Universidade da Beira Interior
Departamento de Informática

Nº 112 - 2019: *Codificação da Cor de Hologramas Digitais Usando Multivistas*

Elaborado por:

Raquel Sofia Brás Guerra

Orientador:

Professora Doutora Maria Manuela Areias da Costa Pereira de Sousa

30 de Agosto de 2020

Agradecimentos

Conteúdo

Conteúdo **iii**

Lista de Figuras **v**

Lista de Tabelas **vii**

1 Introdução **1**
1.1 Enquadramento 1
1.2 Motivação 1
1.3 Objetivos 2
1.4 Organização do Documento 2

2 Estado da Arte **3**
2.1 Introdução 3
2.2 Perspetiva Histórica 3
2.3 Conceitos Base 3
2.3.1 Holografia 3
2.3.1.1 Princípios de Holografia 3
2.3.1.2 Representação de Dados Holográficos 4
2.3.1.3 Reconstrução de Holograma 4
2.3.2 Compressão 4
2.3.3 JPEG2000 4
2.3.4 Cor 4
2.3.4.1 *Red, Green & Blue* (RGB) 4
2.3.4.2 YCbCr 5
2.4 Estado da Arte 5
2.5 Conclusões 6

3 Tecnologias e Ferramentas Utilizadas **7**
3.1 Introdução 7
3.2 Tecnologias e Ferramentas 7
3.2.1 Software de Reconstrução de Hologramas e sua Transcrição para Python 8

3.2.2 *Kakadu Software* 8
3.3 Materiais Utilizados 9
3.3.1 *Hardware* 9
3.3.2 Hologramas 10
3.4 Conclusões 12

4 Etapas de Desenvolvimento e Implementação 13
4.1 Introdução 13
4.2 Reconstrução dos Hologramas 14
4.3 Compressão dos Hologramas Reconstruídos 16
4.3.1 Breve Estudo do *Kakadu Software* 17
4.3.2 Implementação e Execução do *Script* de Compressão . 17
4.3.3 Automatização dos *Scripts* 18
4.4 Determinação das Métricas de Compressão 19
4.4.1 Criação de gráficos 20
4.5 Conclusões 20

5 Testes e Resultados 21
5.1 Introdução 21
5.2 Resultados 21
5.3 Conclusões 21

6 Conclusões e Trabalho Futuro 23
6.1 Conclusões Principais 23
6.2 Trabalho Futuro 23

A Documentação de Funções 25
A.1 Função `load_hologram` 25
A.2 Função `propagate_asm` 26
A.3 Função `reconst_hologram` 27
A.4 Função `reconst_16views` 28
A.5 Função `compress_views` 29
A.6 Função `cod_jpeg2000` 30
A.7 Função `dec_jpeg2000` 31
A.8 Função `psnr` 32

Lista de Figuras

3.1 Holograma `Dices4k` (imagem original). 11
3.2 Holograma `DiffuseCar4k` (imagem original). 11
3.3 Holograma `Piano4k` (imagem original). 12

4.1 Diagrama do processo efetuado sobre um holograma para obtenção dos resultados. 15
4.2 Diagrama de dependências da função `reconst_16views`. 16
4.3 Diagrama de dependências da função `compress_views`. 18

Lista de Tabelas

A.1 Documentação da função `load_hologram`. 25
A.2 Documentação da função `propagate_asm`. 26
A.3 Documentação da função `reconst_hologram`. 27
A.4 Documentação da função `reconst_16views`. 28
A.5 Documentação da função `compress_views`. 29
A.6 Documentação da função `cod_jpeg2000`. 30
A.7 Documentação da função `dec_jpeg2000`. 31
A.8 Documentação da função `psnr`. 32

Acrónimos

ASM *Angular Spectrum Method*

JPEG *Joint Photographic Experts Group*

JSON *JavaScript Object Notation*

kdu *Kakadu Software*

PSNR *Peak signal-to-noise ratio*

RGB *Red, Green & Blue*

SDK *Software Development Kit*

Capítulo

1

Introdução

1.1 Enquadramento

A história da captura, armazenamento e visualização de imagens é extremamente rica e milenar. Marcos importantes destacam-se, sendo do particular interesse no Século XXI os grandes passos dados na imagem digital.

Contudo, a vasta maioria da fotografia tem-se centrado na captura de imagens estáticas em duas dimensões. O interesse na captura e representação de objetos e momentos em três dimensões tem ganho um interesse crescente nas últimas décadas.

