

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



Captura e Visionamento de Vídeo 3D

João Pedro Alexandre Coelho

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Jorge Alves da Silva

Julho de 2012

© João Pedro Alexandre Coelho, 2012

Captura e Visionamento de Vídeo 3D

João Pedro Alexandre Coelho

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Cristina Ribeiro

Vogal Externo: José Manuel Torres

Orientador: Jorge Alves da Silva

23 de Julho de 2012

Resumo

O presente documento apresenta as principais conclusões de um projeto de investigação proposto pela MOG Technologies. O seu principal objetivo é desenvolver uma solução informática que permita capturar de uma câmara, e armazenar em ficheiro, sinal audiovisual 3D (com dois canais de vídeo). Por outro lado, é estudada uma forma de visualizar o vídeo, capturado em tempo real, de forma a permitir a perceção de profundidade por parte do observador. Neste documento são apresentadas técnicas para produzir vídeo anáglifo, pois tem a característica de causar esta ilusão de profundidade que é pretendida.

A presente investigação foi desenvolvida para no futuro melhorar a produção de programas televisivos 3D. Um dos problemas que a indústria se depara é a escassa oferta de soluções para armazenar vídeo 3D em ficheiro. Deve ser possível que os conteúdos, já guardados, possam ser modificados por *software* de edição de vídeo. Esta característica é vital para o sucesso do projeto, pois garante a interligação das várias etapas associadas à produção de programas para televisão.

Numa fase final do documento, é exposto um processo de integração das técnicas desenvolvidas com *software* já existente e em operação no mercado televisivo tradicional. O produto final, já integrado, tem como aspeto inovador permitir a produtores tradicionais que produzam também programas televisivos em 3D sem custos adicionais.

Keywords: *wrapper*, televisão 3D, produção para TV, anáglifo, GPU, MMX, SSE, mxfSPEEDRAIL S1000, DeckLink 3D, MXF, *asset* 3D, visualização de 3D

Abstract

This document presents the main findings of a research project proposed by MOG Technologies. Its main goal is to develop a software solution that allows the storing in file of 3D audiovisual signal (with two visual channels) that is being captured by a camera. Moreover, a way is devised to view the video, which is being captured in real time, allowing the perception of depth by the observer. This document also presents techniques for producing video anaglyphs because they have the characteristic to cause the illusion of depth that is desired.

This research is designed to improve the future production of television programs in 3D. One of the problems the industry faces is the scarcity of solutions for storing 3D video in file. It should be possible that content, already saved, could be modified by video editing software. This feature is vital to the success of the project as it ensures the interconnection of the various stages associated with the production of TV programs.

In a final stage of the document it is exposed a process for integrating the techniques developed with existing operating software, already available for the traditional TV market. The innovative aspect of the final integrated product would be the possibility for producers to also produce 3D television shows with no additional costs.

Índice

Introdução.....	1
1.1	Enquadramento..... 1
1.2	Motivação e Objectivos..... 2
1.3	Estrutura da Dissertação..... 3
Revisão Tecnológica	4
2.1	Visão Geral..... 4
2.2	Fluxos de trabalho – Televisão Tradicional 4
2.2.1	Aquisição/Captura de Vídeo 5
2.2.2	Encapsulamento e Armazenamento 5
2.2.3	Pós-produção..... 6
2.2.4	Controlo de Qualidade e Legendagem 6
2.2.5	Transmissão..... 6
2.3	Codificação de Vídeo 6
2.3.1	Formatos de Vídeo 7
2.4	Codificação de Áudio..... 8
2.5	<i>Ancillary Data</i> 8
2.6	<i>Wrappers</i> : o Formato MXF..... 9
2.6.1	Estrutura de um Ficheiro MXF 10
2.6.2	Padrões Operacionais 11
2.7	Hardware e <i>Software</i> Associados à Captura de Sinal..... 13
2.7.1	<i>Serial Digital Interface</i> 13
2.7.2	DeckLink..... 14
2.7.3	mxfsPEEDRAIL S1000..... 14
2.8	Metadados 17
2.8.1	mxfsPEEDRAIL S1000 – Soluções para Filme 3D 17
2.9	Estereoscopia em Televisão 18
2.10	Fluxos de Trabalho para Televisão 3D..... 19
2.10.1	Normas Emergentes para Encapsulamento de 3D em MXF 20
2.11	Formas de Visualizar 3D..... 21

2.11.1	Anáglifos.....	21
2.11.2	<i>Shutters</i> de Cristais Líquidos.....	24
2.11.3	Óculos 3D Polarizados.....	24
2.11.4	Capacete de Imersão.....	24
2.11.5	Dispositivos Autoestereoscópicos.....	25
2.12	Aceleração do Processamento de Vídeo.....	25
2.12.1	Computação Paralela.....	25
2.12.2	<i>Multithreading</i> de processador.....	26
2.12.3	<i>Graphics Processing Unit</i>	26
2.12.4	MMX.....	28
Especificação do Projeto.....		29
3.1	Visão Geral.....	29
3.2	Aquisição de Vídeo 3D.....	29
3.3	Encapsulamento em MXF.....	31
3.4	Visualização do Vídeo 3D.....	32
3.5	Integração no Programa mxfSPEEDRAIL S1000.....	32
3.6	Aproveitamento da GPU.....	33
3.7	Avaliação dos Resultados.....	33
3.8	Esquematização do Fluxo de Tarefas.....	34
Implementação da Solução.....		36
4.1	Visão Geral.....	36
4.2	Manipulação da DeckLink HD Extreme 3D.....	37
4.2.1	Aquisição de Vídeo 3D.....	37
4.2.2	Aquisição de Som.....	39
4.2.3	Aquisição de <i>Ancillary Data</i> e <i>Timecode</i>	39
4.3	Avid - Arquitetura de um <i>Asset</i> 3D.....	39
4.3.1	Vídeo 3D <i>Full Frame</i>	41
4.4	Produção de Anáglifos.....	45
4.4.1	Especificação YUV para Vídeo HD.....	46
4.4.2	Criação de Anáglifos em YUV.....	46
4.5	Anáglifos – Aceleração do Processamento.....	47
4.5.1	Simplificação das Fórmulas.....	47
4.5.2	Paralelização da Computação.....	48
4.6	Vídeo HD 3D Anáglifo Codificado em UYVY.....	50
4.7	Validação do <i>Software</i> Desenvolvido.....	52
4.7.1	Preparação dos Vídeos de Teste.....	52
4.7.2	Difusão do Vídeo 3D.....	53
Resultados e Discussão.....		56

5.1	Visão Geral.....	56
5.2	Desempenhos da GPU e da CPU	57
5.3	Desempenho do MMX	59
5.4	Processamento Combinado	60
5.4.1	GPU e <i>Multithreading</i>	60
5.4.2	MMX e <i>Multithreading</i>	61
5.4.3	GPU, MMX e <i>Multithreading</i>	62
5.5	Impacto da Otimização das Fórmulas de Criação de Anáglifos....	63
5.6	Qualidade dos Anáglifos Produzidos	63
5.7	Qualidade dos Ficheiros “MXF 3D” para Ambientes Avid.....	63
Integração com mxfsPEEDRAIL.....		64
6.1	Visão Geral.....	64
6.2	Módulo de Aquisição	64
6.3	Módulo de Encapsulamento	65
6.4	Módulo de Visualização.....	66
Conclusões e Trabalho Futuro.....		67
7.1	Trabalho Realizado	67
7.2	Trabalho Futuro.....	68
Referências.....		69

Lista de Figuras

Figura 1: Fluxos de Trabalho – Televisão Tradicional	5
Figura 2: Estrutura geral de um ficheiro MXF	10
Figura 3: Ficheiro MXF com várias partições	11
Figura 4: DeckLink HD Extreme 3D	14
Figura 5: mxfSPEEDRAIL S1000 - 1	15
Figura 6: mxfSPEEDRAIL S1000 - 2	16
Figura 7: mxfSPEEDRAIL S1000 - 3	17
Figura 8: Exemplo de um par estereoscópico	18
Figura 9: Representação dos campos de visão	19
Figura 10: Proposta de estrutura de ficheiro MXF para vídeo 3D	21
Figura 11: Processo de formação de um anáglifo	22
Figura 12: Fenómeno de ghosting numa imagem sem estereoscopia	23
Figura 13: Óculos Vermelho/Ciano	23
Figura 14: Óculos com <i>shutters</i>	24
Figura 15: Capacete de imersão	25
Figura 16: Organização estrutural de <i>threads</i> - CUDA	27
Figura 17: Representação dos vários registos MMX	28
Figura 18: Fluxo de Trabalho simplificado para captura de vídeo 3D	30
Figura 19: Diagrama de conexões da DeckLink 3D	30
Figura 20: Imagem 3D do tipo “frame compatible”	31
Figura 21: Organização estrutural de um bloco de pixeis UYVY	32
Figura 22: Diagrama referente ao fluxo das tarefas a ser executadas	35
Figura 23: Código para inicialização da DeckLink	37
Figura 24: Código de objeto do tipo <i>callback</i>	38
Figura 25: Código para iniciar aquisição de sinal	39
Figura 26: Excerto do resultado da análise de um ficheiro “Avid MXF 3D”	40
Figura 27: Fluxo de etapas para importação de um <i>asset</i> 3D para o Media Composer	41
Figura 28: Estrutura do MXF de um <i>asset</i> 3D - Avid	42
Figura 29: Vista de edição de um conteúdo em 3D – Media Composer	42
Figura 30: Representação em XML de uma CompositionMob	44

Figura 31: Visão geral de um <i>asset</i> 3D – Media Composer	44
Figura 32: Atributos de um conteúdo em 3D	45
Figura 33: Processo de produção de um anáglifo Vermelho/Ciano em RGB	45
Figura 34: Imagem a cores separada nas suas diferentes componentes YUV	46
Figura 35: Fórmulas de conversão de RGB/YUV e YUV/RGB	46
Figura 36: Esquema para produção de anáglifos em YUV	47
Figura 37: Fórmula para criação de anáglifos Vermelho/Ciano – BT.709 e BT.601	47
Figura 38: Fórmula otimizada para produção de anáglifos V./C. – BT.709 e BT.601	48
Figura 39: Aproximação inteira da fórmula da figura 38	48
Figura 40: Esquematização do processo de paralelização do processamento	49
Figura 41: Exemplo de uma operação de soma com tecnologia MMX	50
Figura 42: Aproximação inteira com <i>shifts</i> das equações de criação de anáglifos	50
Figura 43: Comparticipação dos componentes UYVY na produção de anáglifos V./C.	51
Figura 44: Exemplo de vídeo UYVY anáglifo	51
Figura 45: Processo de validação do <i>software</i>	52
Figura 46: Utilização do Blender para renderização de filme de animação em HD	53
Figura 47: Inicialização do modo de transmissão de sinal	54
Figura 48: Exemplo de código para definição do <i>handler</i> de saída de sinal	54
Figura 49: Definição de classe para suportar <i>frames</i> 3D	55
Figura 50: Exemplo de processo de validação do <i>software</i>	55
Figura 51: Desempenho da CPU, GPU e CPU+GPU	57
Figura 52: Efeitos do aumento do tamanho do vídeo de teste	59
Figura 53: GPU e <i>Multithreading</i> – Efeitos da variação da contribuição da CPU	61
Figura 54: MMX e <i>Multithreading</i> – Efeitos da variação da contribuição da CPU	62
Figura 55: Fórmula para estimação de tempos de execução com CPU, MMX e GPU	62
Figura 56: Exemplo de <i>pipeline</i> de <i>ingest</i> de vídeo 3D	66

Lista de Tabelas

Tabela 1: Padrões operacionais mais comuns em MXF

12

Abreviaturas e Símbolos

AAF	Advanced Authoring Format
AMWA	Advanced Media Workflow Association
API	Application Programming Interface
ATSC	Advanced Television Systems Committee
AVI	Audio Video Interleave
COM	Component Object Model
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
EDH	Error Detection and Handling
GPU	Graphics Processing Unit
HD	High Definition
IPTV	Internet Protocol Television
KLV	Key-Length-Value
LTC	Linear Timecode
MIMD	Multiple Instructions Multiple Data
MMX	Multimedia Extensions
MP4	MPEG-4 Part 14
MPEG	Moving Picture Experts Group
MXF	Material Exchange Format
PAL	Phase Alternating Line
PALPLUS	Phase Alternating Line Plus
PCM	Pulse-code Modulation
QA	Quality Assessment
RGB	Red Green Blue
S3D	Standard 3D
SD	Standard Definition
SDI	Serial Digital Interface
SDK	Software Development Kit
SIMD	Single Instruction Multiple Data
SMPTE	Society of Motion Picture and Television Engineers

SSE	Streaming SIMD Extensions
TDT	Televisão Digital Terrestre
UMID	Unique Material Identifier
UUID	Universally Unique Identifier
VITC	Vertical Interval Time Code
VPID	Video Payload Identifier
XML	Extensible Markup Language
ZDD	Zero Divergence Directive

Capítulo 1

Introdução

1.1 Enquadramento

A televisão 3D já é uma realidade nos dias de hoje, apesar de estar numa fase muito embrionária da sua existência. O recente sucesso de películas 3D para o cinema contribuiu em muito para o despertar do interesse do público por este tipo de produções audiovisuais. Quer no cinema quer em televisão houve uma tentativa para acompanhar o aumento da procura com uma maior oferta de produtos em 3D. Todavia, tem-se verificado que as várias etapas que envolvem a produção de conteúdos televisivos 3D não estão ainda convenientemente interligadas. O ponto crítico está num processo designado “aquisição de sinal”, que consiste na receção de vídeo estereoscópico proveniente da(s) câmara(s) e o seu armazenamento para posterior utilização em *software* de edição de vídeo.

O formato mais usado em produção para armazenar a informação audiovisual é o *Material Exchange Format* (MXF). Um programa televisivo é constituído por muitos tipos de dados, sejam eles visuais, sonoros, temporais (*timecode*), ou de outra ordem. O MXF surge para uniformizar o armazenamento, havendo a possibilidade de incluir vários arquivos independentes no mesmo ficheiro MXF. Este formato permite também a incorporação dos vários tipos de dados no mesmo ficheiro (vídeo, áudio, etc.), ou até de metadados que poderão corresponder a legendas de um filme ou outro tipo de informação anexa útil para o produtor [1].

A indústria ocupou-se de desenvolver *software* especializado em suprir as necessidades evidenciadas em cada etapa da produção televisiva. Empresas como a Avid ou a Adobe desenvolveram soluções informáticas para edição de vídeo. Num outro setor, a Panasonic e a JVC lançaram câmaras e formatos de vídeo para filmagem. Adicionalmente, empresas como a Blackmagic ou a Sony apostaram em tecnologias para fazer aquisição de sinal das câmaras. Por fim, devido ainda à necessidade de adquirir o sinal audiovisual proveniente das câmaras e de o armazenar em MXF, surgiram produtos como o mxfSPEEDRAIL S1000. Porém, e em

concordância com o que foi dito anteriormente, muitos destes chamados “sistemas de *ingest*” ainda não possuem a capacidade de lidar com produções audiovisuais em 3D. Esta lacuna está a criar dificuldades (técnicas e financeiras) aos produtores e, possivelmente, a desencorajá-los de investir no formato 3D para televisão.

1.2 Motivação e Objectivos

O principal propósito deste trabalho foi estudar abordagens e desenvolver soluções que permitissem reduzir este hiato verificado nos sistemas de *ingest* do formato 3D. Atualmente, para se produzir este tipo de conteúdos é necessário comprar equipamento adicional. Todo o *software* já adquirido pelas produtoras e que está em operação torna-se obsoleto. Comprar novo equipamento torna-se uma opção pouco viável. O objetivo foi, por isso, incorporar a funcionalidade 3D num sistema de *ingest* já existente, neste caso o mxfsPEEDRAIL S1000 da MOG Technologies. Desta forma, futuramente quem adquirir *software* de *ingest* terá também a oportunidade de produzir programas televisivos em 3D.

Dotar o S1000 de funcionalidades 3D significa desenvolver um módulo de aquisição de vídeo estereoscópico, áudio e outros dados provenientes do exterior. Significa também o encapsulamento da informação em MXF. O produto final irá certamente adicionar valor a empresas que desejem expandir os seus negócios através da conceção adicional de novos programas televisivos em 3D.

Outro objetivo do trabalho, mas não menos importante, foi produzir uma solução que permitisse ao operador do *software* visualizar o vídeo está a capturar no momento. Para isso, tem de ser possível apresentar no ecrã o filme 3D em tempo real. Para que não haja necessidade de adquirir equipamento especial de visualização de 3D, foram desenvolvidas técnicas para produção de anáglifos com base nos pares estereoscópicos recebidos. O vídeo anáglifo poderá ser reproduzido em monitores normais.

Em aplicações reais, o vídeo chega em tempo real e deve ser processado também em tempo real. Por esta razão também é vital que a solução seja suficientemente eficiente para não introduzir latências indesejadas no processamento. Por isso previu-se a necessidade de paralelizar a computação dos anáglifos.

A existência de vários objetivos pressupôs a sua priorização. De seguida, são apresentadas as prioridades estipuladas:

- Captura de som e vídeo 3D – muito importante
- Encapsulamento da informação em MXF – muito importante
- Produção de anáglifos em tempo real – importante
- Técnicas de paralelização da computação – importante
- Captura de metadados – pouco importante

1.3 Estrutura da Dissertação

Com a finalidade de apresentar de forma compreensível a investigação desenvolvida, o texto que se segue será dividido em 4 capítulos principais. O primeiro apresentará uma breve revisão do estado da técnica referente ao tema em estudo. De seguida, no capítulo 3, serão explicados os requisitos do projeto de forma mais detalhada. Seguir-se-á uma apresentação das conclusões mais importantes retiradas durante a fase de investigação. Serão identificados obstáculos que dificultaram o avanço dos trabalhos assim como as soluções encontradas. Seguidamente, no capítulo 5, serão expostos e analisados os resultados obtidos durante a fase de investigação. Será neste capítulo que será discutida a viabilidade das soluções encontradas. Por fim, um último capítulo irá apresentar as principais decisões tomadas para integrar o *software* desenvolvido, e comprovado, com o mxfsPEEDRAIL S1000.

Capítulo 2

Revisão Tecnológica

2.1 Visão Geral

Este capítulo apresenta o estado atual da arte no domínio da aquisição e visionamento de vídeo, sendo dada uma atenção especial a assuntos relacionados com televisão 3D. Apresentar-se-á uma esquematização das etapas presentes na produção de programas televisivos. De seguida, detalha-se que tecnologias são usadas para codificar e armazenar a informação audiovisual e, principalmente, como proceder para unificar o armazenamento. É dada uma especial atenção à descrição do formato MXF uma vez que será uma tecnologia basilar em todo este trabalho. São apresentados ainda conceitos relacionados com estereoscopia assim como tecnologias para visionamento de vídeo 3D disponíveis atualmente.

Os intervenientes na produção de conteúdos televisivos necessitam que o *software* opere satisfatoriamente em tempo real devido a variadíssimos motivos. Neste sentido o trabalho pressupõe uma grande preocupação com a eficiência e a rapidez da solução a desenvolver. O recurso a mecanismos para paralelizar o processamento torna-se desta forma espectável e por esse motivo também são analisados neste estado da arte conceitos do domínio da computação paralela.

2.2 Fluxos de trabalho – Televisão Tradicional

A produção de programas televisivos está dependente da aquisição de grandes quantidades de dados sejam eles visuais, auditivos ou de outro tipo. Esta informação é muitas vezes gerada por múltiplas câmaras que poderão estar posicionadas em localizações variadas. A recolha dos dados é frequentemente feita em tempo real, havendo uma grande necessidade de manter a

informação coerente e organizada. Neste sentido, a formalização das várias etapas e apresentação dos seus processos assume grande relevância. Tipicamente elas estão organizadas sequencialmente, formando fluxos de trabalho (figura 1).

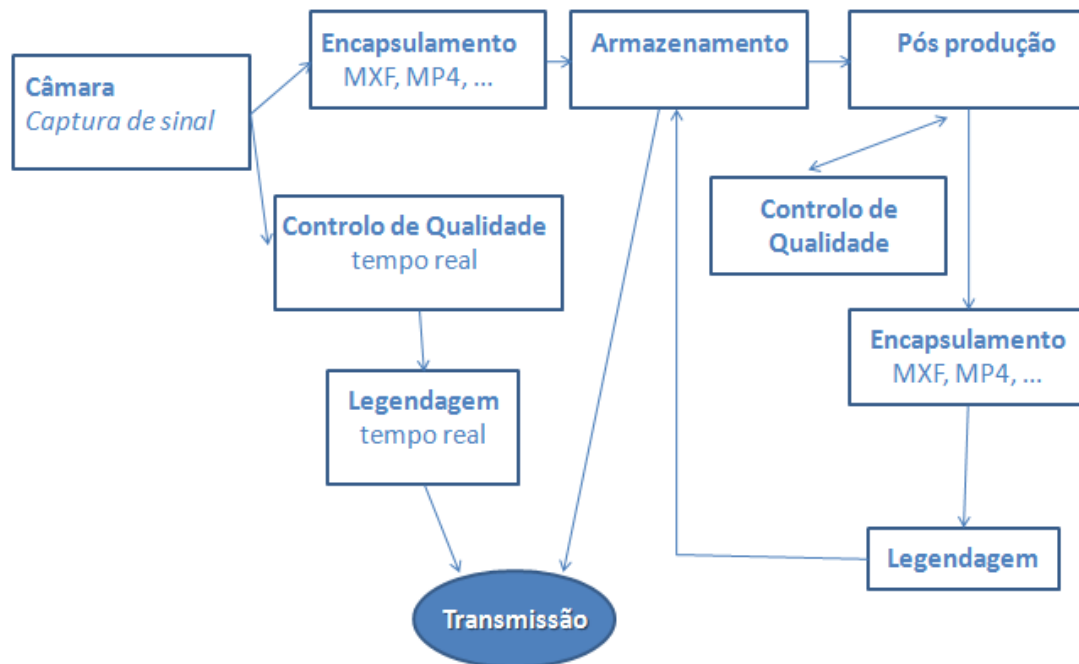


Figura 1: Fluxos de Trabalho – Televisão Tradicional [2]

2.2.1 Aquisição/Captura de Vídeo

Na atualidade, a aquisição de vídeo é feita, em geral, com recurso a câmaras digitais. A informação produzida pelas câmaras pode estar codificada num conjunto enorme de formatos. O sinal captado é então armazenado sendo posteriormente usado em pós-produção. Contudo, se a transmissão for feita em tempo real o sinal é simultaneamente difundido.

2.2.2 Encapsulamento e Armazenamento

Ao contrário do que acontecia no passado em que todo o filme ficava gravado numa só película, atualmente os conteúdos poderão estar dispersos em diversos ficheiros e formatos. É necessário normalizar o armazenamento, e por isso tem havido um grande investimento no desenvolvimento de tecnologias de encapsulamento (os chamados “*wrappers*”). Um *wrapper* é responsável por envolver os conteúdos audiovisuais num único formato de armazenamento. O MXF, apesar de não ser o único formato desenvolvido para o efeito é já uma norma

internacional, sendo líder no mercado da produção. Existe também o MP4, muito usado nos produtos da Apple e da Sony, que está massivamente presente no mercado da distribuição.

2.2.3 Pós-produção

Em pós-produção deve ser dada muita atenção à qualidade dos conteúdos. Se necessário, é nesta etapa que devem ser introduzidos no vídeo os efeitos digitais desejados, como a edição de *clips*, a eliminação de ruído ou a introdução de metadados, como por exemplo as legendas.

Os conteúdos audiovisuais requerem grandes capacidades de armazenamento. Com o aumento da resolução das imagens, a problemática tende a agravar-se. O *software* de edição de vídeo, ao trabalhar com enormes quantidades de informação, acaba por tornar-se mais lento levando a que as alterações se tornem morosas. Por esta razão, os produtores têm adotado a prática de realizar primeiramente as tarefas de pós-produção em exemplares de baixa resolução, sendo posteriormente feito o tratamento do filme de maior qualidade de forma automática. Este processo consiste na repetição das tarefas executadas no filme de menor resolução sobre as imagens equivalentes em alta resolução. Aos arquivos de elevada qualidade atribui-se frequentemente o termo “*HIGH-RES*” e aos de baixa definição “*PROXY*”.

2.2.4 Controlo de Qualidade e Legendagem

Em televisão o controlo de qualidade consiste em verificar se o filme que está a ser gravado não apresenta artefatos, como por exemplo ruído na imagem. A correção de conteúdos audiovisuais defeituosos demora algum tempo e por essa razão é de mais difícil realização em programas em direto. No sentido de mitigar estes problemas, atualmente as produtoras recorrem a *software* automatizado de controlo de qualidade. Uma ferramenta muito usada para o efeito é o programa “Certify”, desenvolvido pela Tektronix [2]. Pode ainda existir a necessidade de legendar os conteúdos. Esta tarefa ainda é feita por operadores humanos.

