*) and its minimum and maximum depth information (*depth-min*, *depth-max*) must be specified through <layout> attributes. The *orientation* attribute specifies how the Stack Layout’s algorithm organizes components inside the container. Figure 46(a) shows examples of Stack layout instances. As in all images shown in this section, depth information of media components is coded in grayscale (255 meaning closer to user, and 0 far away from him). Components are inserted into the layout container in ascending order. The depth of each component is set linearly from depth-min to depth-max.

The following source code shows an example of Stack layout template specification.

```
<lay:layout id="stackLayout" type="Stack"
            width="400px" height="400px"
            min-depth="-0.5" max-depth="0.5"
            orientation="left-to-right"
            align="center">

  <lay:item id="VMC" width="200px" height="200px"/>
</lay:layout>
```

The container has a 2D dimension of 400x400px. All elements inside the container will be linearly spaced on the depth axis from -0.5 to 0.5.

DepthFlow Layout is an extension of the 2D-only *Flow* layout. As in its 2D-counterpart, components are inserted in order on the same line, while there is enough space on the line. If there is not enough space to insert a component, a new line is inserted, below and *in front* (*i.e.* with a greater depth attribute) of the previous one, as shown in Figure 46(b). Each line has a different depth value. Therefore, they are rendered at a different distance from the viewer. As usual, the min- and max-depth of the DepthFlow layout container can be specified as attribute values. The DepthFlow's heuristic algorithm guarantees that the position of the components follows such constraints and that the line's depth information grows linearly. The *align* attribute specifies how objects are aligned on each line. Possible values include "center", "left", and "right". The default value is "center". Moreover, if the attribute *orientation* is equal to "vertical", it is possible to adapt the DepthFlow behavior to work with columns instead of lines.

DepthGrid Layout is an extension of the 2D Grid layout in which components are placed on a grid of cells. Each component takes all available space within its cell, and each cell has exactly the same size. If a grid container is resized, all grid cells are resized. In a *DepthGrid* layout container, each cell of the grid can be associated to a depth information. Therefore, it is possible to bring some cells inside or outside of the screen plane defining their depth information. Figure 46(c) shows an example of DepthGrid layout, in which only the fifth cell has a different depth.

Carousel Layout presents its components in a 3D carousel interface, as shown in Figure 46(d). The carousel orientation is specified by the namesake attribute, which can receive the vertical or horizontal value. The spacing among the carousel's components is specified by the *x-radius* and *y-radius* attributes.

Whereas it is possible to use the above layout templates individually, an interesting feature is to create complex user interface composing them hierarchically. In such case, each layout container must respect not only the 2D-spatial constraints of their parent, but also the min- and max-depth defined by the parent heuristic of the container instance.

Another interesting possibility is to compose 2D-only layout templates with depth-based layouts. For example, it is possible to include a Stack layout container as a cell of a 2D-only grid layout. In such case, the 2D-only layout container behaves as if *min-* and *max-depth* attributes are, respectively, the minimum and maximum depth values supported by the multimedia player (*e.g.*, using NCL+depth extensions, *min-* and *max-depth* will be -1.0 and 1.0, respectively). The following source code shows an example of how to specify nested layout templates integrating a 2D flow layout and a depth-based Stack layout.

```
<lay:layout id="flowLayout"
            type="flow"
            width="200px" height="200px"
            align="center">

  <lay:item id="flowLayoutItem" .../>

  <lay:layout type="depthFlow">
    <lay:item id="flowLayoutItem" .../>
  </lay:layout>
</lay:layout>
```

B.4.1. Depth-based Layout Processor

There are at least two possibilities of using the above depth-based layouts: through layout templates natively supported by the multimedia language player; or by translating them to low-level depth primitives supported. The implementation reported in this appendix follows the second approach. We call *layout processor* the translator. Its prototype has been implemented using the Lua[13] scripting language, and follows the architecture illustrated in Figure 47.

[13] http://lua.org

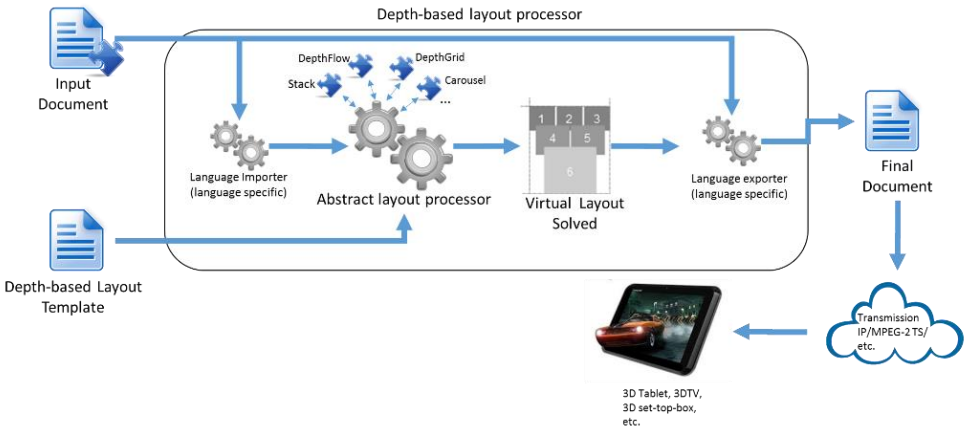


Figure 47 Architecture of the depth-based layout processor.