A área dedicada ao estudo deste modelo, a **holografia**, carece de vários marcos que já fazem parte do quotidiano da fotografia clássica, nomeadamente padrões *standardizados* para a codificação e compressão de **hologramas** em formato digital.

1.2 Motivação

Dada a referida ausência de *standards* no armazenamento e representação da informação, reconstrução e codificação de um holograma, é do interesse da comunidade do JPEG Pleno estudar os codificadores existentes para melhor perceber qual a sua adaptabilidade aos hologramas e quais as modificações necessárias para resolver a falta de padrões nos pontos mencionados.

1.3 Objetivos

Tendo em mente a motivação apresentada na secção 1.2, o presente projeto tem por objetivo principal investigar o desempenho do codec JPEG2000 na codificação de hologramas a cores em multivistas.

Por seu turno, os objetivos secundários — os quais refletem as diferentes fases da investigação — são os seguintes:

1. Implementar um reconstrutor para hologramas com cor;
2. Reconstruir hologramas em vários pontos de vista;
3. Comprimir hologramas reconstruidos com recurso ao codificador JPEG2000;
4. Avaliar a qualidade das imagens comprimidas face à reconstrução original.

Os objetivos supra-mencionados refletem o objetivo geral de estudar holografia e, assim, expandir o conhecimento na área das tecnologias multimédia.

1.4 Organização do Documento

TODO

Capítulo

2

Estado da Arte

2.1 Introdução

TODO

2.2 Perspetiva Histórica

TODO

2.3 Conceitos Base

TODO

2.3.1 Holografia

Quando um objeto é iluminado, a luz é dispersa pela superfície desse objeto, criando uma onde. Esta onda contém toda a informação sobre a luz: a amplitude define o brilho e a fase representa a forma do objeto. Enquanto as fotografias clássicas gravam apenas a intensidade da luz, um holograma preserva a fase do objeto através das características de interferência e difração da luz, guardando assim toda a informação necessária à reconstrução 3D do objeto original.

2.3.1.1 Princípios de Holografia

O principio de holografia foi descoberto em 1948 pelo físico Dennis Gabor enquanto investigava microscopia de eletrões.

Ao contrário da fotografia convencional, que permite a captura da intensidade da luz, holografia permite guardar a amplitude e a fase da onde de luz dispersa por um objeto. Com a iluminação correta, o holograma produz a onda de luz original, permitindo ao utilizador observar o objeto tal como se estivesse fisicamente presente.

2.3.1.2 Representação de Dados Holográficos

Os dados holográficos podem ser representados de várias formas. Embora sejam todas equivalentes no sentido em que representam o mesmo objeto, algumas tornam a compressão mais eficiente.

No âmbito deste projeto, apenas é relevante a representação no campo de onda complexo.

- Dados reais e imaginários — Utiliza um sistema de coordenadas cartesiano para representar amplitudes complexas;
- Dados da amplitude e fase — Os valores complexos são expressos num sistema de coordenadas polares.

Os hologramas utilizados neste projeto são representados pelo formato de amplitude-fase.

2.3.1.3 Reconstrução de Holograma

TODO

2.3.2 Compressão

TODO

2.3.3 JPEG2000

TODO

2.3.4 Cor

TODO

2.3.4.1 *Red, Green & Blue* (RGB)

TODO

2.3.4.2 YCbCr

TODO

2.4 Estado da Arte

Primeira proposta para codificação digital de hologramas data 1991, Sato et al. captura franjas holográficos usando uma câmara que foram por sua vez modulados em sinal TV e transmitidos para um recetor [1]. (captured the holographic fringes using a camera, which was then modulated into a TV signal and transmitted to the receiver.);

Em 1993, Yoshikawa notou que não era prática a aplicação da compressão de imagem 2d diretamente no holograma. Propôs a compressão do holograma em segmentos que correspondem a diferentes perspetivas de reconstrução. Segmentos foram comprimidos com MPEG-1 e MPEG-2 [2,3]. (ver resultados)

Em 2002, Naughton et al. estudou a compressibilidade da holografia digital de mudança de fase usando vários algoritmos de compressão sem perdas [4]. Concluiram que são esperadas melhores taxas de compressão quando o holograma digital é codificado em componentes reais e imaginarias independentemente.