2.2.5 Transmissão

Durante esta fase, os conteúdos finais são transmitidos à audiência. À medida que o produto está a ser distribuído, principalmente se for em tempo real, deve ser feito um controlo de qualidade exaustivo para garantir que a informação audiovisual cumpre as exigências.

2.3 Codificação de Vídeo

Como foi dito anteriormente, o filme pode ser produzido de formas diferentes. Existem vários formatos disponíveis no mercado, apresentando cada um as suas especificidades e os seus

objetivos específicos. Por exemplo, há formatos específicos para serem usados na Web. Escolher o formato implica ter noção da qualidade máxima esperada, mas também é preciso ter noção do espaço necessário para armazenar o vídeo.

2.3.1 Formatos de Vídeo

Um dos primeiros esforços para uniformizar o armazenamento da informação nas câmaras foi iniciado em 1995 com a definição do formato DV [3]. Este corresponde a uma metodologia de compressão de vídeo que opera *frame a frame*, deixando o áudio sem qualquer tipo de compressão. Logo em 1995, a Panasonic propôs a sua implementação baseada no DV, o DVCPRO [3]. Ao longo dos anos, esta empresa foi introduzindo novas soluções baseadas no DVCPRO original. O último formato disponibilizado foi o DVCPRO HD que tem como características principais o suporte das resoluções 960x720 e 1440x1080 [3].

A Sony divulgou em 2003 a sua própria linha de produtos para gravação digital, o XDCAM [4]. Um dos mais utilizados actualmente é o XDCAM HD422, orientado para conteúdos de alta definição [5]. A Panasonic lançou ainda em 2007 o AVC-Intra, que tem como objectivo melhorar o armazenamento e posterior edição de conteúdos audiovisuais [6].

ESPAÇOS DE COR

As imagens de cor podem ser representadas por uma grande variedade de espaços de cor. O mais conhecido é o RGB, onde cada cor é representada por um vetor tridimensional que inclui valores de três componentes: vermelho, verde e azul. Apesar da popularidade do RGB, em televisão o espaço de cor mais utilizado é o YUV. Neste modelo, as cores são definidas por três valores: luminância (medida da intensidade da luz incidente num ponto), cromaticidade da gama dos azuis e cromaticidade da gama dos vermelhos.

COMPRESSÃO

A informação visual presente em cada fotograma pode estar num formato comprimido que confere ao vídeo um tamanho muito inferior ao da versão não comprimida. Há várias técnicas de compressão de vídeo, mas todas podem ser enquadradas em duas grandes categorias:

1. *Compressão sem perda de informação* – esta classe de métodos é semelhante aos métodos utilizados para comprimir outro tipo de informação. Não existe perda de qualidade, pelo que é possível reconstruir o sinal original a partir do arquivo comprimido. Tipicamente, não se consegue obter grandes taxas de compressão com base neste tipo de técnicas [7].
2. *Compressões com perda de informação* – frequentemente são desenvolvidas soluções que permitem a compressão com perda de alguma qualidade. Porém, com elevados níveis de compressão corre-se o risco de se tornar as imagens impercetíveis. Em certas

circunstâncias, faz de facto sentido aplicar compressão com perdas. O MPEG é um exemplo de um formato vídeo que aplica múltiplas técnicas compressivas com e sem perdas para reduzir o tamanho do vídeo [7].

2.4 Codificação de Áudio

O tipo de codificação áudio mais comum no meio televisivo é Pulse Code Modulation (PCM). A precisão dos valores amostrados e a própria frequência da amostragem são definidas por padrões internacionais. A especificação mais usada para codificação de som do tipo PCM é a SMPTE 299M. É usada para transmissão de áudio referente a conteúdos HD, o que significa que são usadas frequências de amostragem da ordem 48kHz, tendo os valores amostrados 24 bits de resolução.

O formato PCM permite a codificação de som proveniente de múltiplos pontos do espaço. Cada canal é independente dos outros. Os sistemas atuais de reprodução de áudio podem suportar até 24 canais distintos.

2.5 *Ancillary Data*

O conceito de *Ancillary Data* foi introduzido pela norma SMPTE 291M em 1998. Este tipo de dados representa toda a informação que não é vídeo. Por exemplo, som também é reconhecido como sendo *Ancillary Data*. Muitas vezes chamados de “dados ANC”, eles pertencem à categoria dos metadados. Estes últimos podem assumir muitas outras formas e ser usados para muitos contextos, que podem mesmo variar de conteúdo para conteúdo [8]. Por exemplo, num jogo de futebol, os instantes de ocorrência de golos e a identificação dos marcadores podem ser considerados metadados.

Os metadados podem estar situados em qualquer posição do fluxo de dados, desde que cumpra os requisitos definidos pela SMPTE. O principal e mais importante impõe que os *Ancillary Data* não podem ser colocados no interior da área ativa da imagem. Tipicamente, o que acontece é que as *frames* transmitidas contêm regiões em branco por cima e à esquerda da imagem. Este fenómeno deve-se em parte ao passado analógico da televisão, mas que pode ser aproveitado na era digital para incorporar informação extra que se considere relevante. Assim, surgiu uma prática comum que consiste na introdução de metadados numa das regiões livres. Consoante a localização dos mesmos, estes podem ser designados HANC (*Horizontal Ancillary Data*) ou VANC (*Vertical Ancillary Data*) [8].

2.6 *Wrappers*: o Formato MXF

Um programa televisivo está frequentemente dividido em *clips* de vídeo independentes. Esta divisão pode ser resultado do próprio processo de filmagem que pode ocorrer em dias diferentes, com múltiplas câmaras em simultâneo ou até porque possa fazer sentido conceptualmente separar os conteúdos filmados. Por outro lado, com o evoluir do mercado, foram surgindo formatos de vídeo diferentes. Neste contexto, começou a surgir a necessidade de se criar um tipo de ficheiro que permitisse uma maior uniformidade no armazenamento da informação, melhorando a forma como a esta é organizada. Foi assim que surgiu o conceito de *wrapper*. Não é suposto que um *wrapper* se comporte como um *codec* (que é responsável por codificar/descodificar os sinais recebidos de ou enviados para dispositivos eletrónicos) [1]. O objetivo de um *wrapper* é o de armazenar os dados recebidos preservando as várias fontes e formatos respetivos, para que mais tarde seja simples o acesso/edição da informação.

Assim se justifica a grande aceitação deste tipo de formatos, pois facilitam o armazenamento da informação e permitem que grande número de produtos intermédios gerados durante o processo de filmagem possam ser guardados/acedidos/manipulados de forma consertada.

Em televisão os *wrappers* mais utilizados são o MP4 (Quicktime) e o MXF. Enquanto o MP4 é mais utilizado no mercado da distribuição de vídeo, o MXF é muito utilizado no desenvolvimento de produtos audiovisuais. Uma grande vantagem oferecida pelo MXF é a possibilidade de incorporar num único ficheiro várias *Edit Decision Lists* (EDLs) [1], que consistem em listas ordenadas de *clips* e *timecode* que no seu conjunto definem o vídeo final. Este tipo de estrutura é muito apreciado pois facilita a pós-produção.

Apesar de todas as vantagens, é consensual que o MXF não resolve todos os problemas de interoperabilidade. Existem ainda lacunas relacionadas com formatos comprimidos assim como alguns tipos de metadados [9]. Esta dificuldade faz com que muitas estruturas de metadados sejam arquivadas num formato complementar ao MXF que se designa AAF (*Advanced Authoring Format*).

2.6.1 Estrutura de um Ficheiro MXF

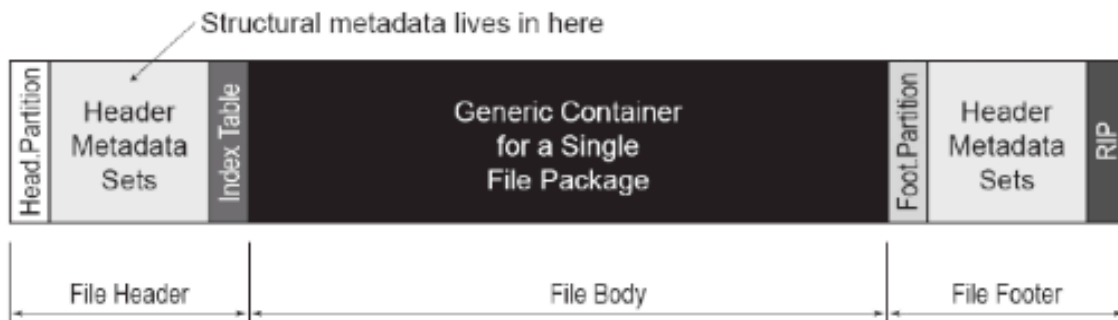


Figura 2: Estrutura geral de um ficheiro MXF [9]

Um documento MXF é tipicamente constituído pelos seguintes elementos (figura 2):

CABEÇALHO (*HEADER*)

Esta secção pode conter metadados, que são frequentemente divididos em duas categorias principais [10]:

1. *Metadados Estruturais* – informação referente à estrutura de um ficheiro MXF. A informação auxiliar é geralmente modelada segundo uma especificação orientada a objetos. Esta secção assume especial importância pois é nela que geralmente se incorpora informação que descreve a forma como os vários componentes presentes no ficheiro se relacionam ou até a sua origem. [11].
2. *Metadados Descritivos* – é a informação não audiovisual propriamente dita (por exemplo legendas).

Poderá ainda existir uma *Tabela de Indexação (Index Table)* que terá muita utilidade para acelerar o acesso a componentes presentes no ficheiro.

CORPO DO FICHEIRO (*BODY*)

O corpo do ficheiro poderá estar dividido em vários contentores de informação. O sistema foi desenhado para que os contentores sejam definidos como *plug-ins* independentes [12]. Desta forma, a arquitetura poderá ser especificada caso a caso e adaptada aos requisitos definidos (modelo do contentor genérico) [13].

Por outro lado, vários contentores podem estar agrupados em partições (figura 3), que servem para separar a informação em blocos lógicos. Cada bloco pode ser manipulado ou ignorado consoante a finalidade da aplicação. O objetivo final é o de permitir o acesso de forma aleatória ao conteúdo do ficheiro, no sentido em que aplicações que desejem aceder a um determinado tipo de informação não precisem de analisar outras partições que não sejam relevantes [14].

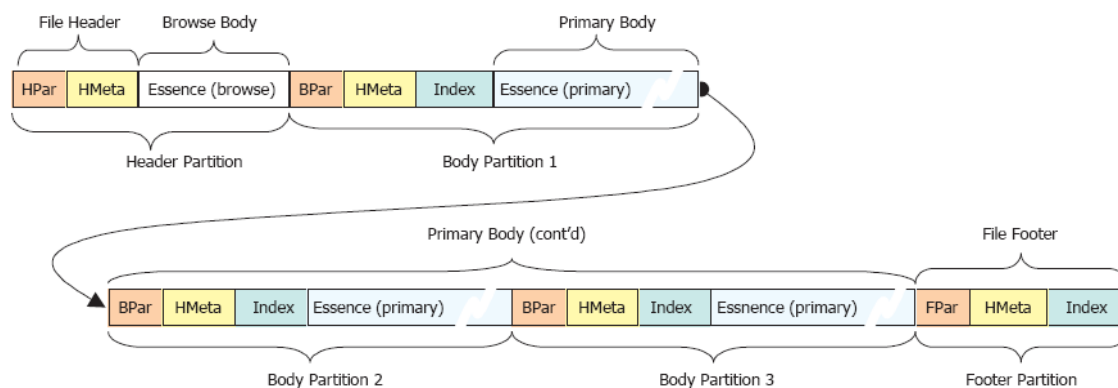


Figura 3: Ficheiro MXF com várias partições [9]

RODAPÉ (*FOOTER*)

Esta secção formaliza o fim do documento MXF, mas por vezes poderá apresentar informação extra, como a repetição de indexação ou metadados presentes no cabeçalho.

2.6.2 Padrões Operacionais

O MXF permite a definição de um conjunto de configurações diferentes para incorporar os dados. Porém, certas configurações são mais complexas que outras, pelo que podem não ser necessárias em casos mais simples. Desta forma, as entidades competentes decidiram apresentar um conjunto de padrões operacionais que visam organizar a informação audiovisual no ficheiro MXF segundo alguns parâmetros. A definição destes padrões não invalida a criação de abordagens específicas e alternativas.

Tipicamente, os padrões operacionais podem ser caracterizados da seguinte forma:

1. *Complexidade dos itens* presentes em cada contentor [9]. Neste sentido, cada contentor pode ser constituído por:
 - a. Um único item que é reproduzido na totalidade (*single item*);
 - b. Vários itens que são reproduzidos sequencialmente (*playlist items*);
 - c. Vários itens que podem ser editados e manipulados (*edit items*);
2. *Complexidade dos contentores* presentes no ficheiro MXF [9]. Num contentor, existem tendencialmente dois tipos de informação: o chamado *Material Package* que incorpora, entre outras coisas, referências temporais (também conhecidas por *tracks*). As *tracks* são adicionadas para que a essência¹ presente no *File Package* do

¹ Termo frequentemente utilizado como sinónimo de “conteúdos audiovisuais”.

contentor não perca sincronismo. Frequentemente, a complexidade do contentor é medida através da forma como se estabelecem as relações entre o *Material Package* e os *File Packages*:

- a. *Material Package* só permite o acesso a um *File Package* de cada vez (*Single Package*);
- b. *Material Package* permite o acesso a um ou mais *File Packages* de cada vez (*Ganged Packages*);
- c. Poderá haver vários *Material Packages* alternativos que podem aceder a um ou mais *File Packages* de cada vez (*Alternate Packages*).

Estes padrões são muitas vezes organizados de forma tabular (tabela 1) para uma melhor compreensão dos mecanismos que são apresentados. Um dos eixos representa a complexidade dos itens presentes nos *File Packages*, enquanto o outro descreve a complexidade dos contentores.

Tabela 1: Padrões operacionais mais comuns em MXF

Package Complexity	Item Complexity		
	Single Item 1	Play-list Items 2	Edit Items 3
Single Package A			
Ganged Packages B			
Alternate Packages C			

Nota: MP – Material Package; FP – File Package [9]

Em cada célula da tabela 1 está representado um padrão operacional específico, que tem como atributos a complexidade do(s) contentor(es) assim como o tipo de itens presentes em cada contentor. O OP1A [15], que é dos padrões operacionais mais utilizados, corresponde à

primeira célula (canto superior esquerdo). No fundo é constituído por uma arquitetura do tipo *single item* e *single package* [9].

2.7 Hardware e *Software* Associados à Captura de Sinal

No final do século XX, a captura de sinal ainda estava alicerçada num formato analógico. Com a progressiva introdução de dispositivos digitais, este processo passou a ser feito por equipamento informático especializado. Atualmente, já praticamente toda a produção de conteúdos audiovisuais é digitalizada.

Tipicamente, os conteúdos audiovisuais são transportados por um cabo SDI (*Serial Digital Interface*) e adquiridos por uma placa especializada, integrada num computador que executa *software* especializado. É o caso do *mxSPEEDRAIL S1000*, produzido pela MOG Technologies [16], que tem uma aceitação bastante grande a nível internacional.

2.7.1 *Serial Digital Interface*

O *Serial Digital Interface* (SDI) é muito usado na transmissão de dados não comprimidos e não encriptados. A normalização do formato está a cargo da SMPTE que é responsável pela divulgação de uma família de normas com base no formato original. De seguida apresentam-se algumas das especificações mais conhecidas do SDI.

SMPTE 259M

Esta norma é usada especialmente para formatos vídeo SD, tendo *bitrates* de 177Mbits/s, 143Mbits/s, 270Mbits/s e 360Mbits/s. As resoluções mais comuns que são bem suportadas por este formato são a 480i e a 576i.

SMPTE 292M

A normalização 292M foi criada para dar suporte ao crescimento da alta definição em vídeo. Os *bitrates* suportados são da ordem de 1.485Gbits/s.

SMPTE 372M

Este formato tem a especificidade de possuir dois canais independentes, sincronizados, que permitem a transmissão de informação paralelamente, de forma síncrona. As cadências são de 2.970Gbits/s.

ANCILLARY DATA EM SDI

O transporte de som e outros dados “ANC” (legendas, *timecode* ou de outro tipo) também é importante. Tipicamente, o áudio é do tipo PCM e vem sempre embebido com a especificação SDI, qualquer que ela seja. Existe um suporte generalizado para até 16

canais diferentes com uma frequência de amostragem de 48kHz. Os valores amostrados podem ser de 16 ou 24 bits.

2.7.2 DeckLink

Como foi dito anteriormente, é necessário que o computador possua uma placa específica para receber o sinal SDI resultante do processo de captura de vídeo. As placas DeckLink podem ser usadas para este fim [17]. Existem vários modelos de placas DeckLink, que de seguida são apresentados:

1. *DeckLink SDI* – placa simples e que permite captura de sinal SDI e HD-SDI [17].
2. *DeckLink Duo* – versão melhorada da anterior, com a adição de mais um canal para transmissão de informação de forma independente [17].
3. *DeckLink HD Extreme 3D* – placa com muitas funcionalidades, como entrada HDMI (figura 4). Além disso, é possível a receção de conteúdos 3D pois tem dois canais sincronizados que podem ser usados para transmitir pares estereoscópicos [17].



Figura 4: DeckLink HD Extreme 3D

2.7.3 mxfSPEEDRAIL S1000

Na viragem do século XX, novo *software* foi desenvolvido para permitir a correta aquisição de conteúdos audiovisuais e o seu armazenamento. Um bom exemplo disso é o S1000, desenvolvido pela MOG Technologies. A principal vantagem competitiva deste *software* está relacionada com a sua versatilidade em manipular dados em formatos diversos assim como a possibilidade de adição de metadados enquanto o sinal está a ser recebido.

Revisão Tecnológica

Este *software* permite a captura de sinal SDI ou HD-SDI através de uma placa DeckLink. Através de uma GUI bastante intuitiva é possível ao operador do sistema monitorizar o fluxo dos dados (figura 5), tendo desta forma a possibilidade de controlar a qualidade da essência [18].



Figura 5: mxfSPEEDRAIL S1000 - 1

Como referido anteriormente, a possibilidade de escolha do formato de vídeo que está a ser recebido via SDI é uma grande mais valia. Os protocolos de transmissão de informação suportados são o SMPTE 259M e o SMPTE 292M, com a possibilidade de utilização de 1 ou 2 canais independentes. O áudio suportado é do tipo PCM, com a possibilidade de receção até 12 canais diferentes [18].

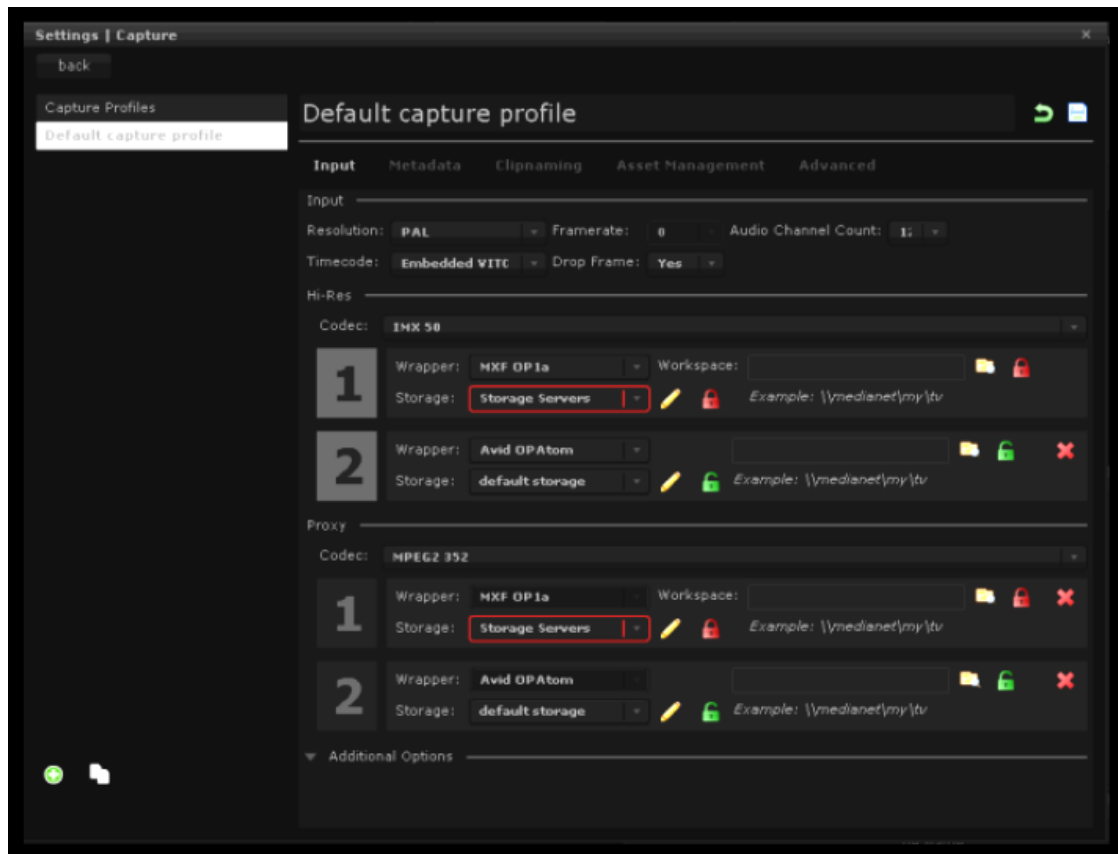


Figura 6: mxfsPEEDRAIL S1000 - 2

À medida que a essência vai sendo recebida no destino, o *software* é responsável por fazer a respetiva conversão/encapsulamento e posterior armazenamento no servidor. Uma opção interessante é a de gravação de duas versões do conteúdo em simultâneo: a *High-Res* e a *Proxy* [18] – ver figura 6. Existe a possibilidade de encapsular os dados num dos seguintes formatos MXF:

1. *Avid OPAtom* – formato usado em *software* de edição de vídeo da Avid (como o Media Composer);
2. *Sony OP1a* – formato usado em cenários de edição ou armazenamento de vídeo;
3. *Quicktime* – formato usado em ferramentas de edição de vídeo da Apple (como o Final Cut Pro).

2.8 Metadados

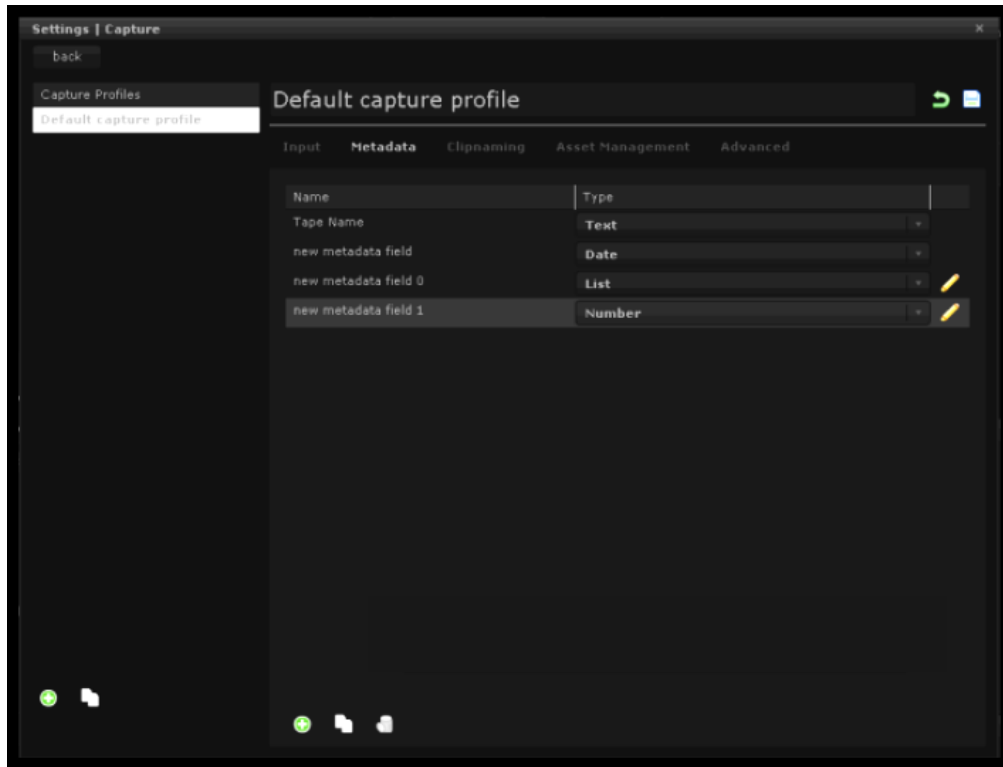


Figura 7: mxfsPEEDRAIL S1000 - 3

Também é possível a inclusão de metadados mais complexos no ficheiro MXF. O sistema permite a criação dinâmica de entidades e registos para assinalar informação variada que possa ser do interesse do produtor (figura 7). Estes registos serão guardados em conjunto com o filme. É ainda possível incluir *timecode*² na gravação e especificar a sua fonte [18].