To be independent from the input and target languages, the layout processor core works with virtual components and virtual layout concepts. Only after accomplishing its tasks (positioning and sizing the components), it fills the final multimedia document. The "Language Importer" and "Language Exporter" components are responsible for converting from the input document to the internal layout model, and from the internal layout model to the final document syntax representation, respectively. As a usage example, a Language Importer and Language Exporter have been developed for NCL extended with the low-level depth primitives of Chapter 3.

Since we are aware that the proposed depth-based layout templates are not sufficient for all kinds of 2D+depth multimedia applications, extensibility is an important requirement for the layout processor prototype. New layout template heuristics can be integrated into the processor, by registering a Lua table that contains the following fields:

- the *name* of the new layout template type (as in *type* attribute);
- the *attributes* (both required and allowed) it supports; and
- the method that implements the heuristics for that layout template type (named *solve* function, in what follows).

The *solve* function is called when solving the layout, hierarchically, and is responsible for positioning and resizing the graphical child components in a container. The *solve* function receives as parameters:

- the *virtual layout container* that must be solved;
- its *attributes* and *constraints* (*width*, *height*, *max-depth*, *min-depth*, etc.); and

- the *virtual components* that are inside the container, and must be positioned and sized.

The API used to extend the layout processor is completely based on virtual components. By using virtual components, a layout template author does not need to be aware of the input or output multimedia language used to define the graphical elements, and is not committed either to input or to output languages.

B.5. Conclusion

This appendix proposed the use of depth-based layout templates for authoring multimedia applications targeting 3D displays. As high-level abstractions, they are easy to use and configure. Some basic depth-based layout templates (Stack, DepthFlow, DepthGrid, and Carousel) and an XML-notation to describe and configure them are defined. However, the concept is not restricted to the layout language used in the appendix and can be applied to other languages, such as CSS.

Most of current GUI toolkits provide basic layout managers for organizing graphical components in a 2D-only space, following a pre-defined layout template, such as *box* (horizontal or vertical), *flow*, and *grid* layouts. Box layouts put their components into a single line (horizontal) or column (vertical). Flow layouts put their components into a single line, starting a new one if the width of the container is not enough to contain a new object. Grid layouts display components on a grid with a predefined number of columns and lines. Spacing between the objects can also be controlled through parameters. Since most of these layout managers are in the scope of general purpose programming languages, they can be extended, through a well-defined API. Graphic designers are also able to create complex GUIs by adding components that follow layout templates to containers, which also follow layout templates, thus creating a hierarchy of layout templates.

The appendix also presents an architecture and implementation for an extensible layout processor. The implementation has been thought so that new layout heuristics can be easily integrated. The layout processor is able to receive depth-based layout template specifications together with an input multimedia document, and to produce the final multimedia document targeting 3D displays. As a usage example, the high-level layout facilities have been integrated into the NCL

language. The NCL application resulted from the layout processor is able to run in an NCL player implementation that supports depth extensions (AZEVEDO; SOARES, 2013). Figure 44 shows an NCL application rendered on a 3D display, in which all aforementioned processes were performed.

So far, we have discussed depth-based layout templates in the scope of multimedia languages. However, an interesting use is to integrate them with document template languages, such as XTemplate (DOS SANTOS; MUCHALUAT-SAADE, 2012) and TAL (SOARES NETO, CARLOSDESALLES; SOARES; DE SOUZA, 2012). These languages allow for defining reusable temporal (and interactive) behavior in multimedia applications. The layout processor proposed in this appendix allows for implementing such integration in a two-step processing workflow (as illustrated in Figure 48). Such approach does not require any modification in either the layout template or document template processors. In the first step, the template processor receives the document template specification and a filling document (*e.g.*, defining the media objects that will be part of the final document). It then creates an intermediate document filling the document template's hot spots. In the second step, the layout processor receives the intermediate document and creates the final document, in which each media object has the positioning, size and depth specification in agreement with the layout template. In this two-step approach, the layout processor becomes independent of the document template authoring language in use.

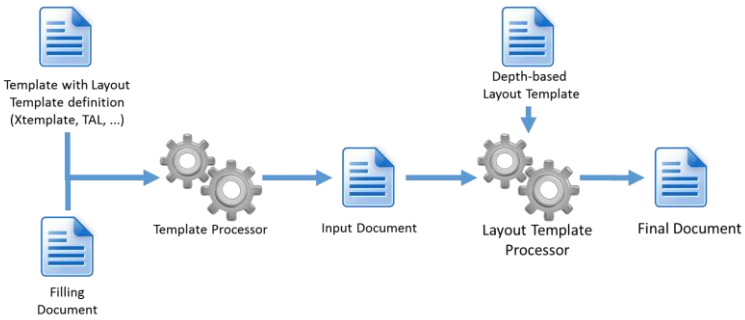


Figure 48 Two-step template processing.

The set of depth-based layouts proposed in this appendix is probably far from complete. As aforementioned, we are in the first steps towards a library of layout templates. Thus, an important future work is to continue searching for new appealing depth-based layout templates. In particular, we are looking for extending

our approach, which currently handles per-component depth, to allow per-pixel depth information; *i.e.*, to allow for the same media component having more than one depth information..

Evaluating the usability of the layout template examples was not among the goals of this appendix. Although they have been well-received by multimedia authors in our initial experiments, future work includes a qualitative research and comparison between depth-based layout templates and 2D-only layout templates, with end users using 3D displays.