Em [4] foram também estudadas outras técnicas de compressão com perdas tais como subamostragem e quantificação, sendo a última muito eficaz. A eficácia da quantização tanto na simulação numérica como na ótica foi confirmada por Mills e Yamaguchi [5].

A quantização no domínio da reconstrução (não sei o que isto quer dizer) de hologramas de mudança de fase de foram analisados por Darakis and Soraghan [6].

Naughton et al. em 2003 e Darakis et al. em 2006 demonstraram que a aplicação direta de wavelets standard em hologramas não é muito eficiente, visto que as wavelets standard são tipicamente usadas no processamento de sinais com poucas variações (smooth signals). Propuseram a utilização de uma outra familia de wavelets Fresnelets. Fresnelets foram também aplicadas em 2003 por Liveling et al. [8]

Em 2006, Seo et al. propôs comprimir segmentos do holograma usando multi-vistas e temporal prediction dentro de MPEG-2 modificado.

Em 2010 Darkis et al. Determinaram a taxa de compressão mais elevada que pode ser obtida em hologramas mantendo uma qualidade de reconstrução visualmente sem perdas. Nos seus ensaios foram usados MPEG-4 e Dirac. Na informação amplitude-fase foi aplicado um método multiple description

coding utilizando máximo à posterior. Mostrou-se um mecanismo poderoso para mitigar erros no canais em hologramas digitais.

Em 2013 Blinder investigou a decomposição alternativa em hologramas off-axis. Em 2014 Still, Xing e Dufaux estudaram codificação sem perdas baseada em quantização vetorial.

Recentemente Peixeiro et al. [9] realizou um benchmark dos codificadores standard de imagens aplicados em hologramas digitais, em conjunto com os formatos de representação principais. Foram comparados os seguintes codificadores de imagem padrão JPEG; JPEG 2000; H.264/AVC intra; HEVC intra. Os autores concluiram que os melhores formatos de representação são phase-shiffted e real-imaginário

Em 2016, Dufaux review o estado da arte da compressão de hologramas digitais

2.5 Conclusões

TODO

Capítulo

3

Tecnologias e Ferramentas Utilizadas

3.1 Introdução

O projeto apresentado só foi possível ser concluído devido à existência de tecnologias e ferramentas, assim como de materiais, os quais se revelaram essenciais.

Em particular, este Capítulo aborda:

- O *software* de reconstrução de hologramas;
- A escolha da linguagem de programação para o projeto;
- O *Software Development Kit* (SDK) de codificação de imagens no formato JPEG2000;
- O *hardware* utilizado;
- Os hologramas testados.

3.2 Tecnologias e Ferramentas

O desenvolvimento do projeto envolveu o trabalho conjunto de diversas ferramentas, nomeadamente Python 3, *Kakadu Software* e *software* escrito no âmbito do projeto JPEG Pleno.

3.2.1 Software de Reconstrução de Hologramas e sua Transcrição para Python

Do 80º Encontro do Grupo JPEG, realizado em Berlim entre 7 e 13 de julho de 2018, resultou um software desenvolvido por Antonin Gilles e Patrick Gioia, do *Institute of Research & Technology b<>com*. O respetivo código, fornecido pela professora orientadora, encontra-se implementado em MATLAB.

Dada a ausência de uma licença do MATLAB para utilizar este *sofware* desenvolvido no âmbito do JPEG Pleno, foi necessário transcrever o código para uma nova linguagem de programação. Neste sentido, optou-se pelo Python 3.

O recurso a Python apresenta uma miríade de vantagens, entre elas:

- Linguagem *open source*;
- Utilização e distribuição gratuita da linguagem;
- Maior eficiência face ao MATLAB;
- Utilização comum no contexto de processamento multimédia e respetivos projetos;
- Vasto leque de bibliotecas, facilitando a implementação de *software* específico;
- Abundância de documentação;
- Forte comunidade *online* de suporte.

A escolha do Python foi, portanto, natural no âmbito do presente projeto.

Durante a fase de transcrição decorreu uma atualização do Python, tendo sido a última versão do código executada e testada na versão 3.8.2 em três distribuições GNU/Linux de 64 bits: Ubuntu 18.04 LTS, Fedora 31 e Linux Mint 20.