2.8.1 mxfsPEEDRAIL S1000 – Soluções para Filme 3D

Tal como referido anteriormente, o S1000 não suporta captura de conteúdos 3D. A carência deve-se principalmente ao facto de não ter sido desenvolvido o módulo de encapsulamento de filme estereoscópico em MXF. Além do mais, também ainda não é possível ao operador do *software* a visualização dos anáglifos à medida que estes vão chegando. Deste modo, torna-se difícil para o operador fazer correções à convergência das câmaras caso o 3D obtido não seja agradável de ver.

² Termo frequentemente usado para descrever formas de etiquetar *frames* de vídeo com valores temporais. [44]

2.9 Estereoscopia em Televisão

O termo “visão estereoscópica” remete para a capacidade de um sistema (humano ou não) inferir, a partir de duas imagens obtidas de posições diferentes, informação 3D referente à estrutura e distância de objetos presentes numa cena [19].

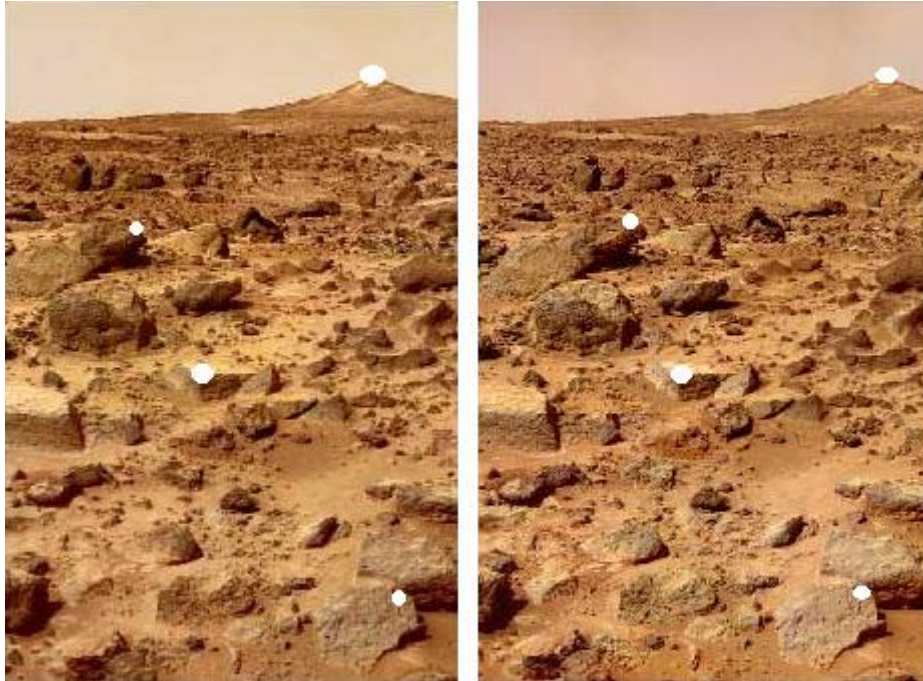


Figura 8: Exemplo de um par estereoscópico

Como se pode ver na imagem da figura 8, devido a uma mudança ligeira de perspectiva nas duas imagens adquiridas pelas câmaras, um ponto no espaço pode não aparecer na mesma posição em ambas as imagens. A esta diferença de posição dá-se o nome de disparidade. Se houver uma elevada disparidade entre pixéis correspondentes, o sistema visual sabe que eles correspondem a um ponto no espaço que está próximo do observador. Por outro lado, se a disparidade for baixa, então o ponto está mais afastado. Sistemas que usem as disparidades para inferir informação de profundidade só conseguem inferir distâncias relativas.

Neste sentido, em casos onde se pretende obter informação exata sobre distâncias, deve-se complementar a informação das disparidades com outro tipo de dados. Sabendo as características dos sensores visuais, passa a ser possível mapear uma distância relativa para uma distância absoluta. É por isso necessário determinar os parâmetros dos próprios sensores (por exemplo, das câmaras). Os mais importantes são os seguintes:

- Distância focal das câmaras;
- Distância entre câmaras (*baseline*);
- Ângulo de convergência dos eixos óticos das câmaras;

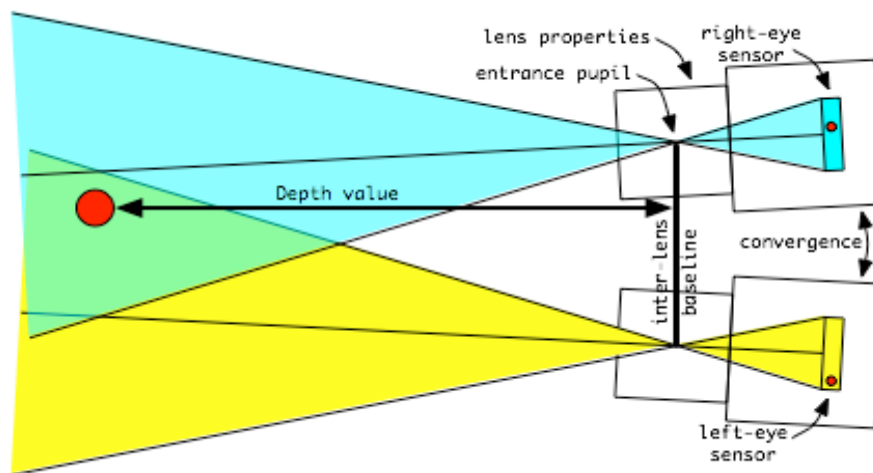


Figura 9: Representação dos campos de visão

Como evidenciado na figura 9, somente se consegue obter informação 3D por via estereoscópica na zona que corresponde à interseção dos campos de visão dos dois sensores. Consegue-se, através do ajuste do ângulo de convergência, aumentar o campo de visão em 3D. No entanto, um ângulo de convergência demasiado grande vai dificultar o processo de estabelecimento de correspondências entre as imagens, uma vez que a diferença de perspectiva vai fazer com que passem a existir pontos visíveis num sensor que não são visíveis no outro.

Em televisão há algum cuidado para que os parâmetros usados pelas câmaras 3D sejam semelhantes aos dos olhos humanos. Assim reduz-se a irritação causada no espetador devido a uma convergência ou *baseline* exageradas. O filme estereoscópico capturado irá então ser submetido a diversos processamentos antes de ser distribuído ao espetador. Nas secções seguintes serão apresentados brevemente os fluxos de trabalho associados à produção de conteúdos em 3D assim como formas de induzir a ilusão 3D num observador.

2.10 Fluxos de Trabalho para Televisão 3D

O desenvolvimento de soluções televisivas em 3D tem sido modesto apesar de existir alguma procura. Isto acontece por ser necessário modificar os fluxos referentes à televisão tradicional para que fiquem também adaptados à televisão 3D. Os processos para produção de conteúdos 3D são muito semelhantes aos tradicionais, mas com alterações específicas nas etapas referidas na secção 1 deste documento [2].

A aquisição do sinal deverá ser feita por equipamentos especiais, tipicamente duas câmaras de vídeo ou uma câmara com capacidade de capturar duas imagens em simultâneo. A informação capturada deve ser transmitida de forma sincronizada, pois se não acontecer, poderá haver perda de qualidade e o telespetador não conseguirá perceber a imagem 3D. Por outro

lado, é vital a monitorização constante dos ângulos de convergência dos pares estereoscópicos que vão sendo produzidos. Por vezes a percepção 3D pode tornar-se irritante para o utilizador caso os parâmetros de captura do filme (a convergência, por exemplo) não estejam bem adaptados à cena que se está a gravar. A deteção imediata de uma má convergência poderá ser rapidamente corrigida, adaptando a disposição das câmaras.

Um dos problemas com que a indústria se depara atualmente é a inexistência de uma norma que indique como o 3D deve encapsulado em MXF. Porém, já existe uma proposta de normalização que será apresentada adiante.

A pós-produção traz novos desafios. Uma das maiores preocupações dos produtores prende-se com a colocação de legendas em produtos audiovisuais em 3D. A posição e profundidade das legendas têm muita influência na qualidade do filme final. Negligenciar este processo pode levar a que o espectador tenha dificuldade em visualizar as legendas ou a profundidade dos objetos da cena. Desta forma, é importante que o operador faça adaptações e normalizações de níveis de profundidade (profundidade mínima e máxima da estrutura da cena) sempre que necessário. Por outro lado, a captura de vídeo por vezes traz artefactos visuais decorrentes de iluminação especular ou texturas presentes nos objetos. Deve existir também uma preocupação de verificar se os artefactos existentes são consistentes nas duas imagens pois, não sendo, também pode conduzir a grande perda de qualidade do conteúdo. A inclusão das legendas ainda tem de ser feita manualmente por não haver *software* disponível que consiga determinar automaticamente a profundidade ótima das legendas.

Ao nível da transmissão existem ainda lacunas por resolver. O *software* de controlo de qualidade ainda não possui capacidade para analisar a qualidade do vídeo 3D [2].

2.10.1 Normas Emergentes para Encapsulamento de 3D em MXF

A inexistência de uma norma para utilizar o MXF como meio de encapsulamento de vídeo 3D é um impedimento para a produção consistente de programas televisivos. Foi criado um grupo de trabalho (“*SMPTE 35PM40 WG on 3D Home Master*”) cujo objectivo é precisamente desenvolver um formato MXF basilar para a televisão 3D. Está previsto que tenha suporte para conteúdos audiovisuais e metadados (incluindo legendas) [20].

Este grupo de trabalho já formalizou uma proposta para estruturar um ficheiro “MXF 3D”. A estrutura proposta assenta no entrelaçamento de *frames* de ambos os olhos. Também está previsto que cada *frame* seja referenciada numa tabela de indexação. Como se pode verificar na figura 10, existe ainda um espaço para dados do tipo VANC (*Vertical Ancillary Data*) [21].

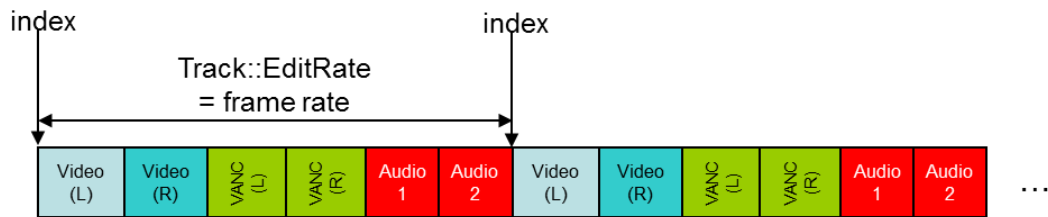


Figura 10: Proposta de estrutura de ficheiro MXF para vídeo 3D

2.11 Formas de Visualizar 3D

A visualização de vídeo 3D pode tornar-se um desafio técnico, pois é necessário “enganar” o sistema visual humano fazendo-o “acreditar” que está a perceber uma determinada estrutura 3D de uma cena quando na realidade não passa de uma ilusão.

Existem várias formas de criar esta ilusão estereoscópica no espetador, apesar de todas se basearem no mesmo princípio: canalizar uma imagem diferente para cada olho. A eficácia de cada técnica está intimamente relacionada com a qualidade do filme 3D propriamente dito. De seguida apresentam-se algumas das soluções mais conhecidas.

2.11.1 Anáglifos

Uma das primeiras técnicas utilizadas para visualizar imagens 3D consiste na fusão de duas imagens pertencentes a um par estereoscópico numa única (processo ilustrado na figura 11). Este tipo de imagem é frequentemente designado anáglifo. Com o auxílio de óculos com lentes de colorações diferentes é possível filtrar certas gamas de cores e transmitir a cada olho somente uma das imagens.

Esta abordagem é reconhecidamente pouco eficaz, conduzindo a perda de qualidade na cor [22], pois a aplicação dos filtros, além de filtrar a imagem, acaba por também retirar alguma informação também. Porém, atualmente há muitas aplicações desta técnica, por ser barata e não haver necessidade de adquirir monitores especiais para perceber efeitos 3D.

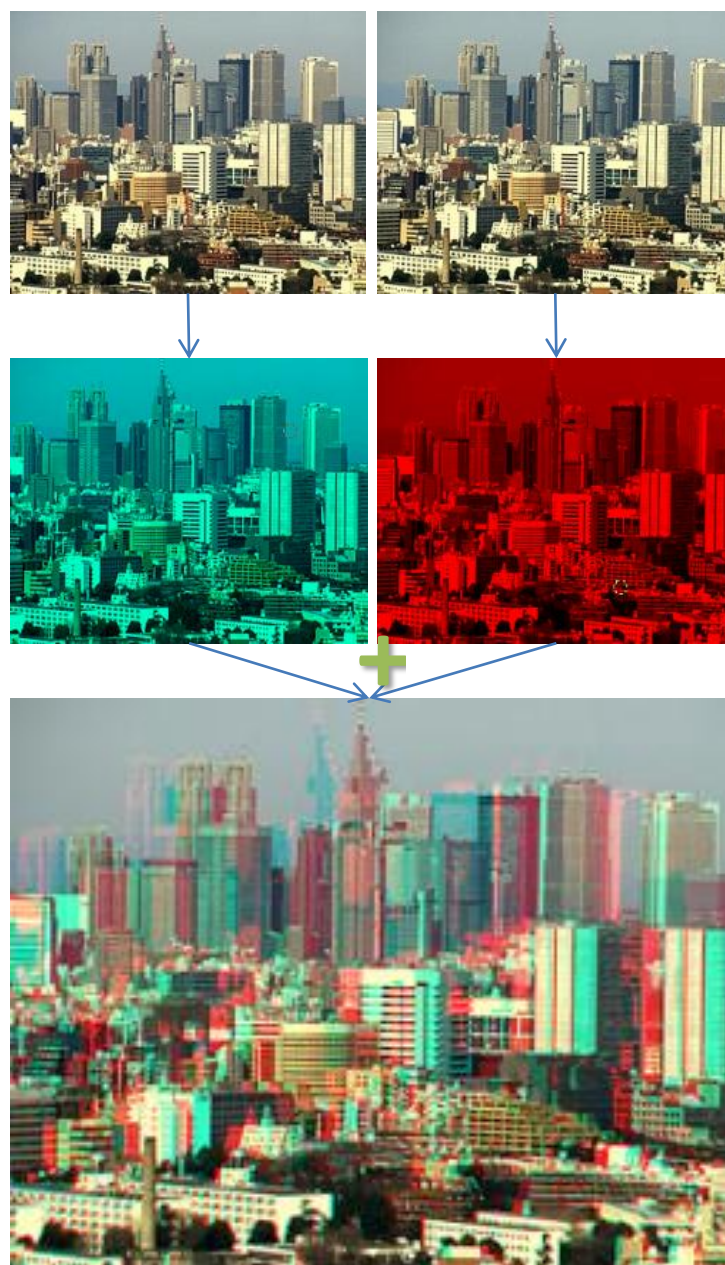


Figura 11: Processo de formação de um anáglifo

Uma das decisões mais importantes a tomar quando se deseja produzir anáglifos é a escolha do par de cores que deve servir de filtro nas lentes. Um dos requisitos principais é que as cores escolhidas estejam tão pouco sobrepostas quanto possível. A sobreposição de partes do espectro das cores faz com que elementos da imagem que não deviam ser filtrados numa lente o sejam, dando origem a um fenómeno chamado *ghosting*. É um efeito que, tal como o nome indica, tende a duplicar objetos cujas colorações estejam dentro da gama de colorações sobrepostas (figura 12).



Figura 12: Fenómeno de ghosting numa imagem sem estereoscopia

Ao longo dos anos foram sendo propostas várias combinações de filtros:

Vermelho-Verde – verificou-se que existe algum *ghosting* em monitores de televisão, apesar de ser uma boa opção para materiais impressos.

Vermelho-Azul – esta combinação oferece pouco *ghosting* em monitores, apesar de ser pouco usada.

Vermelho-Ciano – apesar de não ser a melhor, esta solução é a mais utilizada (figura 13), muito devido à não existência de patente para produção de óculos neste formato. A perceção da imagem sofre alguma perda de qualidade e pode ocorrer também o fenómeno de *ghosting*.



Figura 13: Óculos Vermelho/Ciano

A forma como os monitores emitem a radiação que nos faz perceber uma determinada cor também tem influência no *ghosting* percebido, uma vez que a onda gerada por um determinado pixel poderá ter partes do seu espectro que façam sobreposição com a onda produzida por outro pixel. De facto, existem flutuações no efeito de *ghosting* quando comparados vários tipos de monitores (LCD, CRT ou LED) [23].

A produção dos anáglifos depende muito do espaço de cor que se está a utilizar e dos filtros a implementar. Por exemplo, a aplicação de uma filtragem do tipo Vermelho/Ciano é relativamente simples no espaço RGB, uma vez que a componente de vermelhos está separada no primeiro canal da imagem (R), e o ciano corresponde aos outros dois (G, B). Para fundir duas imagens numa só, basta simplesmente aproveitar o canal R da imagem relativa ao olho direito e os canais G e B da imagem do olho esquerdo [24].

2.11.2 *Shutters* de Cristais Líquidos

Neste tipo de tecnologia, o ecrã transmite duas imagens (uma para o olho esquerdo e outra para o olho direito). O objetivo dos óculos consiste em bloquear a visão de um dos olhos, de forma alternada, permitindo que a imagem que está a ser transmitida seja captada somente pelo olho correto [25].

Como se pode ver na imagem 14, é necessário que os óculos tenham embutido algum tipo de dispositivo eletrónico que permita o sincronismo com o ecrã [26].



Figura 14: Óculos com *shutters*

2.11.3 Óculos 3D Polarizados

Esta técnica aproveita a possibilidade de polarização da luz [27] para apresentar as duas imagens em simultâneo ao espectador. Este só precisa de ter uns óculos capazes de separar as duas imagens, pois estes têm cada lente filtra uma determinada polarização. O único requisito que esta abordagem acarreta é a utilização de dois projetores que projetem dois filmes em simultâneo na mesma tela. Esta tecnologia é actualmente a mais utilizada no cinema devido aos seus baixos custos.

2.11.4 Capacete de Imersão

É mais usado no domínio da realidade virtual. Consiste em dois ecrãs LCD ou OLED colocados em frente aos olhos (figura 15). Tipicamente estes dispositivos trazem incorporado um detetor de movimentos da cabeça muito útil em contextos de realidade virtual ou aumentada.



Figura 15: Capacete de imersão

2.11.5 Dispositivos Autoestereoscópicos

Recentemente têm surgido novos dispositivos que dispensam o uso de óculos [28]. É o caso das televisões 3D ou da Nintendo DS. Contudo, a desvantagem deste tipo de tecnologia é a perda de percepção 3D se o espectador se desviar muito lateralmente relativamente ao centro do monitor.

2.12 Aceleração do Processamento de Vídeo

Por ser computacionalmente pesado, o processamento de vídeo muitas vezes requer o maior aproveitamento possível dos recursos computacionais disponíveis. Paralelizar o processamento é uma das formas de melhorar a rapidez de um programa. Existem algumas formas de o fazer num computador normal:

- Execução de várias *threads* de CPU;
- Utilização da GPU (*Graphics Processing Unit*);
- Utilização de instruções MMX.

2.12.1 Computação Paralela

Resolver um problema com computação paralela significa analisar um problema e dividi-lo em pequenas tarefas que possam ser resolvidas separadamente (*particionamento*). Por fim, as tarefas devem ser mapeadas para os respetivos núcleos de processamento. Estes últimos, por sua vez, irão resolvê-las paralelamente [29].

A forma como o programador concebe o programa depende da arquitetura de computação paralela implementada pelo sistema. Existem duas classes principais: a MIMD (*multiple instruction multiple data*) e a SIMD (*single instruction multiple data*). A primeira, tal como o

nome indica, possibilita a execução de pedaços de código diferentes em cada processador. A segunda só permite a execução de um programa específico mas num conjunto de dados diferentes para cada núcleo de processamento [29].

2.12.2 *Multithreading* de processador

As arquiteturas *multithreading* permitem bifurcar a execução de código. Este tipo de abordagem computacional, apesar de não paralelizar realmente a computação, é vantajosa pois possibilita, entre outras coisas, que a CPU continue a efetuar cálculos aritméticos referentes a uma *thread*, enquanto outra aguarda a execução de uma instrução de acesso a memória. Em sistemas com múltiplos *cores*, é inclusivamente possível que sejam executadas simultaneamente tantas *threads* quanto o número de *cores* existentes no processador.

Por outro lado, ao contrário do que acontece com sistemas multi-processo, que podem não partilhar recursos, todas as *threads* de um programa têm acesso aos mesmos recursos de sistema (por exemplo, à memória) [34]. Esta característica é vantajosa pois permite que elas comuniquem entre si para, por exemplo, partilhar resultados de processamentos ou sincronizar tarefas. Por outro lado, a partilha de memória tem a desvantagem de aumentar drasticamente o risco de problemas de concorrência, devido a acessos em simultâneo das mesmas posições de memória.

2.12.3 *Graphics Processing Unit*

A *Graphics Processing Unit* (GPU) não substituirá nunca a CPU de um computador por não conseguir lidar de forma tão eficiente com todos os tipos de operações possíveis (nomeadamente saltos condicionais). Mas, apesar de tudo, a unidade de processamento gráfico afigura-se uma opção quando se necessitar de resolver problemas que sejam decomponíveis em tarefas mais pequenas e cuja solução seja essencialmente numérica. A sua utilização é cada vez mais frequente, muito devido aos requisitos cada vez mais exigentes das aplicações.

Tipicamente, quando se fala de processamento na GPU são frequentemente referidos dois termos:

- **Kernel (ou procedimento de kernel)** : corresponde a um pedaço de código que é executado na GPU [30];
- **Thread**: corresponde a um fluxo na execução do código. A maioria das GPUs consegue ter múltiplas *threads* em execução simultaneamente.

Um dos maiores fabricantes de GPUs é a NVIDIA. O mercado das GPU evoluiu muito e atualmente toda a tecnologia NVIDIA traz acoplado um modelo unificado de computação paralela, o CUDA (*Compute Unified Device Architecture*). Como implementa uma arquitetura

do tipo SIMD [31], as várias *threads* executam o mesmo *kernel*, ainda que com dados diferentes.

A interface de desenvolvimento disponibilizada permite uma fácil programação da placa gráfica via C++ (ou outra linguagem). Outra vantagem do CUDA é a possibilidade de adaptar automaticamente o programa à capacidade instalada no dispositivo gráfico, permitindo mais

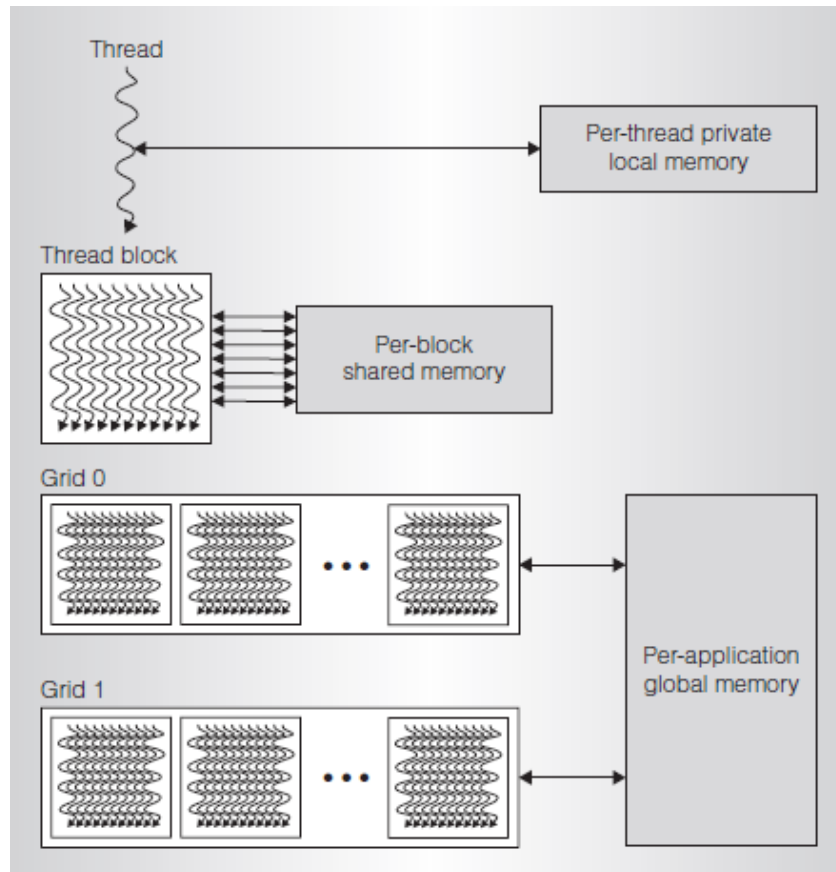


Figura 16: Organização estrutural de *threads* - CUDA [31]

paralelismo em placas mais potentes e restringindo-o nas restantes outras.

Como se pode observar na figura 16, em CUDA as *threads* são organizadas em blocos que partilham memória, tornando mais fácil o sincronismo da computação assim como a partilha de dados. Por outro lado, vários blocos podem agrupar-se em *grids* que têm a propriedade de ter acesso à memória global da aplicação.

Uma outra tecnologia disponível para desenvolver programas com algum grau de paralelismo é o OpenCL. É uma ferramenta de acesso livre criada primeiramente para lidar com paralelismo ao nível do processador. Porém, é possível também desenvolver aplicações para a GPU. Uma característica importante do OpenCL é a sua portabilidade [32]. Código inicialmente escrito para paralelizar tarefas na CPU pode ser facilmente convertido para ser executado na GPU. Por outro lado, uma desvantagem que deve ser referida é que programas escritos em OpenCL são mais lentos do que programas escritos em CUDA. A tecnologia da NVIDIA é

mais rápida a transferir blocos de *bytes* para a memória interna da GPU assim como a executar os *kernels* [33]. Por esta razão, se se pretender desenvolver um programa genérico e não dependente de nenhum fabricante, é preferível a utilização de OpenCL. Por outro lado, se o principal objetivo for a rapidez da execução, CUDA parece ser a melhor opção.

2.12.4 MMX

A tecnologia MMX permite a compactação de várias variáveis num único registo (de dimensão estendida) e a sua utilização para efetuar simultaneamente o mesmo cálculo nas várias variáveis (arquitetura SIMD). Esta tecnologia foi inicialmente introduzida pela Intel em 1996. E, como se pode ver na figura 17, disponibilizava 8 registos diferentes (MM0-MM7) que possuíam uma dimensão de 64 *bits*.

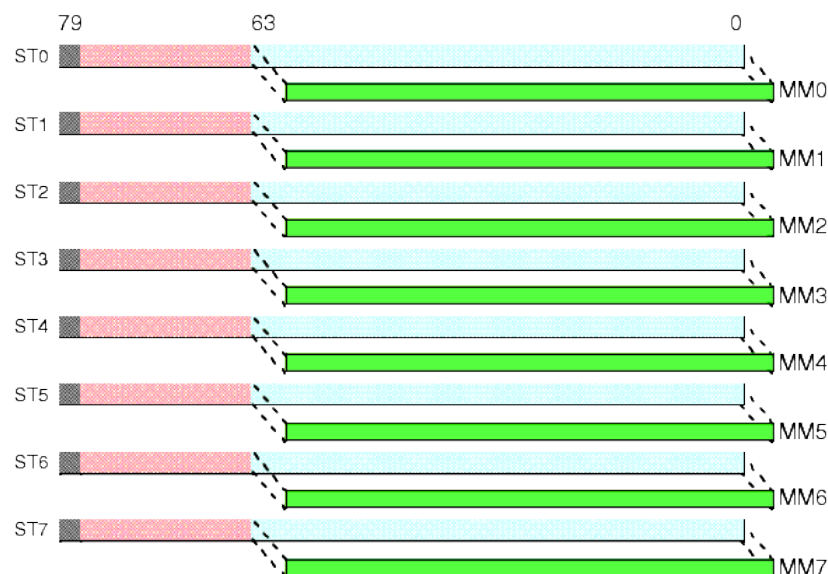


Figura 17: Representação dos vários registos MMX [45]

Mais recentemente foi lançada uma extensão do MMX, o SSE, que tem a particularidade de estender ainda mais a dimensão dos registos. É agora possível empacotar variáveis até perfazerem 128 *bits*. O SSE também possui uma interface de desenvolvimento em C++ que permite a utilização desta tecnologia de forma mais fácil e intuitiva [35].

Capítulo 3

Especificação do Projeto

3.1 Visão Geral

A abordagem que se optou por seguir pressupôs a divisão do trabalho em tarefas relativamente independentes. Para cada uma definiu-se um conjunto de requisitos que deveriam cumprir. Por fim, integrou-se de todo o trabalho, nunca esquecendo a documentação e validação da qualidade dos resultados. O projeto foi então dividido nas seguintes componentes:

1. Aquisição de vídeo 3D e correspondentes metadados via SDI;
2. Encapsulamento dos conteúdos em MXF;
3. Produção de anáglifos a partir de pares estereoscópicos;
4. Melhoramento da etapa 3): aproveitamento da GPU para otimizar recursos computacionais;
5. Integração das soluções informáticas com o *software* mxfSPEEDRAIL S1000.

3.2 Aquisição de Vídeo 3D

Foi recomendado que a captura de sinal fosse feita com auxílio da placa Decklink HD Extreme 3D, da empresa Blackmagic Design (processo ilustrado na figura 18).

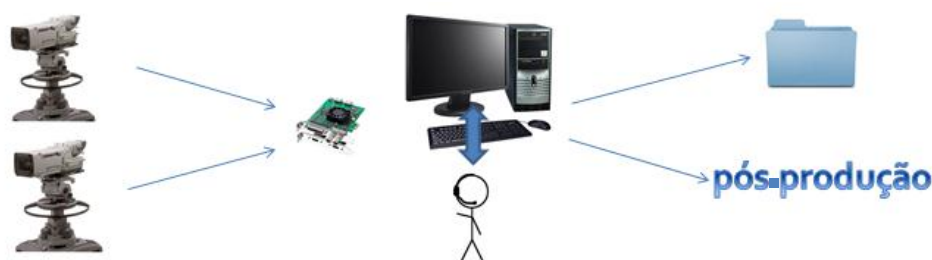


Figura 18: Fluxo de Trabalho simplificado para captura de vídeo 3D

Um requisito importante era que os dois canais de vídeo independentes fossem adquiridos de forma sincronizada e por isso a placa deveria ter características especiais para garantir este mesmo sincronismo. No caso da Decklink Extreme 3D (ver figura 19), estas características já são implementadas pelo fabricante e por essa razão a placa foi uma boa opção para utilizar no projeto.

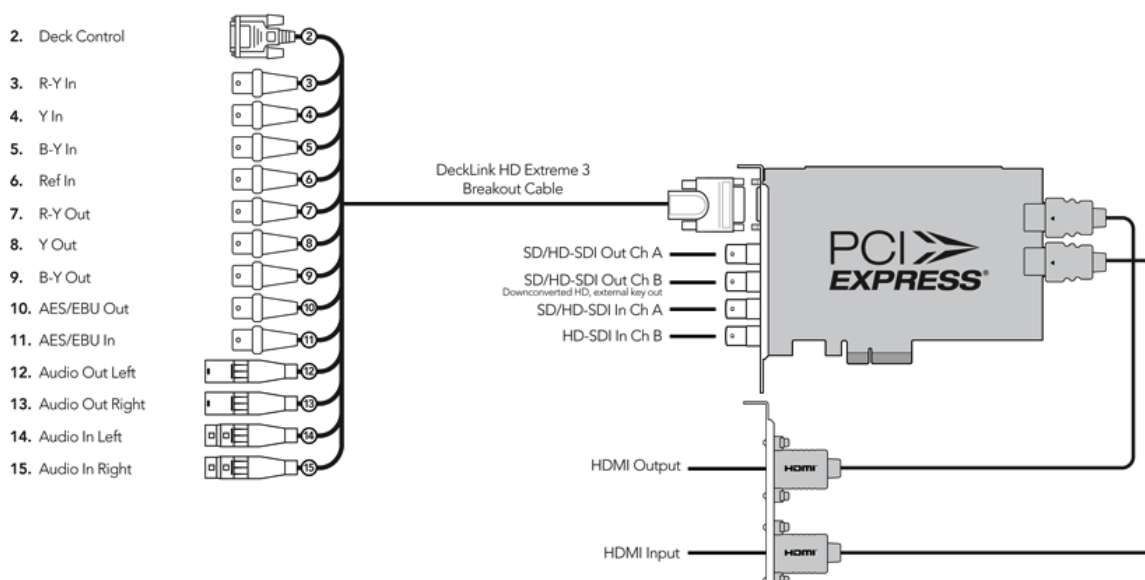


Figura 19: Diagrama de conexões da DeckLink 3D

Uma outra razão para a escolha da DeckLink 3D por parte dos proponentes do projeto é o facto de esta trazer associado um SDK bastante útil, constituído por uma API bem documentada e de fácil aprendizagem, além de um conjunto amplo de exemplos que permitem uma fácil exploração das capacidades da placa.

3.3 Encapsulamento em MXF

Ainda não existe uma normalização internacional para efetuar o encapsulamento em MXF de vídeo 3D. Já existem algumas propostas mas nada foi ainda oficializado. Uma delas é exemplificada de forma simplificada no diagrama que é apresentado na figura 9. Contudo, por muito viável que a solução anteriormente apresentada seja, ela só é útil se for utilizada na prática pelas várias produtoras de programas televisivos. E a realidade é que como ainda não está a ser utilizada na prática, ela perdeu algum interesse para os proponentes do projeto. Eles optaram por identificar que especificações “MXF-3D”, não normalizadas, a indústria utiliza atualmente. Por exemplo, a multinacional Avid já suporta nos seus produtos dois tipos de especificações MXF próprias para vídeo 3D:

- **Full frame**, quando o vídeo estereoscópico é armazenado na sua resolução de captura original (por exemplo 1080p).
- **Frame compatible**, quando o vídeo estereoscópico é comprimido para que um par estereoscópico ocupe o mesmo espaço que num vídeo sem estereoscopia com a mesma resolução. Tipicamente, há várias formas de comprimir o vídeo, mas todas elas requerem que a resolução final do vídeo seja reduzida para metade da resolução do filme original (figura 20).



Figura 20: Imagem 3D do tipo “frame compatible”

A forma como a essência é armazenada no ficheiro MXF difere para cada um destas especificações. Desta forma, a MOG propôs que se investigasse, através de técnicas de engenharia reversa, como é que a Avid gera estes ficheiros MXF e de seguida se desenvolvesse um módulo em C++ que fosse bem sucedido a fazer a transcodificação de um vídeo capturado para MXF suportado pela Avid. Para este projeto, recomendou-se que fosse dada total atenção ao MXF para conteúdos do tipo *full frame*.

3.4 Visualização do Vídeo 3D

A solução que foi recomendada pelos proponentes para visualizar o vídeo estereoscópico consiste em fundir as sequências de imagens do olho esquerdo e do olho direito em anáglifos. Houve a indicação para que fossem estudadas formas de produzir anáglifos, havendo o cuidado por descobrir quais as melhores cores a usar como filtros. Um outro requisito era que o vídeo fosse mostrado no monitor sem atrasos significativos que prejudicassem o trabalho do operador da estação de captura.

A placa Decklink reconhece blocos de *pixels* que estejam codificados em YUV ou RGB. Apesar de tudo, os produtores preferem codificar os seus vídeos em YUV, pois é uma representação mais próxima da que é usada em ecrãs. Por essa razão se deve configurar a Decklink para receber em YUV e produzir os anáglifos diretamente para este espaço de cor. A tarefa complica-se um pouco mais com o facto de os produtores utilizarem uma variante comprimida do YUV, o UYVY.



Figura 21: Organização estrutural de um bloco de pixeis UYVY

Como é exemplificado na figura 21, a compressão afeta as crominâncias. Em vez de cada pixel ser representado por um triplete de *bytes*, cada dois pixeis adjacentes partilham os mesmos valores de crominância, havendo somente a variação da luminância. Desta forma, em média, cada pixel consegue ser representado só com dois *bytes* (em vez de três). Dada esta pequena explicação adicional é de salientar que, como o UYVY é muito usado em televisão houve a necessidade de introduzir um requisito adicional: o módulo de produção de anáglifos deveria operar no espaço de cor UYVY.

3.5 Integração no Programa mxfSPEEDRAIL S1000

A integração de todo o *software* produzido com as soluções da empresa era um requisito muito importante. Isso significaria tornar o código compatível com a arquitetura do mxfSPEEDRAIL S1000 e redesenhar localmente alguns módulos do próprio S1000 sempre que se justificasse. Foram inicialmente previstas algumas modificações importantes a efetuar ao S1000 previamente existente (podendo haver a possibilidade de incluir outras futuramente):

- Produzir versão com 64 *bits*, o que permite alocar maiores quantidades de memória;
- Redimensionar *buffers* internos para suportarem canais de dados extra:
 1. Para armazenar canal extra do vídeo 3D que é capturado;

2. Para armazenar canais resultantes de processamento interno, como por exemplo o vídeo em modo anáglifo;
- Adaptar os módulos de *wrapping* de dados para permitirem o encapsulamento dos conteúdos para MXF suportado pela Avid;
- Adicionar novos componentes à interface gráfica relacionadas com a captura 3D:
 1. Escolha do formato de captura (se é estereoscópico com dois canais vídeo ou não);
 2. Escolha do formato MXF de destino;
 3. Escolha do modo de visualização do vídeo 3D.

3.6 Aproveitamento da GPU

Apesar de ser facultativa a utilização da GPU no mxfsPEEDRAIL, recomendou-se que fosse feito um levantamento dos benefícios que esta ferramenta possibilita. Por exemplo, uma das métricas a analisar deve ser o tempo de execução de um algoritmo (neste caso para produzir anáglifos), executado só na CPU (aproveitando *multithreading* ou MMX), só na GPU, ou na CPU e GPU em simultâneo (balanceando o processamento em cada unidade).

3.7 Avaliação dos Resultados

De forma a garantir a qualidade do *software* produzido, foi pedido que fossem desenvolvidos (sempre que possível) testes unitários. Por outro lado, no caso da aquisição do sinal e encapsulamento em MXF, dever-se-iam simular casos reais de funcionamento do mxfsPEEDRAIL, dando como *input* vídeos conhecidos e avaliando a receção na placa e a transcodificação subsequentes.

Visto que não é fácil quantificar a qualidade dos anáglifos de forma automática, a validação dos algoritmos desenvolvidos para esta fase deveria ser feita qualitativamente por parte de utilizadores escolhidos especificamente para o efeito. No âmbito do projeto, poderiam ser profissionais da MOG Technologies.

Relativamente ao desempenho da abordagem computacional com auxílio da GPU, a métrica para avaliar o sucesso da solução seria o tempo de execução do algoritmo. Os valores obtidos deveriam ser analisados comparativamente para decidir qual a melhor abordagem.

Por outro lado, no sentido de verificar que o código está a cumprir com as especificações, foi requisitado que fosse implementada uma bateria de testes unitários.

3.8 Esquematização do Fluxo de Tarefas

Na figura 22 pode-se ver o esquema do plano de trabalhos. Existiu uma preocupação por definir dependências entre tarefas. Assim foi mais fácil saber por onde começar o desenvolvimento, pois não se corria o risco de deixar uma determinada tarefa dependente de outra que ainda não tinha sido iniciada. Por outro lado, houve um cuidado em separar logicamente as tarefas. Desta forma, foi possível ter uma melhor noção das competências que seria preciso desenvolver em cada grupo. Por exemplo, num dos grupos seria necessário aprofundar conhecimentos relacionados com MXF (grupo azul), enquanto que noutro deveria ser dada maior atenção a assuntos relacionados com a aquisição de sinal audiovisual (grupo verde). Em algumas situações verificou-se ainda que havia dependência mútua entre duas tarefas, como por exemplo no processo de produzir e testar ficheiros “MXF 3D”. Este fenómeno deveu-se ao facto de não se saber exactamente a arquitetura do ficheiro e conduziu por isso à necessidade de abordagem mais iterativa para resolver o problema.

Por outro lado, foi é de realçar que foi alocada uma grande quantidade de tempo (não representada no diagrama da figura 22) para apreender as tecnologias utilizadas pela empresa. Entre elas destacam-se todos os módulos que fazem parte do mxfsPEEDRAIL S1000, assim como todas as ferramentas e metodologias auxiliares para criação de projetos, compilação de código, controlo de versões, etc..

Especificação do Projeto

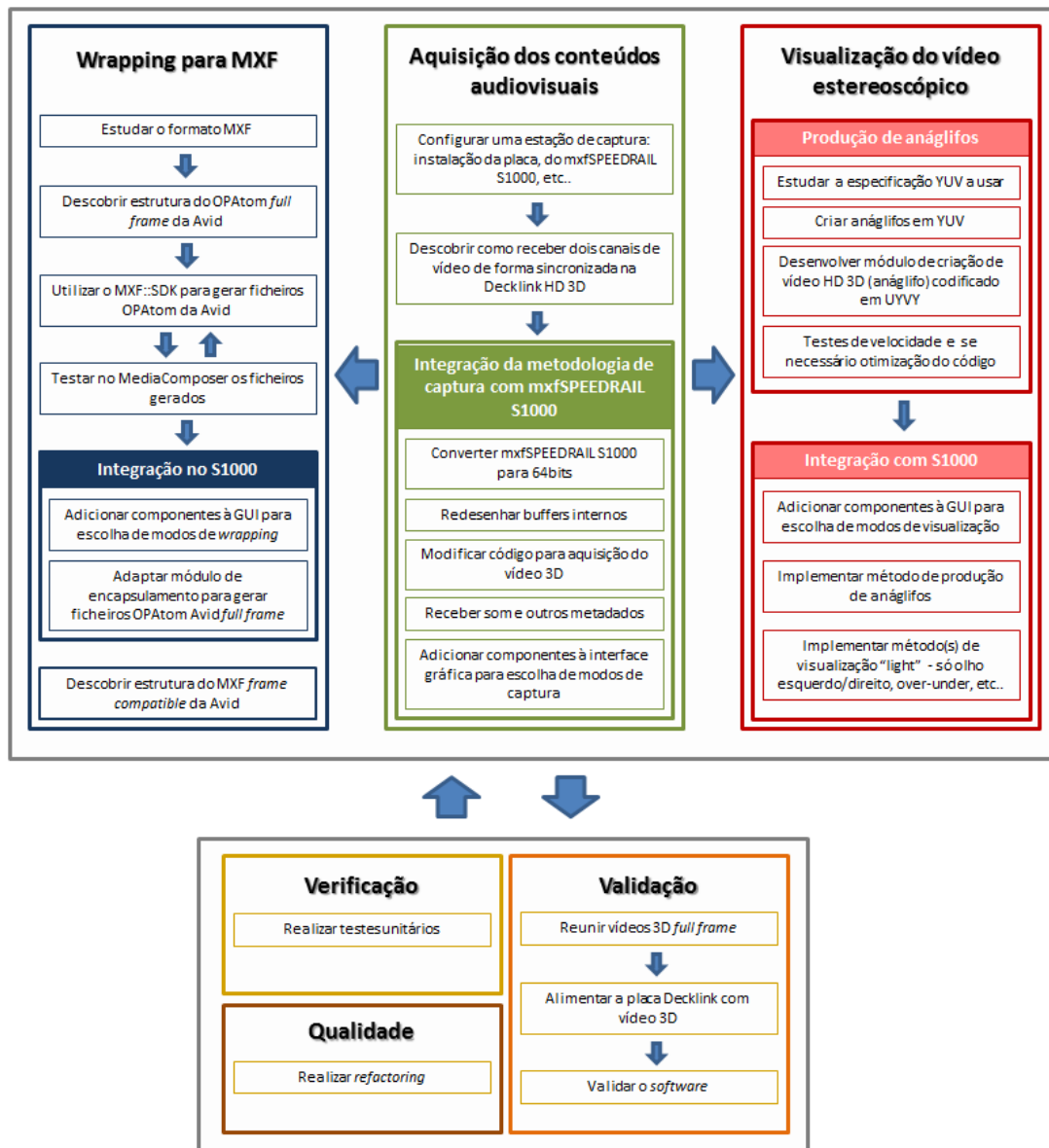


Figura 22: Diagrama referente ao fluxo das tarefas a ser executadas

Capítulo 4

Implementação da Solução

4.1 Visão Geral

Através da análise dos requisitos, explicitados em pormenor no capítulo anterior, foi possível estipular que problemas fundamentais era necessário resolver:

- Como utilizar o SDK de desenvolvimento da DeckLink para adquirir vídeo, áudio e metadados?
- Qual a arquitetura de um conteúdo audiovisual em 3D no Avid Media Composer?
- Como desenvolver um módulo que permita o encapsulamento para “Avid MXF” de conteúdos audiovisuais?
- Como produzir anáglifos em UYVY?
- Como desenvolver um módulo de produção de anáglifos que permita o processamento de *frames* estereoscópicas em tempo real?
- Como validar o *software* desenvolvido?

Todas estes problemas foram resolvidos. As soluções encontram-se explicadas pormenorizadamente nas secções seguintes.

4.2 Manipulação da DeckLink HD Extreme 3D

Para se poder trabalhar com a(s) placa(s) é necessário instanciar, através de COM³, um iterador de uma lista que, ao ser percorrida, permitirá o acesso individual a cada placa. É possível aceder aos seus atributos e identificar qual delas possui características 3D [36].

Cada DeckLink é acedida através de um objeto do tipo “IDeckLink”. Este estende um conjunto de interfaces que são responsáveis por definir um protocolo para a manipulação dos diversos dispositivos presentes na placa. São exemplos de alguns deles os sistemas de aquisição e envio de sinal [36].

O acesso aos controladores dos dispositivos pode ser feito através do método “QueryInterface”. Como é exemplificado na figura 23, este método é executado para obter dois controladores (“IDeckLinkInput” e “IDeckLinkOutput”) [36]. Como se vai poder verificar adiante, eles serão muito importantes para executar a receção e difusão do sinal 3D.

```
// Extrair do ficheiro idl o header da API da Decklink
#include "DeckLinkAPI.h"

// ...

unsigned int MAX_DECKLINK = 16;
CComPtr<IDeckLinkIterator> pIterator = NULL;

IDeckLink* m_pDeckLink = new IDeckLink[MAX_DECKLINK];
IDeckLinkInput* m_pInputCard;
IDeckLinkOutput* m_pOutputCard;

// Obter iterador para as placas DeckLink
if (CoCreateInstance(CLSID_CDeckLinkIterator, NULL, CLSCTX_ALL, IID_IDeckLinkIterator, (void**) &pIterator) == S_OK
    && pIterator)
{
    for (unsigned int i=0; i<MAX_DECKLINK; i++)
    {
        // Percorrer as placas instaladas no computador
        if (pIterator->Next(&m_pDeckLink[i]) != S_OK)
            break;

        // Obter os dispositivos de receção de sinal (SDI-IN neste caso) de uma placa
        m_pDeckLink[i]->QueryInterface(IID_IDeckLinkInput, (void**) &m_pInputCard);

        // Obter os dispositivos de envio de sinal (SDI_OUT neste caso) de uma placa
        m_pDeckLink[i]->QueryInterface(IID_IDeckLinkOutput, (void**) &m_pOutputCard);
    }
}

// ...
```

Figura 23: Código para inicialização da DeckLink

4.2.1 Aquisição de Vídeo 3D

A arquitetura da Decklink permite o registo de *callbacks* que são executadas sempre que uma *frame* de vídeo seja adquirida pela placa. Para efetuar o registo basta definir um objeto que implemente a interface “IDeckLinkInputCallback” [36]. Um dos métodos mais importantes a implementar é o “VideoInputFrameArrived”, que é executado sempre que é desencadeado um

³ “Component Object Model (COM) é uma plataforma da Microsoft para componentes de software lançada em 1993. Ela é usada para permitir a comunicação entre processos e a criação dinâmica de objetos em qualquer linguagem de programação que suporte a tecnologia.”

evento de recepção de uma nova *frame*. Como se pode ver na figura 24, este método é responsável por receber dois objetos como argumento: um do tipo “IDeckLinkVideoInputFrame”, que incorpora um *buffer* de pixels codificados em YUV ou RGB, e outro do tipo “IDeckLinkAudioInputPacket” que por sua vez inclui um *buffer* de amostras de áudio.