3.2.2 *Kakadu Software*

Uma vez que o JPEG2000 é o formato alvo deste projeto, e tendo em conta a dificuldade encontrada em projetos anteriores no contexto da holografia em utilizar a ferramenta *FFmpeg*, foi recomendado pela professora orientadora a utilização do *Kakadu Software*.

Este é um SDK para codificação e descodificação de imagens com recurso ao formato JPEG2000, segundo o *standard* pela *Joint Photographic Experts Group* (JPEG).

Os comandos deste SDK de relevo para o projeto são os seguintes:

1. `kdu_compress`: codifica uma imagem para o formato JPEG2000.
   Sintaxe:

```
kdu_compress -i input_file -o output_file -rate n
↪  Cycc=<yes|no> -precise -quiet
```

   onde:

   - `-i input_file`: imagem de *input*;
   - `-o output_file`: ficheiro de *output*;
   - `-rate n`: número de *bits* por amostra (*n* pode ser um número flutuante);
   - `Cycc=<yes|no>`: `yes` caso seja usada a codificação com transformada de cor de RGB para YCbCr, `no` em caso contrário;
   - `-precise`: força o uso de representações de 32 *bits*;
   - `-quiet`: suprime o *output* do programa.

2. `kdu_expand`: descodifica uma imagem no formato JPEG2000.
   Sintaxe:

```
kdu_expand -i input_file -o output_file -rate n -quiet
```

   onde:

   - `-i input_file`: imagem de *input*;
   - `-o output_file`: ficheiro de *output*;
   - `-rate n`: número de *bits* por amostra (*n* pode ser um número flutuante);
   - `-quiet`: suprime o *output* do programa.

3.3 Materiais Utilizados

3.3.1 *Hardware*

De notar que esta implementação do *software* de reconstrução de hologramas requer computadores com especificações mais generosas.

Para este projeto, dois computadores em particular executaram as várias iterações de desenvolvimento do *software* transcrito em Python:

1. *Desktop*: processador Ryzen™ 7 2700X 3.7–4.3GHz, placa gráfica NVidia® Quadro K5000 (4GB), memória RAM de 32GB e *swap* de 96GB, armazenamento SSD de 1TB;
2. Portátil: Intel® Core™ i5-10210U 1.6–4.2GHz, memória RAM de 16GB e *swap* de 8GB, armazenamento SSD de 512GB.

3.3.2 Hologramas

Os hologramas reconstruidos com o *software* desenvolvido no âmbito deste projeto são fornecidos pelo *Institute of Research & Technology b<>com*. De entre os disponíveis, foram utilizados os seguintes hologramas com as respetivas características:

1. `Dices4k` (Figura 3.1):
    - Resolução: 4096×4096;
    - *Pixel pitch*: 0.4 µm;
    - Comprimento de onda vermelho: 640 nm;
    - Comprimento de onda verde: 532 nm;
    - Comprimento de onda azul: 473 nm;
    - Localização da cena: entre 0.164 e 0.328 cm.
2. `DiffuseCar4k` (Figura 3.2):
    - Resolução: 4096×4096
    - *Pixel pitch*: 0.4 µm;
    - Comprimento de onda vermelho: 640 nm;
    - Comprimento de onda verde: 532 nm;
    - Comprimento de onda azul: 473 nm;
    - Localização da cena: entre 0.11 e 0.25 cm.
3. `Piano4k` (Figura 3.3):
    - Resolução: 4096×4096
    - *Pixel pitch*: 0.4 µm;
    - Comprimento de onda vermelho: 640 nm;
    - Comprimento de onda verde: 532 nm;
    - Comprimento de onda azul: 473 nm;
    - Localização da cena: entre 0.17 and 0.313 cm.

Figura 3.1: Holograma `Dices4k` (imagem original).