Na realidade, o “IDeckLinkVideoInputFrame” inclui duas *frames*, uma para o olho esquerdo e outra para o olho direito [36]. Contudo, a segunda não está diretamente acessível ao programador, uma vez que a *callback* também é usada em contextos em que o conteúdo só possua um canal de vídeo. É necessário por isso definir uma extensão do tipo “IDeckLinkVideoFrame3DExtensions” e passá-la como argumento ao método “QueryInterface” da *frame* recebida. Assim que essa etapa esteja concluída, o objeto estendido passa a conseguir devolver a segunda *frame* através do método “GetFrameForRightEye” [36].

```
class CVideoDelegate : public IDeckLinkInputCallback
{
private:
    // ...

public:
    CVideoDelegate ();

    // Fazer override aos outros métodos
    // ...

    virtual HRESULT STDMETHODCALLTYPE VideoInputFrameArrived( IDeckLinkVideoInputFrame* inFrame, IDeckLinkAudioInputPacket*)
    {
        // Obter extensão do objeto referente ao segundo canal
        IDeckLinkVideoFrame3DExtensions *p3DEctension = NULL;
        HRESULT rst = inFrame -> QueryInterface(IID_IDeckLinkVideoFrame3DExtensions, (void**)&p3DEctension);

        if (rst != S_OK)
        {
            DEBUG("Erro a obter interface 3D.");
            return ~S_OK;
        }

        // Obter frame do canal 2 (olho direito)
        IDeckLinkVideoFrame *pRightEyeFrame;
        rst = p3DEctension -> GetFrameForRightEye(&pRightEyeFrame);

        if (rst != S_OK)
        {
            DEBUG("Erro a obter o frame do canal 2.");
            return ~S_OK;
        }

        // ...

        // Libertar memória
        pRightEyeFrame -> Release();
        inFrame -> Release();
    }
};
```

Figura 24: Código de objeto do tipo *callback*

Tendo sido definido o *handler* para receber os conteúdos, é necessário inicializá-lo. Como se pode ver pela figura 25, recorrendo à função “SetCallBack”, é possível registar no controlador de entrada/saída o “objeto *callback*” que se definiu anteriormente [36].

Por outro lado, tem que se definir o formato do vídeo e áudio assim como habilitar a recepção de vídeo 3D (através das funções “EnableVideoInput” e “EnableAudioInput”). Sem esta última habilitação na *callback* não será possível registar a extensão

Implementação da Solução

“IDeckLinkVideoFrame3DExtensions” e obter o segundo canal de vídeo. Por fim, para confirmar o início do processo de aquisição, deve ser executado o método “StartStreams” [36].

```
HRESULT      theResult;

// Registrar callback de que é chamada sempre que uma frame chega
CVideoDelegate* m_pDelegate = new CVideoDelegate();
theResult = m_pInputCard->SetCallback(m_pDelegate);
if (theResult != S_OK)
    DEBUG("O registo da callback falhou: 0x%08x\n", theResult);

// Configurar o formato de receção
// > resolução (p.e. 1920x1080)
// > codificação da cor (p.e. YUV)
// > outros parâmetros (p.e. se o vídeo é 3D ou não)
BMDDisplayMode displayMode = bmdModeHD1080i50;
theResult = m_pInputCard->EnableVideoInput(displayMode, bmdFormat8BitYUV, bmdVideoInputDualStream3D);
if (theResult != S_OK)
    DEBUG("EnableVideoInput falhou: 0x%08x\n", theResult);

// Iniciar a receção dos dados
theResult = m_pInputCard->StartStreams();
if (theResult != S_OK)
    DEBUG("Input StartStreams falhou: %08x\n", theResult);
```

Figura 25: Código para iniciar aquisição de sinal

4.2.2 Aquisição de Som

A aquisição do som pode ser feita na mesma *callback* que o vídeo, uma vez que a função recebe sempre um parâmetro que corresponde a um pacote de *bytes* PCM.

4.2.3 Aquisição de *Ancillary Data* e *Timecode*

Os dados do tipo ANC, como o próprio conceito sugere, vêm embebidos em cada *frame*. Para lhes aceder basta chamar o método “GetAncillaryData” da *frame* passada à *callback* [36]. De notar que num ambiente 3D, é possível enviar *Ancillary Data* no canal esquerdo, no direito ou em ambos. Ao programar o *software* de aquisição, deve-se ter em atenção a qual das *frames* se está a aceder. O processo de recolha de *timecode* está acessível através do método “GetTimecode” da *frame* em questão [36].

4.3 Avid - Arquitetura de um *Asset* 3D

O Avid Media Composer consegue manipular *assets*⁴ em 3D. E para o fazer recorre ao MXF para embeber o vídeo e o áudio e ao AAF (*Advanced Authoring Format*)⁵ para guardar definições importantes.

⁴ Em multimédia, um *asset* é tipicamente um conjunto de diversas essências (áudio, vídeo, metadatos, etc..) inter dependentes. *Software* de edição de vídeo importa o *asset* de uma só vez e não cada essência em separado.

Antes de se conseguir escrever ficheiros “MXF Avid” foi crucial descobrir qual a sua arquitetura. A interpretação de um ficheiro MXF é feita com auxílio de ferramentas desenvolvidas pela MOG, especificamente para o efeito. Isto acontece porque não se consegue simplesmente abrir um ficheiro deste tipo e ler o seu conteúdo (como acontece com o XML ou o HTML). Na realidade, os objetos guardados são identificados por sequências de *bytes* sem significado aparente. É necessário conhecer o significado dos conjuntos de *bytes*. Este trabalho é feito pelas tais ferramentas especializadas em interpretar MXF. A MOG possui um conjunto delas que foram muito úteis para o projeto. A principal, designada “MXF_SDK_MetadataSetParser”, é responsável por identificar a estrutura de alto nível de um ficheiro MXF assim como apresentar uma representação dos metadados presentes no ficheiro em análise (ver figura 26):

```
0x1998B 06.0e.2b.34.02.53.01.01.0d.01.01.01.01.01.36.00
<Name: Material Package>
- 3c.0a : (0x10) Instance UID
  UID 94.b2.5f.b1.f7.7b.48.80.b7.f2.fb.a9.b0.f1.be.bf
- ff.f8 : (0x01) UnknownLocalTag_07
  UInt8 0
- ff.f9 : (0x28) AvidMetadataItemArray
  Array
    StrongReference 42.65.c7.73.07.c7.4d.78.97.a4.19.7b.62.73.69.10
    StrongReference 22.05.9e.85.de.ef.46.e1.b6.8d.ff.b6.46.39.4f.27
- 44.06 : (0x48) AvidMaterialPackageMetadataItemArray
  Array
    StrongReference 39.d0.cd.7b.35.2c.4c.05.9a.56.bd.45.6e.96.0c.4d
    StrongReference ec.13.a8.c1.81.b7.49.58.8c.00.42.ee.99.b8.a9.90
    StrongReference 8c.51.96.e0.18.ac.45.75.97.38.ee.bd.c1.26.2c.12
    StrongReference 83.ba.cf.ba.e8.08.4a.03.9a.49.ed.bb.3e.87.4e.f7
- 44.03 : (0x18) Tracks
  Array
    StrongReference 4b.0e.54.25.4d.8e.47.6d.9c.a9.28.2e.d3.ab.86.20
- 44.04 : (0x08) Package Modified Date
  Timestamp 2012/02/17 11:34:41,000
- 44.05 : (0x08) Package Creation Date
  Timestamp 2012/02/17 11:34:23,000
- 44.02 : (0x2E) Name
  UTF16String >boatardeMONO2_R.new.01<
- 44.01 : (0x20) Package UID
  UMID32 06.0a.2b.34.01.01.01.01.0f.00.13.00.00.00.4f.3e.3b.3f.9f.41.06.e9.06.0e.2b.34.7f.7f.2a.80

0x19ABF 06.0e.2b.34.02.53.01.01.0d.01.01.01.01.01.3f.00
<Name: AvidMetadataItem>
- 3c.0a : (0x10) Instance UID
  UID 42.65.c7.73.07.c7.4d.78.97.a4.19.7b.62.73.69.10
- 50.03 : (0x31) AvidMetadataAttributeValue
  DataStream
    0: 4C 00 02 10 01 00 00 00 00 06 0E 2B 34 01 04 01 L.....+4...
    16: 01 4D 00 4F 00 4E 00 4F 00 43 00 55 00 4C 00 41 .M.O.N.O.C.U.L.A
    32: 00 52 00 5F 00 52 00 49 00 47 00 48 00 54 00 00 .R.._.R.I.G.H.T..
    48: 00
- 50.01 : (0x1A) AvidMetadataAttributeName
  UTF16String >_S3D_CHANNEL<
```

Figura 26: Excerto do resultado da análise de um ficheiro “Avid MXF 3D”

O exemplo da figura 26 mostra a definição de um *Material Package* (conceito introduzido na secção 2.6.2). Todos os objetos possuem UUIDs⁶ que identificam unicamente o objeto em questão. Por outro lado, é raro um objeto estar isolado. Na realidade, cada um possui muito frequentemente referências (“StrongReference”) para outros. No exemplo da figura 26, verifica-se que o *Material Package* possui referências para atributos importantes (neste caso, o “_S3D_CHANNEL”).

⁵ Desenvolvido pela AMWA, o AAF é um formato que permite o intercâmbio orientado a objetos de informação relativa a conteúdos audiovisuais.

⁶ Um UUID é um número de 16 *bytes* gerado aleatoriamente e que é usado para identificar, de forma única, objetos presentes num ficheiro MXF.

Implementação da Solução

A Avid optou ainda por incluir definições extra sobre um *asset* 3D num ficheiro AAF externo. Estas definições são por norma referentes ao modo como o Media Composer vai mostrar o conteúdo em 3D ao operador. Por essa razão, não faz tanto sentido incluir esses dados nos ficheiros MXF, pois eles poderão ser usados noutros contextos que não envolvam o Media Composer.



Figura 27: Fluxo de etapas para importação de um *asset* 3D para o Media Composer

O Media Composer trabalha com o padrão operacional OPAtom, o que significa que cada essência (áudio ou vídeo) deve ficar armazenada num ficheiro diferente (figura 27). Assim, por exemplo, se um produtor tiver um conteúdo audiovisual constituído por 8 canais de áudio e 1 canal de vídeo, o *asset* reconhecido pelo Media Composer deverá ser constituído por 8 ficheiros MXF referentes ao áudio e 1 referente ao vídeo. Isto acontece por razões de rapidez. Como a informação está separada, o *Media Composer* não precisa de multiplexar os dados das várias essências, no mesmo ficheiro, quando se fazem edições ao *asset*.

4.3.1 Vídeo 3D Full Frame

Para se conseguir estudar a arquitetura de um *asset* 3D do tipo *full frame*, foi necessário arranjar alguns exemplos. Assim que se recolheu esta base de estudo, foi possível efetuar uma análise de engenharia reversa para tentar identificar as características presentes nos *assets* que lhe conferiam a propriedade 3D.

Como seria de esperar, verificou-se que os dois canais de vídeo estavam armazenados em ficheiros diferentes. Apesar de tudo, esta evidência não era suficiente para que o *Media Composer* conseguisse mapear o *asset* 3D na sua totalidade. Ainda faltava saber a que olho cada canal de vídeo correspondia, por exemplo.

Após uma análise mais detalhada, chegou-se à conclusão que o MXF 3D era definido na íntegra pela existência de dois tipos de metadados no *Material Package* e na partilha de um

UMID32⁷ por parte dos últimos *Source Packages* presentes no ficheiro (figura 28). Foram detetados os seguintes metadados:

- **S3D_CHANNEL**: responsável por indicar o olho ao qual o canal de vídeo se refere;
- **S3D_GROUP_NAME**: corresponde a um identificador único do *asset*. Todos os ficheiros MXF que partilhem este mesmo identificador pertencem ao mesmo conteúdo.

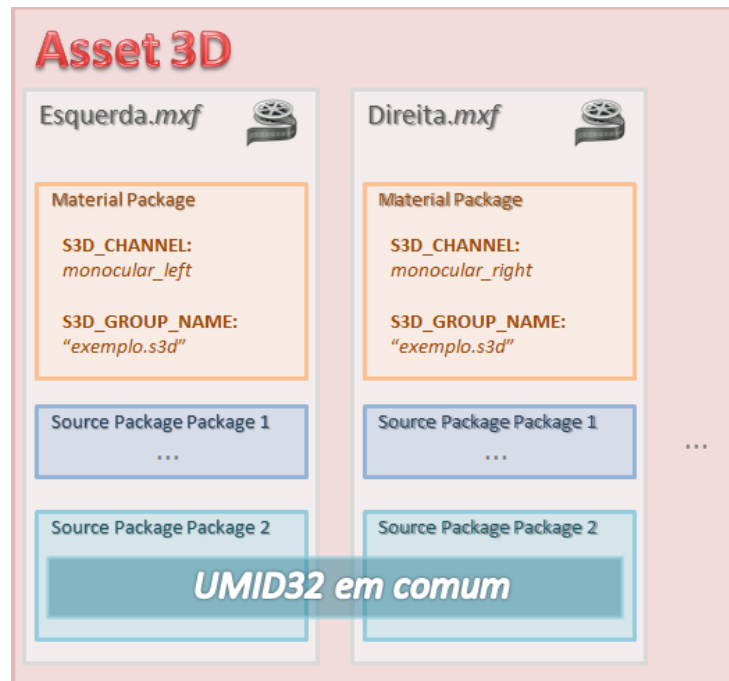


Figura 28: Estrutura do MXF de um *asset* 3D - Avid

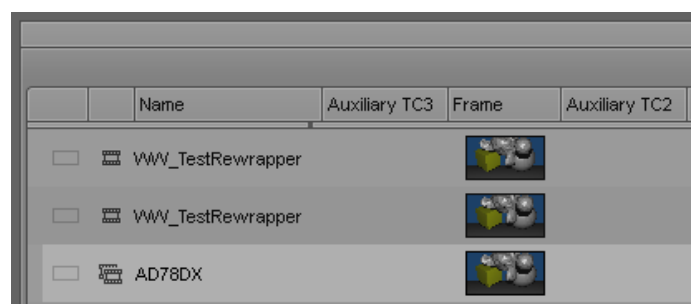


Figura 29: Vista de edição de um conteúdo em 3D – Media Composer

Porém, foi ainda necessário determinar as propriedades do ficheiro AAF. Se este não estiver bem definido, o Media Composer não sabe como interpretar o *asset*. Olhando para um exemplo concreto, consegue-se ter uma noção mais realista dos requisitos de um ficheiro AAF

⁷ Um UMID32 é um identificador de 32 bytes que é usado no MXF para identificar um *Package* ou uma unidade de essência. [9]

Implementação da Solução

3D. Como se pode ver pela figura 29, o *software* reconhece individualmente cada canal de vídeo e acrescenta uma terceira vista que corresponde à junção dos dois canais. Esta abordagem é muito interessante, pois permite ao operador fazer edições num só canal ou no conteúdo todo de uma vez.

Depois de percebido o conceito de aglomeração, é mais simples compreender a estrutura de alto nível de um ficheiro “AAF 3D”. Na prática, o ficheiro contém duas referências: uma para o MXF esquerdo e outra para o MXF direito. Por sua vez, essas duas referências são referenciadas num terceiro objeto, também conhecido por “CompositionMob” [37]. Ele é responsável por definir a vista denominada “AD78DX”, presente na figura 29.

As CompositionMob podem definir um conjunto bastante amplo de vistas. Tipicamente, são usadas para colar vários *clips* de vídeo em sequência. Porém, no caso de um *asset* 3D, ao invés de estarem colocados temporalmente um a seguir ao outro, os dois *clips* de vídeo existem na mesma linha temporal.

Na figura 30 pode-se ver uma representação em XML de uma CompositionMob. Como se pode ver, existe uma linha temporal para o vídeo (*TimelineMobSlot*) que engloba as tais duas referências para os dois *clips* de imagem (*SourceClip*). O atributo *SourceID* do *SourceClip* guarda o UMID32 do *Material Package* do MXF correspondente [38]. Assim, e como cada ficheiro MXF tem o seu *Material Package*, é possível ao *Media Composer* estruturar o *asset* e mostrá-lo corretamente ao utilizador.

O aspeto final do conteúdo audiovisual é o apresentado na figura 31. O *software* de edição permite a definição de variados modos de visualização do vídeo. Neste caso, escolheu-se visualizar o vídeo anáglifo a partir do congénere estereoscópico.

Analisando mais em pormenor o *asset*, verifica-se pela figura 32 que as propriedades correspondentes ao par estereoscópico foram corretamente carregadas para o programa. Verifica-se que o “Group Name” está correto, que existe um canal para o olho direito e outro para o olho esquerdo, confirmando-se também quais são todos os contribuintes do *asset* (atributo “Tracks” e “S3D Contributors”). Neste caso, além dos dois canais de vídeo, está também presente som no conteúdo importado.

```

- <CompositionMob>
  <MobID>urn:x-umid:060a2b340101010101010f00-13-000000-060e2b347f7f2a804f8c50d1b81d1662</MobID>
  <MobName>AD78DX</MobName>
  <MobSlots>
    <TimelineMobSlot>
      <SlotID>1</SlotID>
      <SlotName/>
      <MobSlotSegment>
        <NestedScope>
          <ComponentDataDefinition>DataDef_LegacyPicture</ComponentDataDefinition>
          <ComponentLength>0</ComponentLength>
          <NestedScopeSlots>
            <OperationGroup>
              <ComponentDataDefinition>DataDef_LegacyPicture</ComponentDataDefinition>
              <ComponentLength>0</ComponentLength>
              <Operation>urn:uuid:c45bbc19-7112-4f7c-b277-fd5825504ef7</Operation>
            </OperationGroup>
            <InputSegments>
              <SourceClip>
                <ComponentDataDefinition>DataDef_LegacyPicture</ComponentDataDefinition>
                <ComponentLength>60</ComponentLength>
                <SourceID>urn:x-umid:060a2b340101010101010f00-13-000000-060e2b347f7f2a80a8508c4f405e6216</SourceID>
                <SourceMobSlotID>1</SourceMobSlotID>
                <StartTime>0</StartTime>
              </SourceClip>
              <SourceClip>
                <ComponentDataDefinition>DataDef_LegacyPicture</ComponentDataDefinition>
                <ComponentLength>60</ComponentLength>
                <SourceID>urn:x-umid:060a2b340101010101010f00-13-000000-060e2b347f7f2a80a8508c4ff92c6216</SourceID>
                <SourceMobSlotID>1</SourceMobSlotID>
                <StartTime>0</StartTime>
              </SourceClip>
            </InputSegments>
            <Parameters>
              <OperationGroupRendering>
                <ScopeReference>
                  <NestedScopeSlots>
                    <NestedScope>
                      <MobSlotSegment>
                        <PhysicalTrackNumber>1</PhysicalTrackNumber>
                        <TimelineMobSlotEditRate>25/1</TimelineMobSlotEditRate>
                        <SlotOrigin>0</SlotOrigin>
                      </MobSlotSegment>
                    </NestedScope>
                  </ScopeReference>
                </OperationGroupRendering>
              </Parameters>
            </InputSegments>
          </NestedScope>
        </MobSlotSegment>
        <PhysicalTrackNumber>1</PhysicalTrackNumber>
        <TimelineMobSlotEditRate>25/1</TimelineMobSlotEditRate>
        <SlotOrigin>0</SlotOrigin>
      </TimelineMobSlot>
      <MobLastModified>2012-04-16T17:03:13.00Z</MobLastModified>
      <CreationTime>2012-04-16T17:03:13.00Z</CreationTime>
      <UsageCode>Usage_TopLevel</UsageCode>
      <this:AppCode>15</this:AppCode>
    </this:MobAttributeList>
    <TaggedValue>
      <TaggedValueName>_S3D_GROUP_NAME</TaggedValueName>
      <TaggedValueValue aaf:actualType="String">AD78DX</TaggedValueValue>
    </TaggedValue>
    <TaggedValue>
      <TaggedValueName>_S3D_LEADING_EYE</TaggedValueName>
      <TaggedValueValue aaf:actualType="String">LEFT</TaggedValueValue>
    </TaggedValue>
    <TaggedValue>
      <TaggedValueName>_USER_POS</TaggedValueName>
      <TaggedValueValue aaf:actualType="String">0</TaggedValueValue>
    </TaggedValue>
  </this:MobAttributeList>
</CompositionMob>

```

Figura 30: Representação em XML de uma CompositionMob

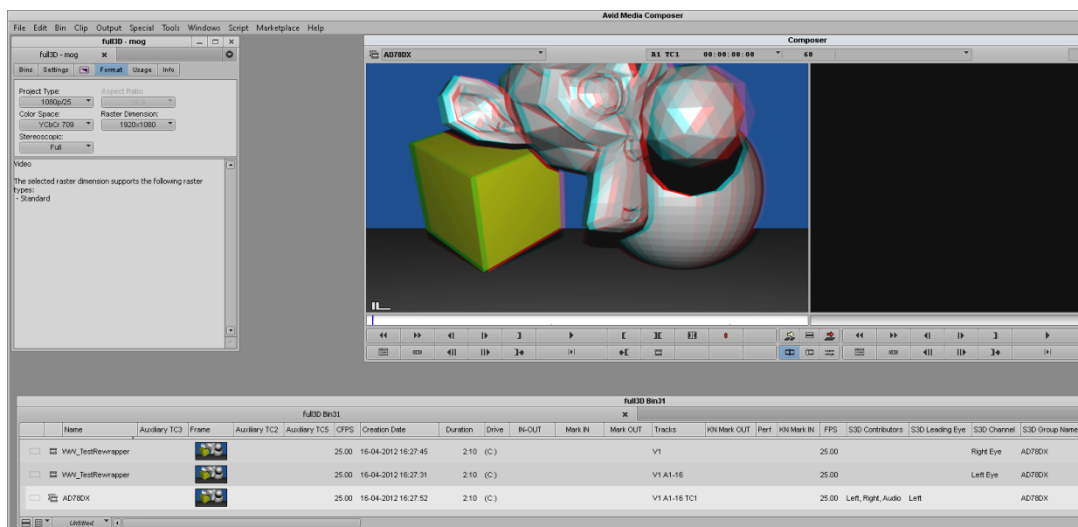


Figura 31: Visão geral de um *asset* 3D – Media Composer

Implementação da Solução

Tracks	KN Mark OUT	Perf	KN Mark IN	FPS	S3D Contributors	S3D Leading Eye	S3D Channel	S3D Group Name
V1				25.00			Right Eye	AD78DX
V1 A1-16				25.00			Left Eye	AD78DX
V1 A1-16 TC1				25.00	Left, Right, Audio	Left		AD78DX

Figura 32: Atributos de um conteúdo em 3D

4.4 Produção de Anáglifos

O processo de criação de anáglifos em RGB é bastante simples. Um programa aloca três *buffers*, em que cada um não é mais do que um *array* de *bytes* cuja dimensão varia consoante a resolução da imagem. Num *buffer* estará a imagem do olho esquerdo, num outro a do olho direito e num terceiro será guardado o anáglifo. Nos casos mais simples de representações baseadas em RGB, as três componentes de cor vêm entrelaçadas, tal como mostra a figura 33. Assim, tipicamente um programa criador de anáglifos só terá de criar três iteradores, um para cada imagem e, ao percorrer as imagens, copiar *byte* a *byte* o conteúdo para a imagem final. Se o *byte* em análise no momento pertencer à gama dos vermelhos, então o programa deverá copiar da imagem direita. Senão, o *byte* será copiado da imagem esquerda. Apesar de, em RGB, o processo ser relativamente simples, tal não acontece se a imagem for representada noutro espaço de cor.

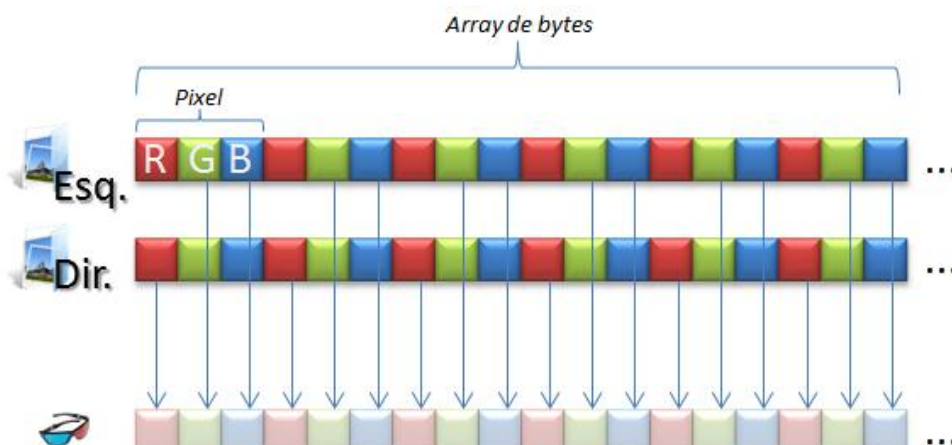


Figura 33: Processo de produção de um anáglifo Vermelho/Ciano em RGB

O vídeo capturado pelas câmaras costuma vir codificado em YUV. Isto deve-se em grande parte à herança deixada pela transição da “televisão a preto e branco” para a televisão a cores. No passado, os fabricantes criavam televisores com a capacidade para receber somente a informação na escala dos cinzentos. Com a passagem para a televisão a cores, a forma mais simples de representar a imagem seria adicionando dois canais extra ao previamente existente (figura 34). Esta opção trouxe muitas vantagens operacionais e acabou por se tornar num

paradigma que ainda hoje em dia se mantém. Por essa razão foi necessário adaptar os algoritmos de criação de anáglifos para operarem no espaço de cor correto.