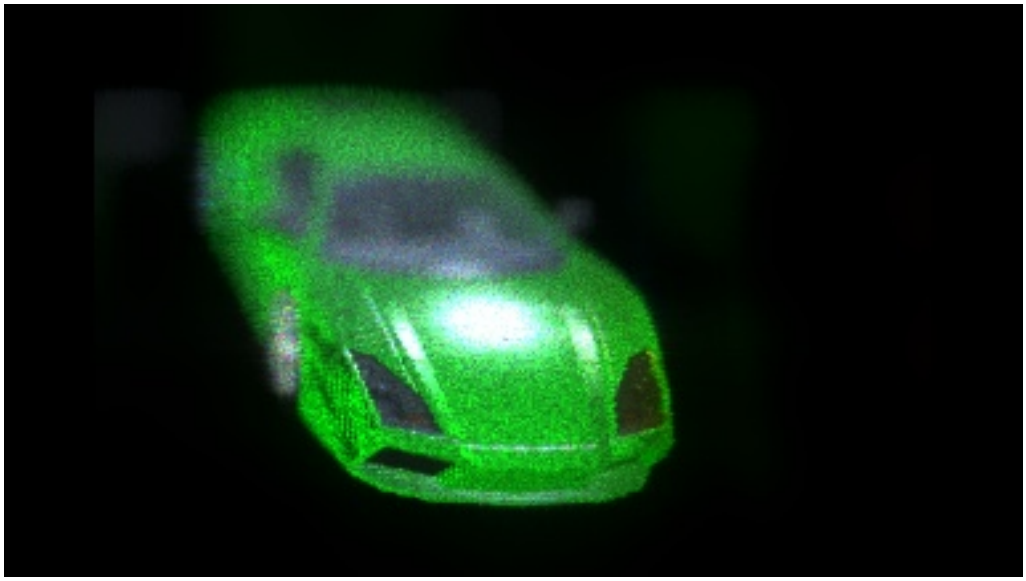

Figura 3.2: Holograma `DiffuseCar4k` (imagem original).

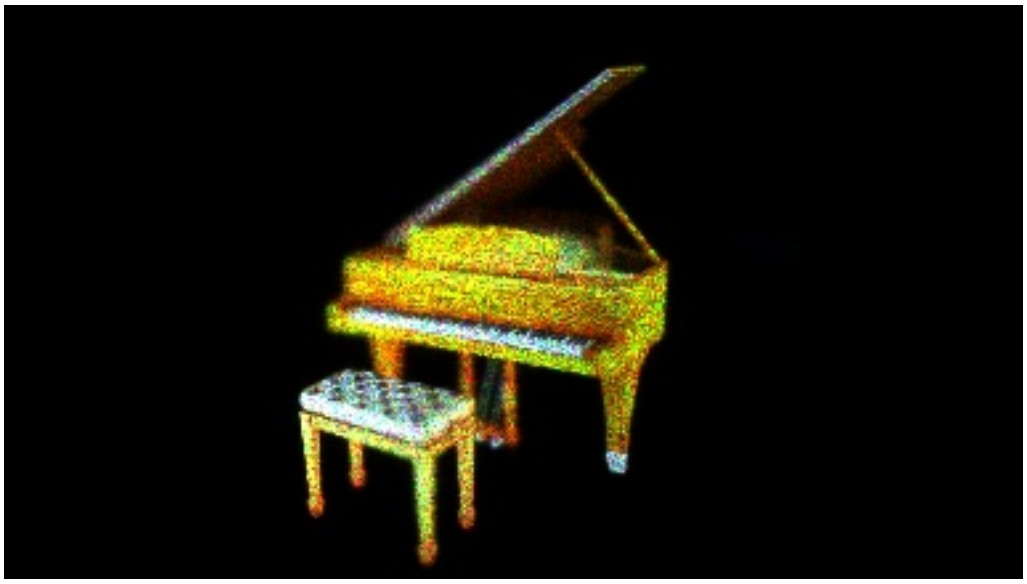

Figura 3.3: Holograma `Piano4k` (imagem original).

3.4 Conclusões

Após a seleção das tecnologias e materiais, conforme supra-mencionados, ir-se-á proceder no Capítulo 4 ao delineamento da estratégia de investigação do projeto, a qual está intimamente ligada às escolhas apresentadas no presente Capítulo, entre elas a escolha da linguagem Python para transcrição do *software* original de reconstrução de hologramas, o SDK para realizar a codificação no formato JPEG2000 e os hologramas testados.

Capítulo

4

Etapas de Desenvolvimento e Implementação

4.1 Introdução

Para o sucesso deste projeto foi essencial delinear uma estratégia de investigação. Esta divide-se em 3 partes: reconstrução dos hologramas, compressão dos hologramas reconstruídos, e determinação das métricas de compressão. A cada uma destas fases atribuíram-se objetivos para a sua profícua execução:

1. Reconstrução dos hologramas:
    a) Estudar a holografia;
    b) Estudar as funções originais em MATLAB do Projecto JPEG Pleno;
    c) Transcrever estas funções para a linguagem de programação Python;
    d) Comparar o *output* entre as funções originais em MATLAB e as respetivas transcrições em Python;
    e) Desenvolver um *script* para reconstruir cada holograma em 16 vistas distintas.
2. Compressão dos hologramas reconstruídos:
    a) Estudar o uso do *Kakadu Software* (kdu);
    b) Automatizar a execução dos *scripts* e *softwares* envolvidos;
    c) Implementar um *script* para compressão (com e sem transformada de cor em diferentes *bitrates*) e descompressão dos hologramas reconstruídos.