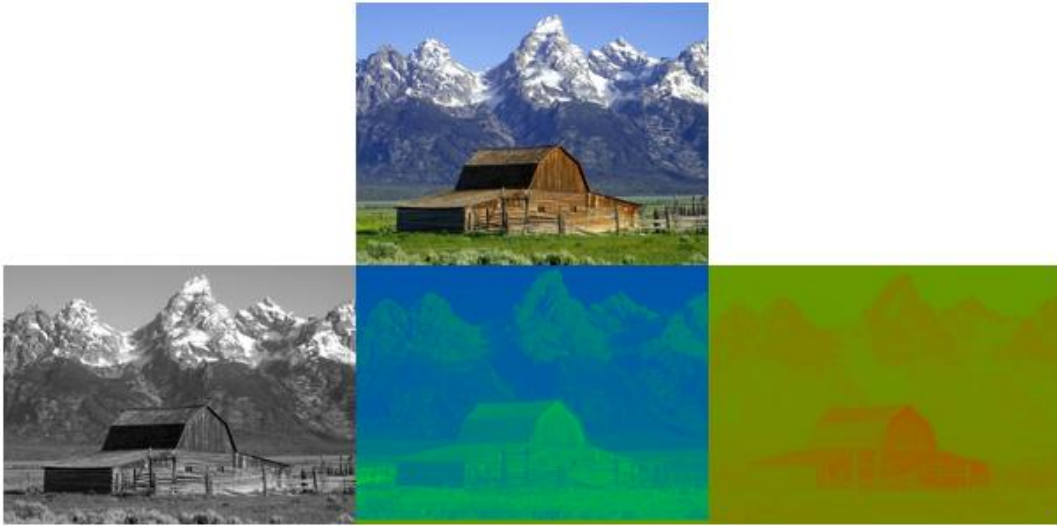


Figura 34: Imagem a cores separada nas suas diferentes componentes YUV

4.4.1 Especificação YUV para Vídeo HD

Existem várias variantes do YUV. Para o caso do vídeo de alta definição, as fórmulas definidas pela ATSC para converter YUV em RGB e vice-versa (normas BT.601 e BT.709) são as seguintes:

$$\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.2126 & 0.7152 & 0.0722 \\ -0.09991 & -0.33609 & 0.436 \\ 0.615 & -0.55861 & -0.05639 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.28033 \\ 1 & -0.21482 & -0.38059 \\ 1 & 2.12798 & 0 \end{bmatrix} \begin{bmatrix} Y' \\ U \\ V \end{bmatrix}$$

Figura 35: Fórmulas de conversão de RGB/YUV e YUV/RGB

4.4.2 Criação de Anáglifos em YUV

Uma das primeiras abordagens para resolver o problema de criar anáglifos em YUV consistiu na conversão da imagem para RGB, processamento do anáglifo em RGB e por fim na conversão inversa do anáglifo para YUV (processo exemplificado na figura 36).

Implementação da Solução

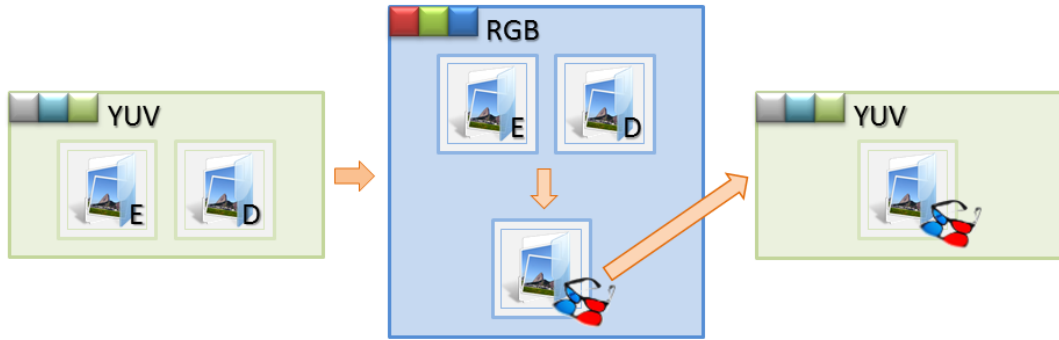


Figura 36: Esquema para produção de anáglifos em YUV

A computação torna-se bastante morosa porque envolve múltiplos cálculos matriciais. A fórmula presente na figura 37 espelha esta complexidade. Em primeiro lugar é necessário multiplicar os vetores em YUV de ambas as imagens por uma constante matricial para obter a imagem em RGB, depois isolar em cada imagem os canais pretendidos, somar de seguida as duas seleções e por fim fazer uma multiplicação que corresponde à conversão inversa de RGB para YUV.

$$\begin{bmatrix} Y'_a \\ U'_a \\ V'_a \end{bmatrix} = \begin{bmatrix} 0.2126 & 0.7152 & 0.0722 \\ -0.09991 & -0.33609 & 0.436 \\ 0.615 & -0.55861 & -0.05639 \end{bmatrix} \left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1.28033 \\ 1 & -0.21482 & -0.38059 \\ 1 & 2.12798 & 0 \end{bmatrix} \begin{bmatrix} Y'_d \\ U'_d \\ V'_d \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1.28033 \\ 1 & -0.21482 & -0.38059 \\ 1 & 2.12798 & 0 \end{bmatrix} \begin{bmatrix} Y'_e \\ U'_e \\ V'_e \end{bmatrix} \right)$$

Figura 37: Fórmula para criação de anáglifos Vermelho/Ciano – BT.709 e BT.601

4.5 Anáglifos – Aceleração do Processamento

Verificou-se que a implementação de um módulo baseado na solução matricial resultava, tal como esperado, num algoritmo muito lento e com poucas possibilidades de utilização numa solução real, por não conseguir dar vazão em tempo real ao sinal que está a ser adquirido. Por essa razão foi necessário efetuar algumas otimizações:

1. Melhorar fórmulas para o cálculo do anáglifo;
2. Paralelizar a computação.

4.5.1 Simplificação das Fórmulas

Esta etapa consistiu na simplificação da equação matricial apresentada na figura 37. Na realidade, como o procedimento envolvia uma conversão inicial e no fim uma conversão inversa, muitas das parcelas acabavam por se anular, introduzindo computações supérfluas. Assim, chegou-se a um conjunto de fórmulas computacionalmente mais simples e que são apresentadas na figura 38.

$$\begin{aligned}
Y_a &= 0.787*Y_e + 0.213*Y_d + 0.272*(V_d - V_e) \\
U_a &= 0.1*(Y_e - Y_d) + U_e + 0.128*(V_e - V_d) \\
V_a &= 0.615*(Y_d - Y_e) + 0.787*V_d + 0.213*V_e
\end{aligned}$$

Figura 38: Fórmula otimizada para produção de anáglifos Vermelho/Ciano – BT.709 e BT.601

A simplicidade das equações é evidente, passando a ser possível calcular cada componente de cor com apenas 6 operações aritméticas. Contudo, ainda existem possibilidades de otimização. As constantes são apresentadas em forma decimal, o que computacionalmente significa a alocação de variáveis do tipo vírgula flutuante, que ocupam 32 bits. Desta forma, o cálculo de um só componente de cor requeria o processamento de quatro *bytes* em vez de um só. Decidiu-se portanto escrever as equações anteriores num outro formato, de forma a ser possível representá-las, de forma aproximada, através de constantes inteiras (figura 39). Uma vantagem marginal, mas que acabou por ser aproveitada, foi a possibilidade de aproveitar os registos MMX para computar grupos de 8 pixeis de cada vez (em vez de 4), sendo assim aumentada a rapidez da produção dos anáglifos. Adiante será apresentada a abordagem de criação de anáglifos com MMX.

$$\begin{aligned}
Y_a' &= \frac{7Y_e + 2Y_d}{9} + \frac{V_d - V_e}{4} \\
U_a &= \frac{Y_e - Y_d}{10} + U_e + \frac{V_e - V_d}{8} \\
V_a &= \frac{3}{5}(Y_d - Y_e) + \frac{7V_d + 2V_e}{9}
\end{aligned}$$

Figura 39: Aproximação inteira da fórmula da figura 38

4.5.2 Paralelização da Computação

Paralelizar o processamento dos anáglifos é uma tarefa relativamente simples pois o cálculo de um pixel é independente de outro qualquer. Desta forma, é possível computar vários pixeis em simultâneo sem haver concorrência no acesso às mesmas posições de memória (figura 40). Na figura seguinte está esquematizada uma atribuição de pequenos blocos de pixeis a *threads* diferentes. Cada *thread* pode ser executada na GPU ou na CPU. Numa execução perfeita, totalmente paralelizada, o tempo de execução será muito menor do que numa abordagem sem paralelismo.

Implementação da Solução

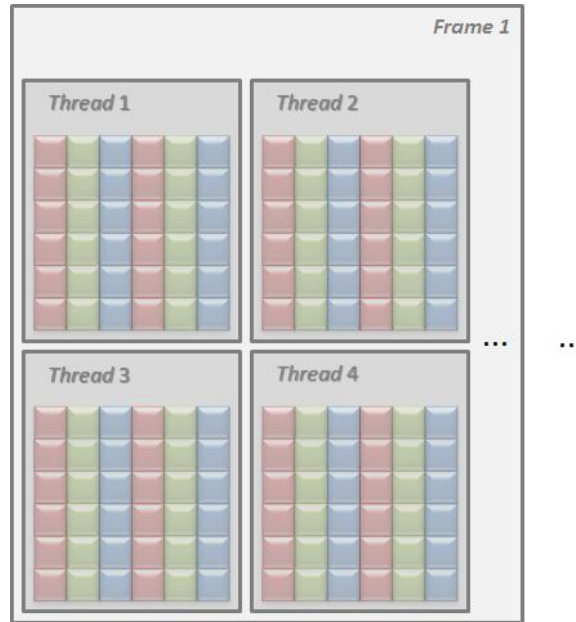


Figura 40: Esquematização do processo de paralelização do processamento

No contexto da tese foram exploradas algumas formas de paralelizar a computação. A primeira, e mais simples de todas, consistiu na alocação de várias *threads* de processador para processar vários blocos de memória independentes. Neste contexto, o **multithreading** é útil para aproveitar o “tempo morto” criado no acesso à memória, sendo aproveitado para que uma outra *thread* efetue cálculos aritméticos em *cores* livres da CPU.

Foi ainda experimentado o uso da **GPU (arquitetura CUDA)**. Neste cenário, foi desenvolvido um módulo responsável por transferir para a memória interna da GPU dois blocos de *pixels* e posteriormente efectuar uma computação do anáglifo com múltiplas *threads* [39]. Por fim, o anáglifo foi copiado para a memória acessível à CPU. Foram experimentados vários ambientes de execução do módulo na GPU, variando o número de *threads* por bloco assim como o próprio número de blocos.

Adicionalmente foi analisada uma outra opção para acelerar o cálculo. A Intel disponibiliza um conjunto de registos de processador que permitem a aplicação simultânea de uma operação sobre várias variáveis. Estes registos usam a tecnologia **MMX**. Os registos MMX têm a dimensão de 64 ou 128 bits (2 ou 4 vezes mais que os tradicionais), possibilitando assim o empacotamento num só registo de valores referentes a vários pixeis. No caso das aproximações inteiras das equações de criação de anáglifos, foi possível agrupar de uma só vez variáveis referentes a 8 *pixels* (ver figura 41). Como a tecnologia MMX distingue as várias variáveis que estão agrupadas num registo e efetua o mesmo cálculo em cada variável, teoricamente é possível aumentar a rapidez de computação em 8 vezes. Porém, por vezes esse ganho pode não ser totalmente atingido, devido a muitos fatores. Por exemplo, o processo de empacotamento das variáveis nos registos estendidos irá sempre adicionar processamento adicional que de outra forma não existiria. Os resultados obtidos serão apresentados e discutidos no capítulo seguinte.

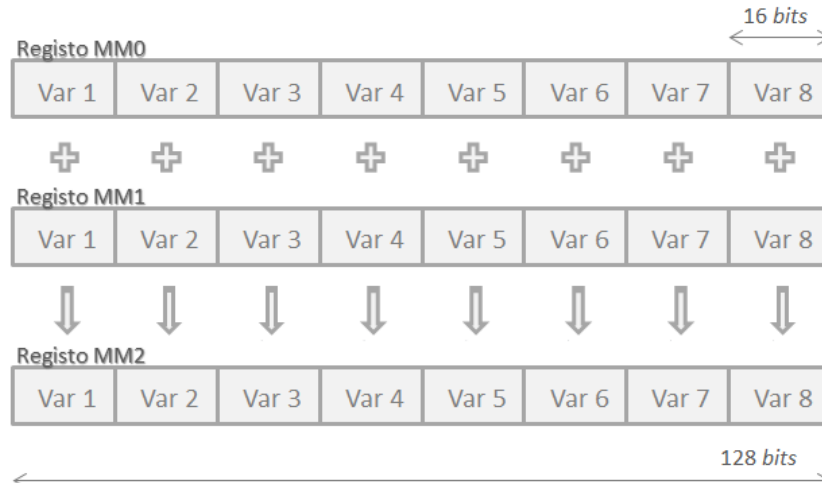


Figura 41: Exemplo de uma operação de soma com tecnologia MMX

Uma vez que a tecnologia MMX não disponibiliza instruções nativas para efetuar a divisão de inteiros, houve a necessidade de reescrever as equações da figura 39, de forma a que todas as divisões fossem substituídas pela operação de *shift* para a direita. Uma vez que um *shift* para a direita equivale à divisão inteira por dois, foi necessário transformar as equações da figura 39 de forma a terem o denominador múltiplo de dois. Houve uma atenção especial em confirmar que os erros das conversões das equações não afetam a qualidade final dos anáglifos. As equações finais obtidas para o cálculo recorrendo à tecnologia MMX são apresentadas na imagem 42.

$$\begin{aligned}
 Y'_a &= ((25Y'_e + 7Y'_d) >> 5) + ((V_d - V_e) >> 2) \\
 U_a &= (3(Y'_e - Y'_d) >> 5) + U_e + ((V_e - V_d) >> 3) \\
 V_a &= (5(Y'_d - Y'_e) >> 3) + ((25V_d + 7V_e) >> 5)
 \end{aligned}$$

Figura 42: Aproximação inteira com *shifts* das equações de criação de anáglifos

4.6 Vídeo HD 3D Anáglifo Codificado em UYVY

Uma das codificações de cor mais usadas pelas empresas associadas ao audiovisual é o formato UYVY. Como foi referido anteriormente, é uma representação comprimida de YUV, onde cada par de *pixels* adjacentes partilha os mesmos *bytes* de crominância (U e V). Apesar de o UYVY não variar significativamente do YUV original, a variação é suficiente para afetar a forma como o algoritmo de criação de anáglifos funciona.

Como se ilustra na figura 43, dado que existem dois valores de luminância, é necessário decidir qual deles vai ser utilizado nas fórmulas de produção do anáglifo. No contexto da tese, optou-se por calcular um valor médio, utilizando-o posteriormente em todos os cálculos envolvendo as crominâncias.

Implementação da Solução

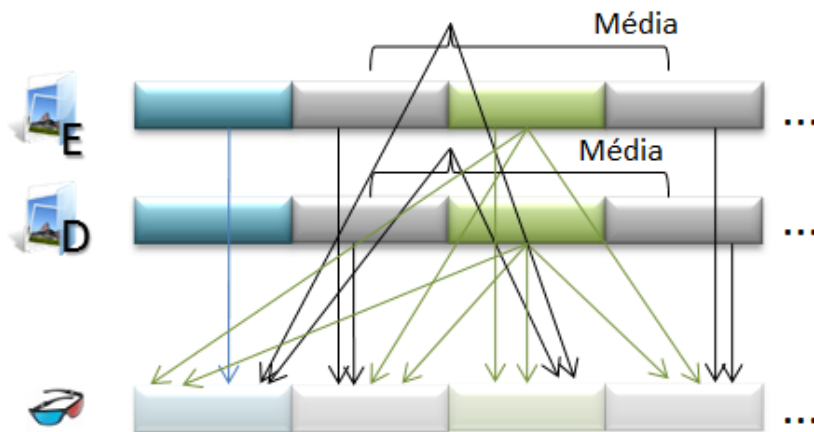


Figura 43: Comparticipação dos componentes UYVY na produção de anáglifos Vermelho/Ciano

Foi inicialmente desenvolvido um protótipo de criação de anáglifos que recebe dois ficheiros com extensão *.yuv*, processa-os e guarda o resultado final num terceiro ficheiro *.yuv*. Para verificar se a aplicação funcionava corretamente utilizou-se um programa chamado “Yuv Viewer”, que permite a visualização de vídeos em formato *.yuv*. Como se pode ver pela figura 44, especialmente com auxílio de óculos vermelho/ciano, é possível extrair informação sobre a profundidade relativa dos objetos.

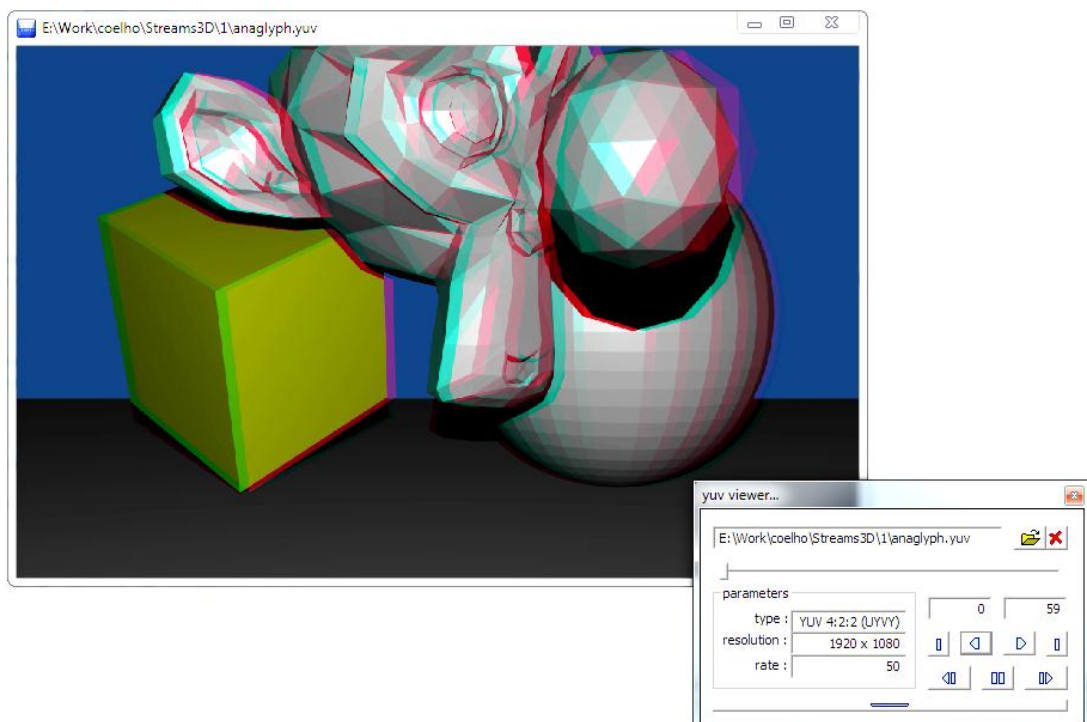


Figura 44: Exemplo de vídeo UYVY anáglifo

4.7 Validação do *Software* Desenvolvido

A placa DeckLink também permite que pacotes de dados sejam enviados para o exterior através de canais SDI de saída. Esta funcionalidade foi utilizada no projeto para validar o *software* que foi desenvolvido. Uma vez que as câmaras de filmar profissionais são muito caras e não estavam acessíveis durante o tempo em que se desenvolveu o projeto, escolheu-se simular o processo de filmagem. Para isso, desenvolveu-se um programa que encaminha vídeo importado de dois ficheiros para os portos SDI de saída. Por sua vez, uma segunda estação, equipada com um mxfsPEEDRAIL S1000, efetuava a aquisição dos vídeos e verificava a sua integridade (ver esquema da figura 45).

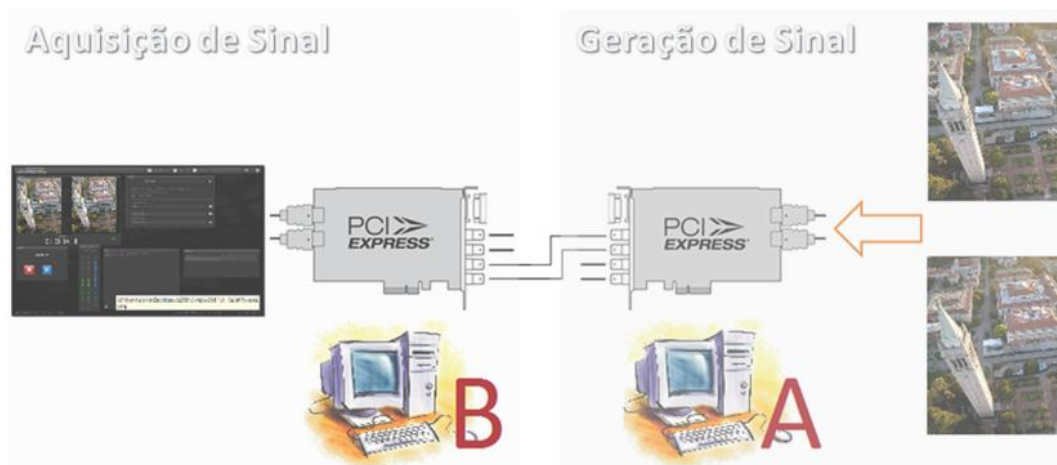


Figura 45: Processo de validação do *software*

4.7.1 Preparação dos Vídeos de Teste

Um outro entrave à validação do *software* é a pouca disponibilidade de vídeos 3D do tipo *full frame*. Além do mais, também é muito importante que a informação do vídeo esteja armazenada como se tivesse acabado de ser codificada por uma câmara, ou seja, em UYVY.

A solução encontrada para produzir grandes quantidades de filme de teste passou pelo recurso a *software* para modelação de cenas 3D. O *software* escolhido foi o Blender. A sua principal vantagem é ser de acesso livre e permitir a exportação de animações para vídeo.

Criaram-se ambientes 3D com animações simples e por fim renderizaram-se essas animações para filme HD. O vídeo final foi arquivado em AVI (ver figura 46). O filme foi renderizado duas vezes a partir de duas perspetivas ligeiramente diferentes.

Implementação da Solução

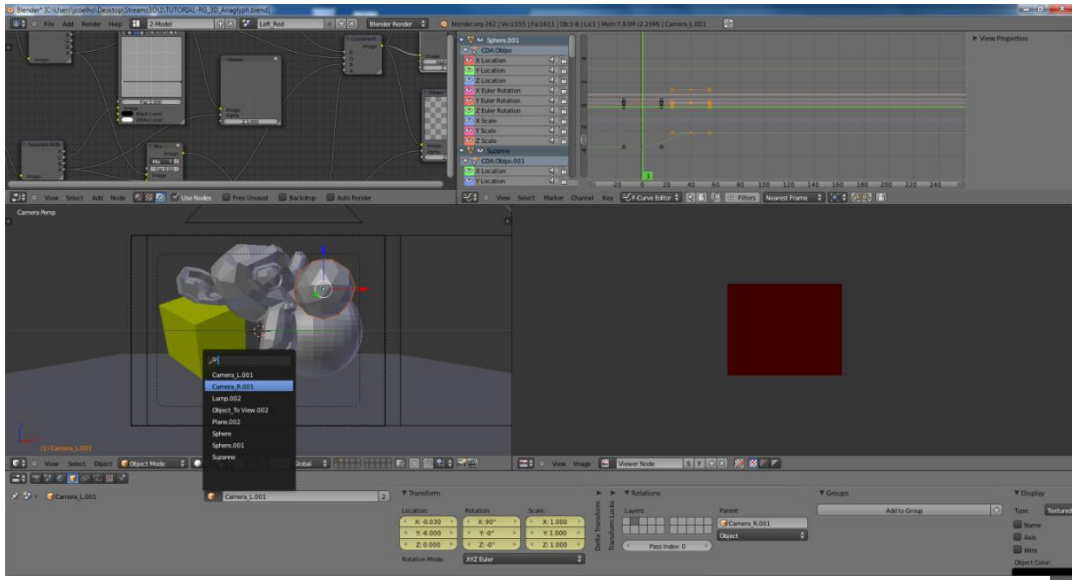


Figura 46: Utilização do Blender para renderização de filme de animação em HD

Uma vez que o espaço de cor que o Blender usa é o RGB, foi necessário extrair a essência do AVI e convertê-la para UYVY. Para o conseguir recorreu-se a um outro *software* livre, o FFMPEG. Este programa possui ferramentas para ler, reproduzir e transcodificar vídeo. Para converter cada AVI para UYVY bastou executar o seguinte comando:

```
ffmpeg -i "input_file.avi" -pix_fmt uyvy422 "output_file.yuv"
```

Por fim, estando os dados armazenados em "output_file.yuv", passou a ser possível efetuar a fase de validação.

4.7.2 Difusão do Vídeo 3D

Como os vídeos já estavam prontos para serem usados em testes, avançou-se para a tarefa de desenvolver um pequeno programa que, usando a API da DeckLink, os enviasse para a estação de captura equipada com o mxfSPEEDRAIL S1000.

À semelhança do que sucede com a aquisição de vídeo na DeckLink, a difusão requer o registo de uma *callback* (ver figura 48). Isto porque o envio dos pacotes tem que ser agendado para um determinado momento no futuro [36]. Assim que esse momento é atingido, a *frame* é enviada e um evento desencadeado (dando origem à execução da *callback*, como evidenciado na figura 47).

Geralmente, sempre que uma *frame* é enviada, é também agendada a difusão de uma outra (enquanto houver *frames* para enviar). Este método é designado de "*post role*". Por outro lado, antes de se dar início à transmissão propriamente dita, é prática comum agendar previamente o envio de uma grande quantidade de *frames*. Este método é conhecido por "*pre role*".

```

class MyOutputHandler : public IDeckLinkVideoOutputCallback, public IDeckLinkAudioOutputCallback
{
    // ...

    // Método executado sempre que uma frame é enviada
    HRESULT ScheduledFrameCompleted (
        IDeckLinkVideoFrame* completedFrame,
        BMDOutputFrameCompletionResult result)
    {
        // ...

        DeckLink3dVideoFrame* tmp = GET_NEXT_FRAME();

        // Agendar saída da frame
        this->YUV_Reader->ScheduleNextFrame ( false, tmp );

        // Libertar memória
        completedFrame->Release();
        return S_OK;
    }
}

```

Figura 47: Inicialização do modo de transmissão de sinal

```

class DeckLink3dVideoFrame : public IDeckLinkVideoFrame, public IDeckLinkVideoFrame3DExtensions
{
public:

    // ...

    // Override...

    void SetRightEyeFrame(DeckLink3dVideoFrame* right) { mRightEyeFrame = right; }

    HRESULT STDMETHODCALLTYPE GetFrameForRightEye(IDeckLinkVideoFrame** rightEyeFrame)
    {
        *rightEyeFrame = mRightEyeFrame;

        if (mRightEyeFrame)
        {
            mRightEyeFrame->AddRef();
            return S_OK;
        }
        else
        {
            return S_FALSE;
        }
    }

private:

    // ...

    DeckLink3dVideoFrame* mRightEyeFrame;
};

```

Figura 48: Exemplo de código para definição do *handler* de saída de sinal

No caso do vídeo 3D, as *frames* não podem ser representadas pela tradicional “IDeckLinkVideoFrame”. Antes, deve ser criada uma classe que a estenda e que implemente as funcionalidades previstas na interface “IDeckLinkVideoFrame3DExtensions” [36]. Como se vê na figura 48, o método “GetFrameForRightEye” deve estar bem implementado para poder ser executado internamente no momento da transmissão da *frame* 3D.

Implementação da Solução

Por fim, e uma vez mais à semelhança do que acontece com a aquisição, a *callback* deve ser registada através da função “SetScheduledFrameCompletionCallback” [36]. O modo de difusão do sinal 3D também deve ser ativado através do método “EnableVideoOutput”, não esquecendo de passar o formato de vídeo correto assim como a *flag* referente à extensão 3D (ver figura 49).

```
// Registrar objeto para receber notificação sempre que uma frame for enviada
MyOutputHandler * oh = new MyOutputHandler();
m_deckLinkOutput->SetScheduledFrameCompletionCallback(oh);

// Configurar a placa DeckLink para enviar em 3D
if (m_deckLinkOutput->EnableVideoOutput(displayMode, bmdVideoOutputDualStream3D) != S_OK)
{
    MessageBox(L"Erro a configurar DeckLink para enviar em 3D.");
    return ~S_OK;
}

// ...
```

Figura 49: Definição de classe para suportar *frames* 3D

Na figura 50 mostra-se um exemplo do processo de validação do *software*. Foram inicializadas duas instâncias do Windows Remote Desktop (cada uma numa máquina diferente). Na máquina “fo4” estava em execução um programa para envio do vídeo 3D enquanto que na outra (“s1100”) estava a ser executada uma instância do mxfsPEEDRAIL S1000 para aquisição do sinal difundido.

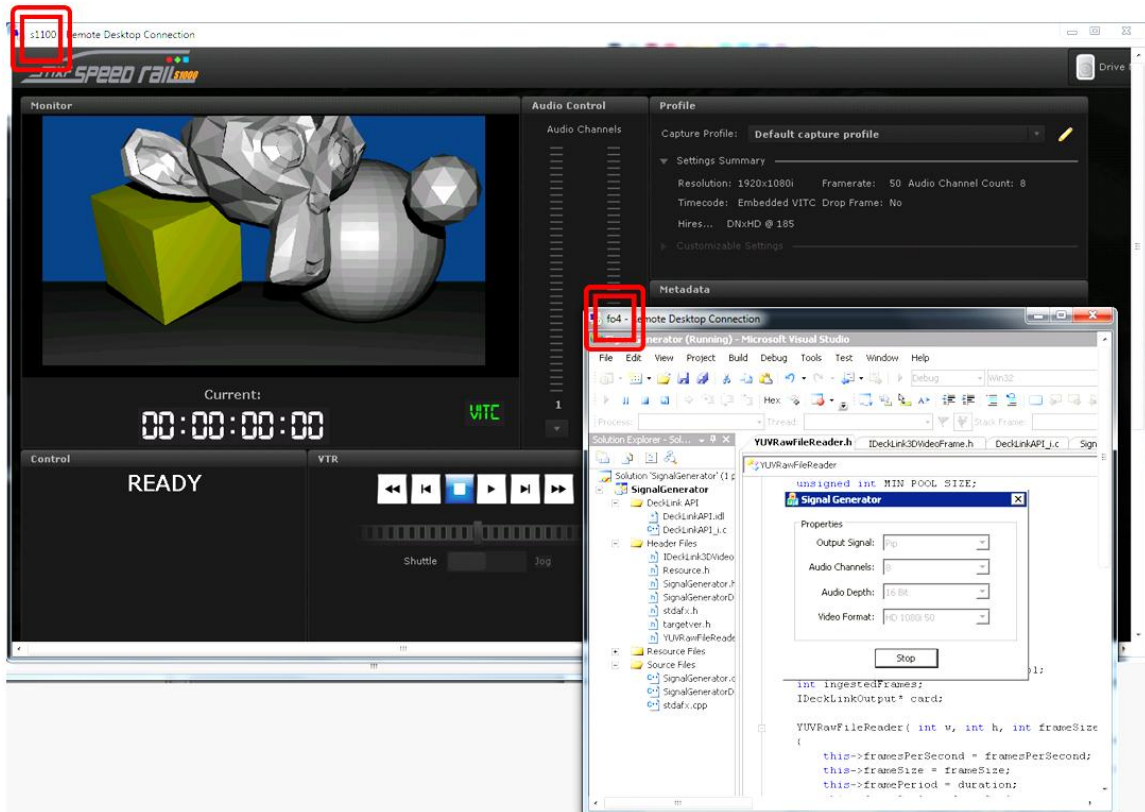


Figura 50: Exemplo de processo de validação do *software*

Capítulo 5

Resultados e Discussão

5.1 Visão Geral

Um grande objetivo da tese consistiu em medir o ganho na velocidade de processamento possibilitado por técnicas de computação paralela. Para atingir este objetivo, foram montados alguns cenários de teste onde foram experimentadas algumas abordagens, nomeadamente através do uso da GPU, de MMX e também de *multithreading* ao nível da CPU.

Para se conseguir ter uma noção mais realista dos ganhos oferecidos pelo paralelismo, os testes procuraram variar alguns dos parâmetros mais importantes dos dispositivos usados. Tanto a CPU como a GPU usam a abstração “*thread*” para representar um fluxo de execução do código (podendo ser bifurcado em cenários de execução em paralelo de várias *threads*). No caso da arquitetura CUDA (do tipo SIMD) [31], as *threads* são mais limitadas do que numa CPU convencional, pois não é possível executar vários *kernels* em *threads* diferentes. Esta característica acabou por não influenciar os testes, pois o cálculo de anáglifos é um processo do tipo SIMD. Assim, na prática só foi definido um *kernel* para todas as *threads* (CPU e GPU). Para tornar os testes mais enquadrados no tema da tese, optou-se por usar como *kernel* de teste um método que recebesse duas *frames* YUV e gerasse um anáglifo.

Por outro lado, houve também o cuidado de que todos os testes fossem executados na mesma máquina, para que diferenças nas capacidades de processamento de computadores diferentes não adulterassem os resultados finais. O processador utilizado foi um Intel Core 2 Quad Q6600, a 2.40GHz. A unidade gráfica escolhida foi uma NVIDIA GeForce 8500 GT [40].

5.2 Desempenhos da GPU e da CPU

Uma primeira abordagem consistiu na execução do *kernel* na CPU e na GPU. Para argumento escolheu-se um clip de vídeo 1080p com 1 segundo. Para reduzir a variância, o teste foi repetido 25 vezes e os resultados finais correspondem à média dos valores obtidos nas várias execuções. A escolha da resolução 1080p deveu-se ao facto de ser a maior possível em vídeo 3D para televisão.

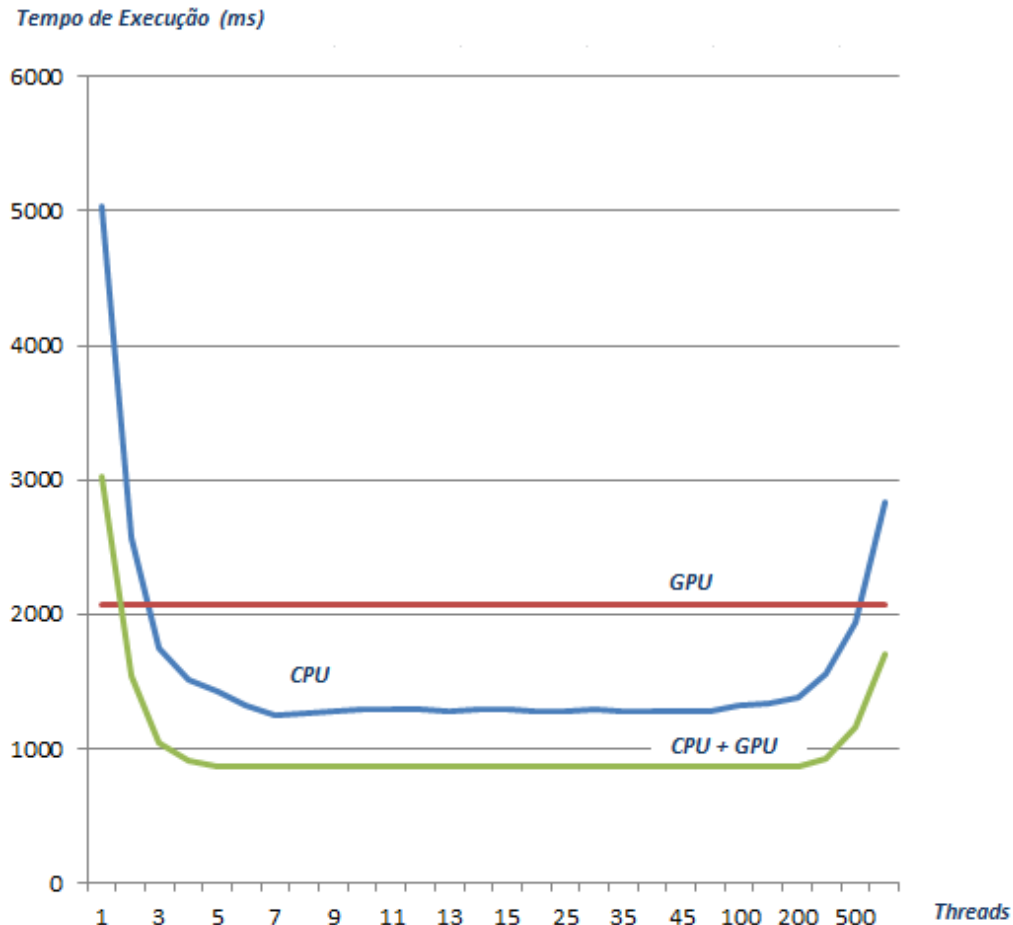


Figura 51: Desempenho da CPU, GPU e CPU+GPU

Posteriormente, compararam-se os resultados. Além de estabelecer qual dos métodos era mais rápido, foi também interessante avaliar qual o número de *threads* que minimizava o tempo de execução. Por outro lado, ao analisar os dados provenientes dos testes com a GPU, duas perguntas surgiram de imediato:

- Porque é que o tempo de execução da GPU é maior do que o da CPU?
- Porque é que o tempo de execução não varia com a variação do número de *threads*?

Como se pode verificar pela figura 51, estranhamente, variar o número de *threads* na GPU não pareceu influenciar o desempenho final do processamento. Tal pode dever-se ao facto de o

dispositivo não ser topo de gama e ver, por isso, a sua capacidade de paralelização limitada. Esta possível explicação parece ser confirmada quando se analisa o tempo de execução, que se situou perto dos 2 segundos. Era de esperar resultados francamente melhores. Em termos de poder de paralelização, a GPU testada está longe de ser a melhor. A GeForce 8500 GT possui somente 16 *cores* e possui um *bitrate* de 12.8 GB/s para comunicação com a memória [40]. Já se analisarmos os valores de uma GPU otimizada para efetuar cálculos paralelamente, como por exemplo a Quadro 6000 [41], verificamos que esta última já possui 448 *cores* assim como um *bitrate* de comunicação com a memória da ordem de 144 GB/s. A velocidade de comunicação com a memória é crucial, visto que para se computar anáglifos na GPU é necessário copiar os *buffers* presentes na memória RAM da CPU para os *buffers* internos da GPU (e depois o processo inverso). Este processo introduz atrasos indesejados na computação e por isso é possível que, para certas GPUs, os ganhos verificados no processamento interno da GPU possam não ser suficientes para contrabalançar as perdas inerentes à cópia dos *buffers*.

Já ao nível da CPU, foram verificados resultados ainda piores quando o *kernel* era executado com poucas *threads* (até 3). Porém, aumentando o seu número foi possível obter resultados relativamente bons (1.3 segundos). Com base no gráfico da figura 51, estima-se que o número ótimo de *threads* de CPU ronde as 7 unidades. Todavia, não se deve esquecer que este número não é fixo e possivelmente varia de *kernel* para *kernel*. Apesar de o processador só possuir 4 *cores*, o facto de o número ótimo de *threads* ser 7 pode levantar, à primeira vista, algumas dúvidas. Porém, não se deve desconsiderar outros fatores que muitas vezes influenciam o desempenho da CPU, principalmente em cenários de computação paralelizada. Um deles, e talvez o mais importante, é o facto de todos os *cores* estarem associados a uma memória comum. Além do mais, uma instrução de acesso a memória é muito mais lenta do que uma instrução aritmética do processador. Isto significa que é certo que o processador vai ficar muitas vezes a aguardar que uma instrução de leitura/escrita na memória seja efetuada. Nesta altura, uma ou mais *threads* podem aproveitar este “tempo morto” para efetuar processamento adicional, enviado de seguida os resultados em grupo para a memória, assim que o acesso lhes seja permitido[47]. Assim sendo, não é necessariamente verdade que o número ótimo de *threads* seja igual ao número de *cores* do sistema. Tudo depende da relação entre a latência de processamento na CPU, do tempo de leitura/escrita em memória, se o sistema possui mecanismos para agrupar vários acessos à memória partilhada[47] e até de outros mecanismos, como as caches[48]. Ou seja, se um determinado *kernel* produtor de anáglifos for 10 vezes mais rápido a efetuar o processamento do que a armazenar os resultados em memória, então em teoria, e havendo um sistema ótimo de controlo, agrupamento e sincronização de acessos a memória, o número ótimo de *threads* de processador deveria rondar as 10.

Uma preocupação que também se teve foi verificar como reagiam os dispositivos à medida que se aumentava o tamanho do vídeo. Era necessário garantir que os tempos de execução aumentavam linearmente. Foi escolhido um vídeo 1080p com a duração de 200 milissegundos (na figura 52 corresponde ao vídeo 1x). O teste foi repetido 25 vezes e os valores finais

correspondem à média dos resultados obtidos para cada execução do teste. Tal como seria de esperar, a linearidade foi verificada (figura 52), tanto para a GPU como para a CPU. Porém, os declives das retas não são os mesmos e tal pode dever-se uma vez mais às diferenças nas características das memórias internas da CPU e da GPU e na necessidade de migrar os dados entre dispositivos.

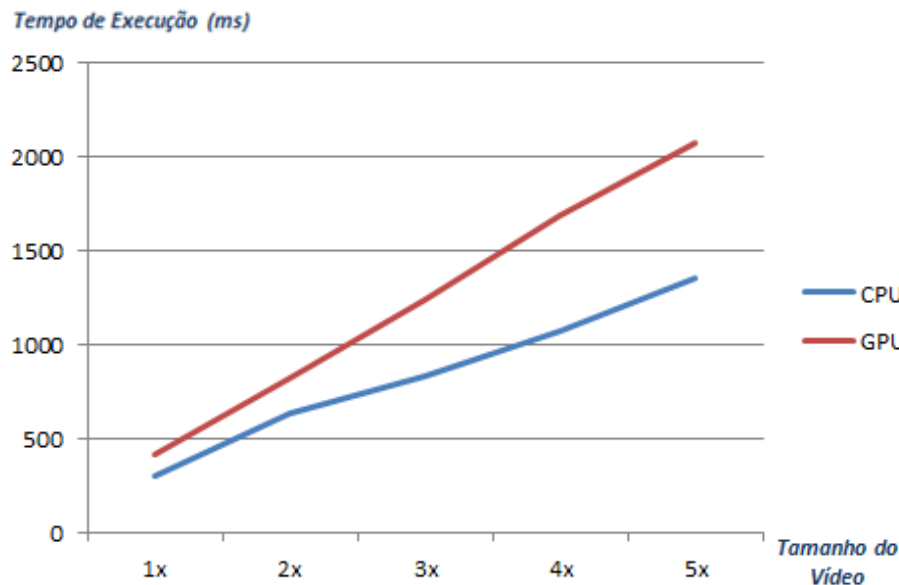


Figura 52: Efeitos do aumento do tamanho do vídeo de teste

5.3 Desempenho do MMX

Os resultados obtidos pela utilização desta tecnologia foram muito animadores. Porém, um obstáculo interessante que se encontrou foi a impossibilidade de efetuar processamento *multithreaded* com os registos MMX. Teoricamente, esta evidência faz sentido, uma vez que o processamento desta natureza é da responsabilidade do coprocessador e não da CPU principal (que é o dispositivo que está dividido em *cores*).

Para o desenvolvimento dos programas de teste optou-se por utilizar uma ferramenta disponibilizada pelo C++, o SSE Intrinsics [42], que permite trabalhar de forma intuitiva com os registos MMX. Por outro lado, ao contrário dos outros testes, desta vez modificou-se o *kernel* para a versão mais otimizada existente (figura 42). Desta forma foi possível analisar também os ganhos de eficiência que as novas fórmulas de produção de anáglifos, anteriormente apresentadas na figura 42, permitiram. Os resultados obtidos serão apresentados na secção 5.4.

5.4 Processamento Combinado

Apesar de ser ter analisado o desempenho individual de cada dispositivo, na realidade pode-se tirar um proveito maior do computador distribuindo a computação de forma combinada pelos vários dispositivos (também visível na figura 51). Cada componente ficará incumbido de processar um determinado grupo de pixels, havendo no final um agrupamento dos resultados. A concorrência por recursos continua a existir, principalmente no acesso à memória principal. Contudo, o tempo que se ganha no processamento torna a abordagem viável.

Um problema desta técnica é que por vezes um processo pode ficar congelado à espera dos resultados de um determinado dispositivo. Por exemplo, quando se considerou uma solução combinada entre a CPU e a GPU, teve-se muito cuidado para que o motor de anáglifos não ficasse à espera da GPU, uma vez que é o dispositivo com menor poder computacional do grupo. A forma de ultrapassar esta diferença de rapidez consistiu no balanceamento da contribuição relativa de cada componente no processamento total.

5.4.1 GPU e *Multithreading*

A definição da contribuição relativa de cada dispositivo foi o passo seguinte a tomar. Caso não se utilizasse o valor correto, estar-se-ia sem dúvida a perder poder computacional precioso. Esta diferença de eficiência é evidente no gráfico da figura 53. À medida que se aumenta a percentagem de CPU, o tempo de execução diminui até ser atingido o valor ótimo (quando a contribuição da CPU ronda os 60%). É de notar que este teste foi efetuado num motor de anáglifos com 7 *threads*. O vídeo usado foi do tipo 1080p com 1 segundo de duração e repetido 25 vezes. Os resultados obtidos podem ser validados com a literatura existente. Efetivamente, houve outros grupos de trabalho que obtiveram resultados semelhantes. Por exemplo, uma investigação sobre o cálculo de multiplicação de matrizes utilizando a CPU e a GPU demonstrou que o ganho da computação híbrida rondaria os 40.1% [46]. Naturalmente, os valores divergem ligeiramente dos resultados obtidos, muito provavelmente devido a diferenças na natureza das funções de *kernel* assim como nas características dos dispositivos usados.

Foi também importante verificar se a tendência se mantinha ao variar o número de *threads* de CPU. Na figura 51 é possível ver o processamento balanceado com um número variável de *threads*, confirmando-se assim o balanceamento ótimo que fora determinado no teste inicial.

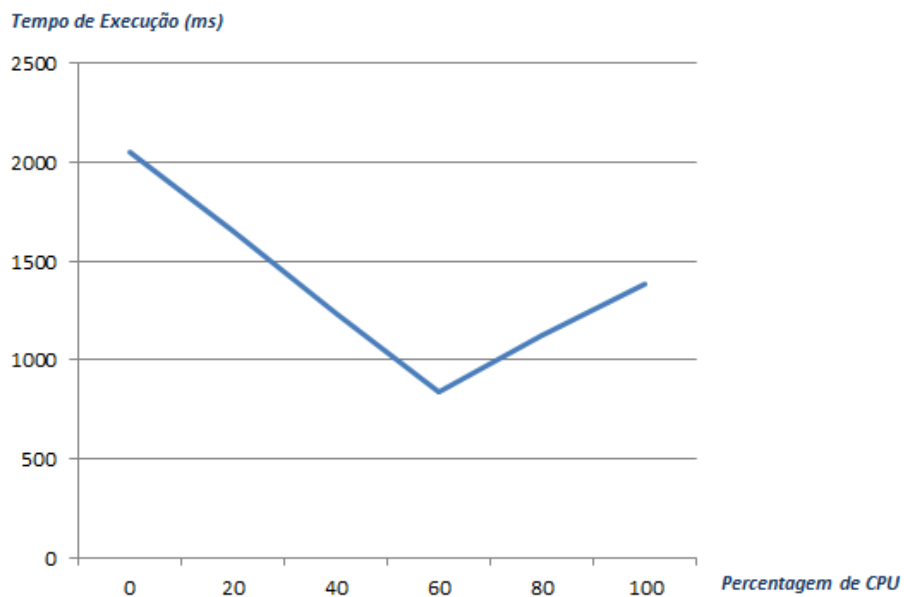


Figura 53: GPU e *Multithreading* – Efeitos da variação da contribuição da CPU

5.4.2 MMX e *Multithreading*

Neste cenário de investigação fez-se um estudo para determinar o número ótimo de *threads* concorrentes ao *kernel* em MMX. Ao contrário das outras implementações anteriores que permitiam a execução de 7 *threads* de CPU em paralelo com o processamento na GPU, neste caso a eficiência máxima foi conseguida somente com 2 *threads* de CPU.

Por outro lado, e à semelhança do que foi exposto na secção anterior, houve uma tentativa para determinar o balanceamento ótimo da contribuição computacional de cada componente. Uma vez mais, a execução atinge a eficiência máxima quando a CPU apresenta uma contribuição de 60% (ver figura 54). Apesar de a contribuição ótima relativa ser semelhante à obtida no teste anterior, não se encontrou justificação teórica para tal ocorrência. É, por isso, possível que os valores semelhantes sejam coincidência. Bastaria que o modelo da GPU usado fosse ligeiramente melhor para que o valor ótimo da contribuição da CPU fosse mais baixo. Ainda assim, a semelhança dos gráficos das figuras 54 e 53 pode significar que existe, por parte dos fabricantes, um cuidado para que exista um equilíbrio das capacidades computacionais dos vários dispositivos instalados no computador.

É ainda de notar que esta abordagem permitiu um tempo de execução de 325 milissegundos (para um teste com 1 segundo de vídeo 1080p, repetido 25 vezes), o que significou que pela primeira vez, no âmbito deste projeto, se tinha desenvolvido um algoritmo verdadeiramente apto para ser utilizado em tempo real.

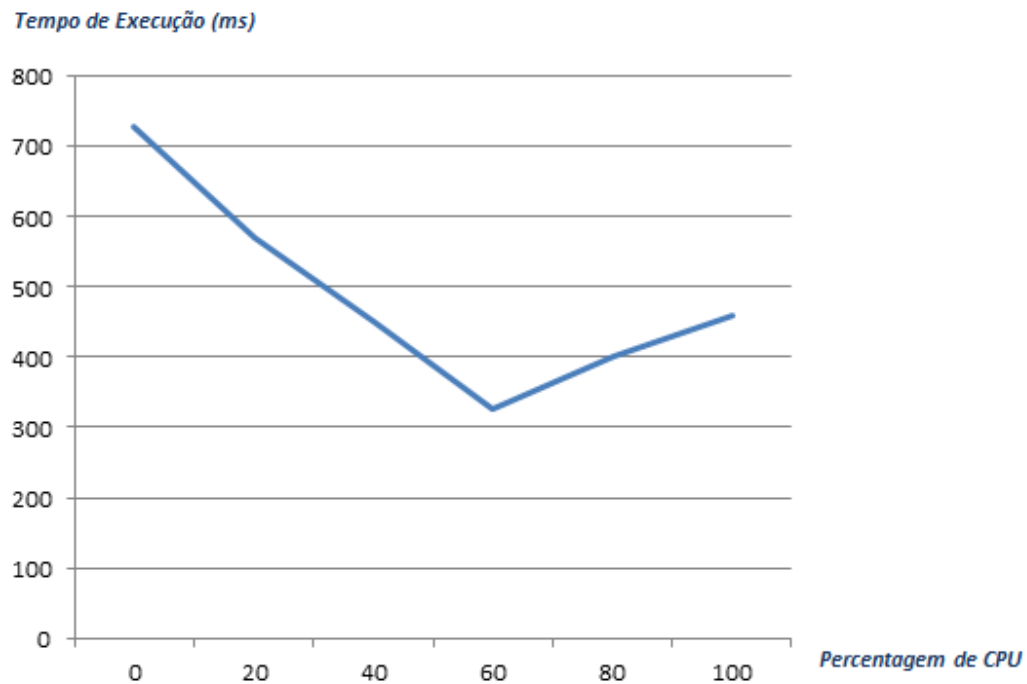


Figura 54: MMX e *Multithreading* – Efeitos da variação da contribuição da CPU

5.4.3 GPU, MMX e *Multithreading*

Visto que os resultados obtidos pela abordagem anterior foram muitíssimo satisfatórios, não se considerou necessário desenvolver um módulo que permitisse o processamento combinado destes 3 dispositivos. Apesar de tudo, através dos conhecimentos obtidos pela análise dos resultados anteriores, foi possível desenvolver um modelo que conseguisse estimar o ganho proporcionado pela adição da GPU. Uma vez que os tempos de computação dependem linearmente do tamanho do vídeo, pode-se estimar quanto tempo um módulo demoraria a executar (para uma determinada contribuição relativa). Visto que todos os módulos ficam à espera uns dos outros, o tempo total de execução será o máximo dos tempos de execução individuais. Para encontrar a contribuição ótima bastou simular vários cenários (com contribuições relativas diferentes) e escolher o cenário onde o tempo total de execução foi mínimo.

$$T_{cpusse} = 325ms$$

$$T_{gpu} = 2069ms$$

$$T_f = \max(fc * T_{cpusse}, (1 - fc) * T_{gpu})$$

Figura 55: Fórmula para estimação de tempos de execução com CPU, MMX e GPU

Neste sentido, utilizando a técnica descrita no parágrafo anterior (figura 55), calcularam-se vários tempos de execução para com fatores de contribuição diferentes (variável fc) e estima-se (com base nos dados obtidos de testes em que a GPU foi utilizada) que a adição deste componente ao módulo final poderia significar um aumento de 10% da eficiência do motor de anáglifos (fc tendo um valor de 90%).

5.5 Impacto da Otimização das Fórmulas de Criação de Anáglifos

O processo de otimização das fórmulas revelou-se bastante vantajoso. Ao analisar as métricas de processamento dos testes da secção “MMX e *Multithreading*” e “GPU e *Multithreading*” verificou-se uma melhoria de desempenho superior a 65%. A execução de um *kernel* otimizado passou a demorar sensivelmente 450 milissegundos, enquanto a versão menos eficiente demorava aproximadamente 1300 milissegundos.

5.6 Qualidade dos Anáglifos Produzidos

Uma vez que não existem algoritmos para medir a qualidade de um anáglifo, inicialmente recorreu-se a um processo qualitativo para testar os vídeos produzidos, e foi consensual que eles induziram uma boa sensação de profundidade aos observadores. Por outro lado, também houve a oportunidade de comparar esses mesmos vídeos com *software* de produção de anáglifos existente. Neste cenário, foi usada uma ferramenta presente no Media Composer. Constatou-se que os vídeos anáglifos gerados pelo *software* externo foram idênticos aos de teste. Desta forma se pode concluir que as equações deduzidas anteriormente estão corretas e podem ser usadas na produção de anáglifos Vermelho/Ciano para o espaço de cor YUV (especificações BT.601 e BT.709).

5.7 Qualidade dos Ficheiros “MXF 3D” para Ambientes Avid

Uma forma de avaliar se as essências de teste eram bem guardadas em MXF era através da importação dos produtos encapsulados para ferramentas da Avid. No âmbito desta tese, foi utilizado o Media Composer para esse efeito. Tal como esperado, não houve qualquer problema no processo de importação. Foram testadas essências de vídeo, áudio (com até 16 canais) e *timecode*. Os *assets* testados comportaram-se como se tivessem sido gerados internamente pelas ferramentas da Avid. Comprovou-se, por isso, que a arquitetura descoberta e apresentada anteriormente consegue definir um *asset* “MXF 3D”.

Capítulo 6

Integração com mxfSPEEDRAIL

6.1 Visão Geral

Tal como tinha sido previsto, houve uma preocupação para integrar o *software* desenvolvido com o mxfSPEEDRAIL S1000. Este processo decorreu sem problemas inesperados e foi concluído com sucesso. Neste capítulo será feita uma breve descrição das decisões que mais influenciaram as tarefas de integração. Será feita, sempre que se afigurar relevante para este tema, uma apresentação da arquitetura geral de alguns componentes.

6.2 Módulo de Aquisição

O S1000 possui um módulo que possibilita a manipulação de placas DeckLink para receção e difusão de sinal. Na figura 56, este módulo aparece representado a azul e é designado “DeckLinkFSyncHandler”. Além de implementar a *callback* de receção de frames, áudio, *timecode* e outros metadados, ele é responsável por gerir e resolver erros que surjam na transmissão, como por exemplo a corrupção de *frames* ou a perda temporária da ligação via SDI.

Sempre que uma *frame* chega à estação de captura, é despoletado um evento no S1000 e uma *callback* é executada. Esta função é responsável por extrair a informação da memória alocada pela placa e preparar a amostra recebida para ser posteriormente injetada numa *pipeline* que será responsável por dar continuidade ao processamento. É também nesta fase que é computado o anáglifo do par estereoscópico recebido o qual será enviado para o módulo de visualização (figura 56).

6.3 Módulo de Encapsulamento

O módulo de encapsulamento é um pouco mais complexo do que o anterior, sendo composto por variados constituintes. Os de primeira ordem, apresentados na figura 56, são responsáveis por receber as amostras UYUV e transcodificá-las para um ou vários formatos comprimidos. O mxfsPEEDRAIL é capaz de fazer o *ingest* com vários *codecs* diferentes em simultâneo. No caso do mxfsPEEDRAIL 3D, os *codecs* de codificação disponíveis são o MPEG2-LGOP e o DNxHD.

Estando finalizada a etapa de transcodificação, o próximo passo será alimentar um componente (designado *Generator*), responsável por preparar o formato de encapsulamento. Existem muitos formatos de encapsulamento disponibilizados pelo S1000. Entre eles estão o MP4, o MXF da Sony ou o MXF da Avid. No âmbito da tese, escolheu-se o formato MXF da Avid para encapsular a informação.

Tipicamente, em *assets* normais, o AvidGenerator instancia um AvidWrapperPipe que será responsável por criar um conjunto de *wrappers*. Para cada essência presente no *asset* haverá um AvidWrapper e, consequentemente um ficheiro MXF (figura 56). No entanto, todos os ficheiros MXF associados a um WrapperPipe partilham os mesmos metadados. No caso do 3D, os metadados não são iguais nos MXFs que guardam as essências vídeo. Por isso, optou-se por instanciar dois objetos do tipo AvidWrapperPipe, sendo cada um responsável pelo encapsulamento de um canal de vídeo. Foi ainda definido que o AvidWrapperPipe referente ao olho esquerdo iria estar sempre associado a todas as essências não visuais (áudio, por exemplo). Ou seja, o AvidWrapperPipe direito só seria responsável por coordenar o encapsulamento de uma essência (o vídeo do olho direito). Todas as outras são responsabilidade do AvidWrapperPipe esquerdo.

Adicionalmente, cada AvidWrapperPipe é responsável por criar um ficheiro AAF para representar o *asset*. No caso do S1000 3D, são criados inicialmente dois ficheiros AAF (um para cada AvidWrapperPipe) e que são numa fase posterior fundidos num terceiro. Desta forma, o utilizador pode optar por importar separadamente para o Media Composer dois *assets* 2D ou um *asset* 3D.

Numa última fase, estando o processo de *ingest* finalizado, os *assets* são encaminhados para os seus destinos de armazenamento (que poderão ser remotos ou não).

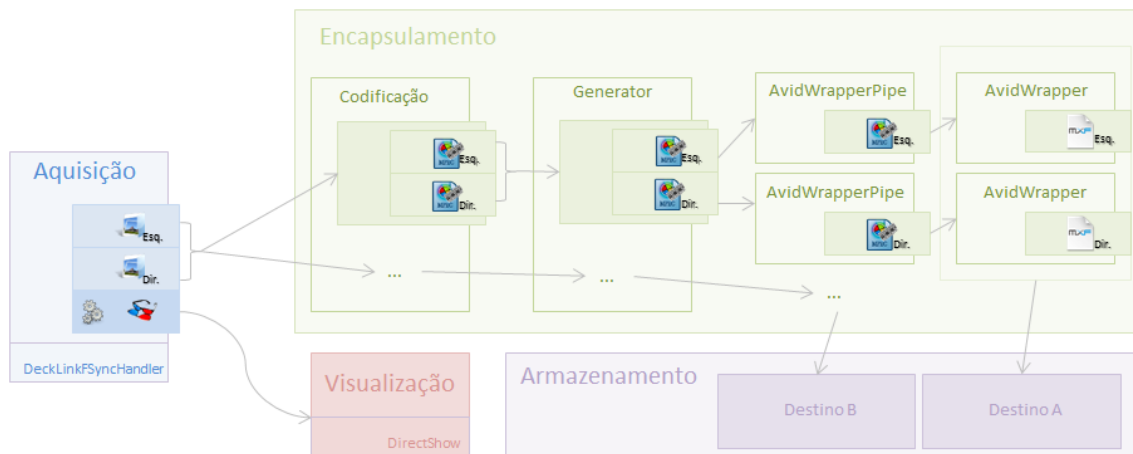


Figura 56: Exemplo de *pipeline* de ingest de vídeo 3D

6.4 Módulo de Visualização

O módulo de visualização é responsável por receber o anáglifo e mostrá-lo no monitor. Este processo é feito com auxílio da ferramenta DirectShow, da Microsoft. O anáglifo, por sua vez, é computado com o motor de anáglifos de processamento combinado (*multithreading* e SSE) desenvolvido durante a fase de testes. O tempo de computação do anáglifo no mxfsPEEDRAIL S1000 é da ordem dos 5 milissegundos por *frame*.

Capítulo 7

Conclusões e Trabalho Futuro

7.1 Trabalho Realizado

O trabalho desenvolvido cumpriu todos os objetivos definidos inicialmente. Foi possível determinar um processo de aquisição sincronizada de vídeo estereoscópico. Foi ainda possível encapsular em MXF os dados adquiridos (vídeo, áudio, *timecode* e outros metadados), tendo também sido bem sucedida a importação do *asset* 3D para *software* de edição de vídeo. Por fim, a produção de vídeo anáglifo em tempo real revelou-se eficaz.

Houve ainda a oportunidade de realizar tarefas não previstas no plano de trabalho inicial, como testes comparativos de rapidez de computação com SSE e *multithreading* de CPU, assim como a combinação do poder de processamento dos vários dispositivos. A utilização destes dispositivos nos testes de rapidez foi benéfica, pois foi possível ter uma ideia mais realista das capacidades reais que os sistemas computacionais atuais possuem. Constatou-se que a utilização da GPU poderia trazer melhor desempenho ao módulo de produção de anáglifos, contudo a melhoria não foi tão extensa quanto se esperava inicialmente para o modelo da GPU testado. Uma outra tarefa que foi executada e que não estava prevista de início foi a criação de ficheiros AAF para facilitar a importação do *asset* 3D para o Avid Media Composer. A decisão de implementar esta funcionalidade revelou-se bastante acertada pois facilitou grandemente a usabilidade do produto final.

Além de se terem cumprido os objetivos, foi também produzido um produto robusto que tem como virtude a integração de todas as funcionalidades já disponibilizadas pelo *mxfsPEEDRAIL S1000* com as capacidades de *ingest* de conteúdos 3D. É de realçar que o produto desenvolvido promete satisfazer as necessidades de produtores de programas para televisão, oferecendo-lhes uma ferramenta inovadora e, por enquanto única, que lhes permitirá produzir também em 3D sem que isso lhes traga grandes custos adicionais.

7.2 Trabalho Futuro

Apesar de ter sido bem sucedido, o projeto poderá ser futuramente estendido para implementar um conjunto mais vasto de funcionalidades. A comunidade de produtores televisivos anseia pela aparição de uma norma internacional que estabeleça um procedimento universal para encapsulamento de conteúdos audiovisuais 3D em MXF. Esta norma terá de ser eficaz em cenários de arquivamento dos conteúdos assim como em edição de vídeo.

A ausência de um formato S3D normalizado faz com que cada empresa possua o seu formato de armazenamento. Atualmente, além da Avid são muito escassas as soluções existentes no mercado. No entanto, é possível que num futuro próximo surjam novos métodos para encapsular vídeo 3D em MXF. Será importante que o mxfSPEEDRAIL seja capacitado para lidar com todos eles.

Por outro lado, poderão ser exploradas outras formas para mostrar o vídeo estereoscópico ao utilizador. Por exemplo, seria interessante explorar os novos monitores de luz polarizada. Vídeo 3D polarizado é mais promissor em termos de qualidade de imagem do que o anáglifo. O aparecimento de óculos como os “Google Glasses” também poderá ser uma boa oportunidade para este meio, pois poderá abrir portas para novas formas de canalizar cada canal de vídeo para cada um dos olhos.

Referências

- [1] P. Ferreira, “MXF - a progress report,” *EBU TECHNICAL REVIEW*, 2010.
- [2] Eurostars, “S3D TV Pipeline Annex,” 2010.
- [3] A. J. Wilt, “The DV, DVCAM, & DVCPRO Formats,” 16 Julho 2006. [Online]. Available: <http://www.adamwilt.com/DV-FAQ-tech.html>. [Acedido em 29 Janeiro 2012].
- [4] Sony, “XDCAM Family,” Fevereiro 2009. [Online]. Available: http://ws.sel.sony.com/PIPWebServices/RetrievePublicAsset/StepID/SEL-asset-152827/original/All_XDCAM_Family_V_2406_A.pdf. [Acedido em 29 Janeiro 2012].
- [5] Sony, “XDCAM HD Family,” Fevereiro 2009. [Online]. Available: http://ws.sel.sony.com/PIPWebServices/RetrievePublicAsset/StepID/SEL-asset-154478/original/XDCAM_HD_Family_V-2383-A.pdf. [Acedido em 29 Janeiro 2012].
- [6] Panasonic, “AVC-Intra (H.264 Intra) Compression,” 7 Setembro 2007. [Online]. Available: ftp://ftp.panasonic.com/pub/Panasonic/Drivers/PBTS/papers/WP_AVC-Intra.pdf. [Acedido em 29 Janeiro 2012].
- [7] P. Ferreira, “Video Coding,” Porto, 2009.
- [8] Society of Motion Picture and Television Engineers, Ancillary Data Packet and Space Formatting, Society of Motion Picture and Television Engineers, 1998.
- [9] B. Devlin, J. Wilkinson, M. Beard, P. Tudor e N. Wells, *The MXF Book*, Focal Press, 2006.
- [10] B. Devlin, “MXF - the Material eXchange Format,” *EBU TECHNICAL REVIEW*,

2002.

- [11] P. Ferreira, "MXF - a technical overview," *EBU TECHNICAL REVIEW*, 2010.
- [12] J. Wilkinson e B. Devlin, "The Material Exchange Format (MXF) and its Application," *SMPTE Journal*, pp. 378-384, 2002.
- [13] Society of Motion Picture & Television Engineers, "SMPTE 379M: Generic Container," Society of Motion Picture & Television Engineers, 2004.
- [14] Society of Motion Picture & Television Engineers, "Material Exchange Format (MXF) - File Format Specification (Standard)," Society of Motion Picture & Television Engineers, 2004.
- [15] SMPTE, "Material Exchange Format (MXF) - Operational Pattern 1a (Single Item, Single Package)," 2004.
- [16] MOG Technologies, "MOG Technologies - Ingest Solutions," 2012. [Online]. Available: <http://www.mog-technologies.com/>. [Acedido em 29 Janeiro 2012].
- [17] Blackmagic Design, "Blackmagic Design: DeckLink," 2012. [Online]. Available: <http://blackmagic-design.com/products/decklink/>. [Acedido em 29 Janeiro 2012].
- [18] A. Silva, *mxfSPEEDRAIL S1000 - User Manual*, 2011.
- [19] E. Trucco e A. Verri, *Introductory Techniques for 3-D Computer Vision*, Prentice Hall, 1998.
- [20] Society of Motion Picture & Television Engineers, "3D in MXF for Operations - Common Provisions," Society of Motion Picture & Television Engineers, 2012.
- [21] Society of Motion Picture & Television Engineers, "3D in MXF for Operations - OP1a Mapping," Society of Motion Picture & Television Engineers, 2012.
- [22] M. Doneus e K. Hanke, "Anaglyph Images - Still a good way to look at 3D-Objects?," *Proceedings of the 17th CIPA Colloquium: Mapping and Preservation for the New Millenium*, 1999.
- [23] A. J. Woods e T. Rourke, "Ghosting in Anaglyphic Stereoscopic Images," *Stereoscopic Displays and Virtual Reality Systems XI*, 2004.
- [24] SCEC, "Making Anaglyph Images in Adobe Photoshop," SCEC, 2002. [Online]. Available: <http://www.scec.org/geowall/index.html>. [Acedido em 6 Junho 2012].
- [25] Turner e Hellbaum, "LC shutter glasses provide 3-D display for simulated flight," *Information Display Magazine*, 1986.
- [26] L. Edwards, "Active Shutter 3D Technology for HDTV," 2009 Setembro 2009. [Online]. Available: <http://www.physorg.com/news173082582.html>. [Acedido em 29

- Janeiro 2012].
- [27] K. Lizuka, “3D Displays,” 2006. [Online]. Available: <http://individual.utoronto.ca/iizuka/research/cellophane.htm>. [Acedido em 29 Janeiro 2012].
 - [28] N. Dodgson, “Autostereoscopic 3D Displays,” *IEEE Computer* 38 (8), p. 31–36, 2005.
 - [29] I. Foster, *Designing and Building Parallel Programs*, Addison-Wesley, 1995.
 - [30] J. Nickolls e W. J. Dally, “THE GPU COMPUTING ERA,” *IEEE Computer Society*, pp. 56-69, 2010.
 - [31] J. Carvalho, *CUDA*, Porto, 2010.
 - [32] P. Du, R. Weber, P. Luszczek, S. Tomov e G. P. a, “From CUDA to OpenCL: Towards a performance-portable solution,” *Elsevier B.V.*, 2011.
 - [33] Kamran, K. N. G., Dickson e F. Hamze, “A Performance Comparison of CUDA and OpenCL,” *CoRR*, 2010.
 - [34] Wikipedia, “Multithreading (computer architecture),” Wikipedia, 28 Maio 2012. [Online]. Available: [http://en.wikipedia.org/wiki/Multithreading_\(computer_architecture\)](http://en.wikipedia.org/wiki/Multithreading_(computer_architecture)). [Acedido em 6 Junho 2012].
 - [35] Microsoft, “MMX, SSE, and SSE2 Intrinsics,” Microsoft, 2012. [Online]. Available: [http://msdn.microsoft.com/en-us/library/y0dh78ez\(v=vs.71\).aspx](http://msdn.microsoft.com/en-us/library/y0dh78ez(v=vs.71).aspx). [Acedido em 6 Junho 2012].
 - [36] Blackmagic Design, “SDK– DeckLink,” Blackmagic Design, 2011.
 - [37] AAF Association, “AAF COM API: Main Page,” AAF Association, 17 Fevereiro 2005. [Online]. Available: <http://aaf.sourceforge.net/docs/aafapiman/>. [Acedido em 6 Junho 2012].
 - [38] AMWA, “Advanced Authoring Format (AAF) - Object Specification v1.1,” AMWA, 2005.
 - [39] LLPanorama, “CUDA Tutorial,” 2008. [Online]. Available: <http://llpanorama.wordpress.com/cuda-tutorial/>. [Acedido em 6 Junho 2012].
 - [40] NVIDIA, “GeForce 8500,” NVIDIA, 2012. [Online]. Available: http://www.nvidia.com/object/geforce_8500.html. [Acedido em 6 Junho 2012].
 - [41] NVIDIA, “NVIDIA Quadro 6000,” NVIDIA, 2012. [Online]. Available: <http://www.nvidia.com.br/object/product-quadro-6000-br.html>. [Acedido em 6 Junho

- 2012].
- [42] formboss.net, “SSE Intrinsic Tutorial,” formboss.net, 22 Outubro 2010. [Online]. Available: <http://www.formboss.net/blog/2010/10/sse-intrinsic-tutorial/>. [Acedido em 6 Junho 2012].
 - [43] T. Borel, “Proposal for Standardizing Dense Disparity Maps,” 2010.
 - [44] C. Poynton, Technical Introduction to Digital Video, John Wiley & Sons, 1996.
 - [45] Webster, “Chapter Eleven - The MMX Instruction Set,” [Online]. Available: <http://webster.cs.ucr.edu/AoA/Windows/HTML/TheMMXInstructionSet.html>. [Acedido em 6 Junho 2012].
 - [46] S. Oshima, K. Kise, T. Katagiri e T. Yuba, “Parallel Processing of Matrix Multiplication in a CPU and GPU Heterogeneous Environment,” em *HIGH PERFORMANCE COMPUTING FOR COMPUTATIONAL SCIENCE - VECPAR 2006 7th International Conference, Rio de Janeiro, Brazil, June 10-13, 2006, Revised Selected and Invited Papers*, Springer Berlin / Heidelberg, 2007, pp. 305-318
 - [47] B. Boothe e A. Ranade, “Improved multithreading techniques for hiding communication latency in multiprocessors,” em *ISCA '92 Proceedings of the 19th annual international symposium on Computer architecture*, Nova Iorque, ACM, 1992, pp. 214-223
 - [48] A. Agarwal, “Performance Tradeoffs In Multithreaded Processors,” *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*, vol. 3, pp. 525-539, 1991