3. Determinação das métricas de compressão:
    a) Calcular o débito com a métrica *Peak signal-to-noise ratio* (PSNR) entre os hologramas originais e as imagens comprimidas;
    b) Determinar o melhor método de armazenamento dos débitos calculados;
    c) Implementar o *output* dos resultados no método selecionado;
    d) Gerar gráficos dos resultados.

As Secções subsequentes expõem o trabalho desenvolvido neste sentido e a Figura 4.1 esquematiza o processo levado a cabo para a execução do projeto.

4.2 Reconstrução dos Hologramas

O projeto iniciou-se com uma pesquisa exaustiva sobre a ciência da holografia, a qual resultou no Estado da Arte resumido no Capítulo 2.

Simultaneamente, foi efetuado um estudo das funções do *software* desenvolvido no âmbito do projeto JPEG Pleno a fim de se poder fazer a respetiva transcrição para Python. O resultado deste estudo e transcrição encontra-se exposto nas tabelas A.1, A.2 e A.3.

Estas funções foram compiladas num único *script* Python. A fim de cumprir o 2º objetivo secundário, uma nova função, `reconst_16views`, foi implementada (Tabela A.4). Esta função está, portanto, encarregue de abrir um holograma, as respetivas especificações e reconstruí-lo em exatamente 16 vistas distintas. De notar que as especificações são fornecidas por um ficheiro *JavaScript Object Notation* (JSON) cujo conteúdo inclui:

- `holo_size`: Resolução do holograma (em pixeis);
- `pitch`: Distância entre pixeis (em metros);
- `wavelenghts`: Lista com os comprimentos de onda respetivos aos canais RGB (em metros);
- `z`: Distância de reconstrução (em metros);
- `pupil_size`: Lista que contém o tamanho da janela de reconstrução (em pixeis).

Todavia, havendo uma transcrição de funções entre linguagens de programação, exigiu-se uma forma de comprovar que as funções transcritas em Python são equivalentes às originais em MATLAB. Para este fim, delineou-se a seguinte estratégia:

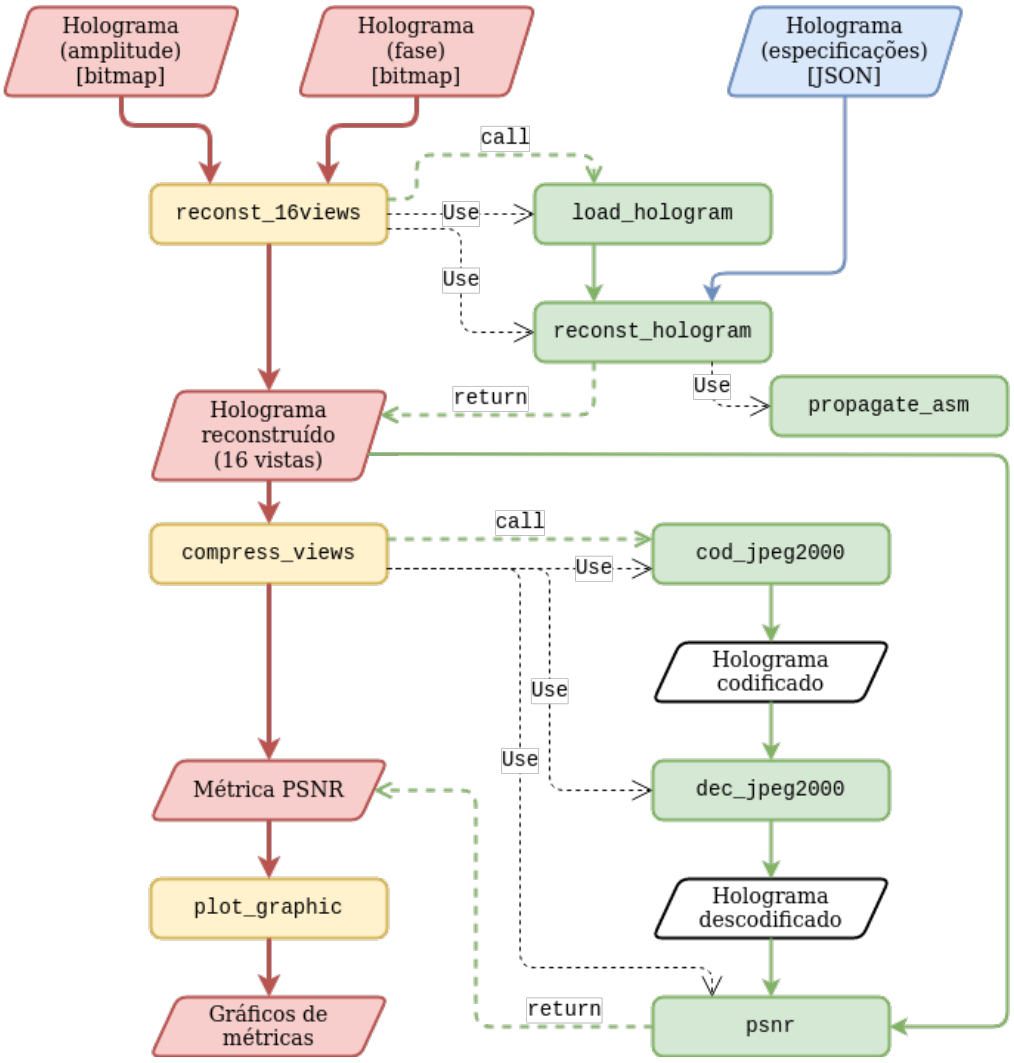

Figura 4.1: Diagrama do processo efetuado sobre um holograma para obtenção dos resultados.
O caminho vermelho/amarelo representa o trajeto principal pelo qual o holograma sofre processamento. Os caminhos verdes detalham os processos intermédios correspondentes a um determinado processo do trajeto principal (a entrada é dada por "*call*" e o retorno é dado por "*return*"). As dependências entre as principais funções são indicadas por "*Use*". A azul estão os ficheiros externos auxiliares.

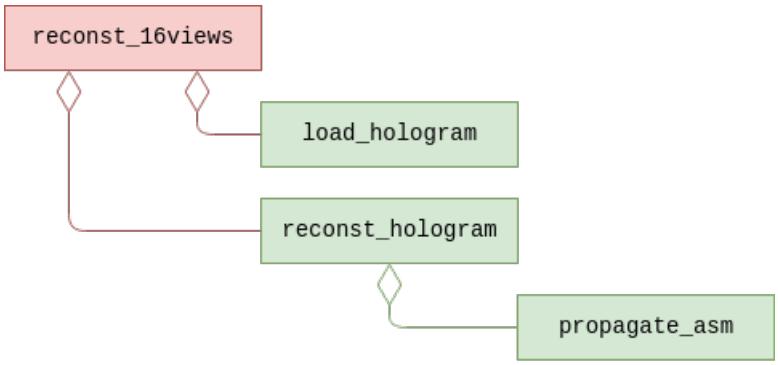

Figura 4.2: Diagrama de dependências da função `reconst_16views`.

1. Etapa de implementação:
   a) Transcrever as funções para Python (forme apresentado).
2. Etapa de reconstrução:
   a) Reconstruir os hologramas numa vista com as funções originais em MATLAB;
   b) Reconstruir os hologramas numa vista com as funções transcritas em Python.
3. Etapa de comparação e resultados:
   a) Comparar os hologramas produzidos pelos *scripts* das duas linguagens através de uma análise visual a olho nu;
   b) Comparar os hologramas produzidos pelos *scripts* das duas linguagens através da métrica PSNR.

Com o *script* devidamente implementado e testado, prosseguiu-se com a seguinte fase do projeto relativo à compressão dos hologramas agora reconstruídos.

4.3 Compressão dos Hologramas Reconstruídos

Após a reconstrução dos hologramas em multivista, segue-se o teste ao formato JPEG2000 através de compressão e descompressão de forma a comparar com a reconstrução do holograma original (cuja métrica será abordada na Secção 4.4).

Conforme a estratégia delineada na Secção 4.1, esta fase do projeto envolveu o estudo da ferramenta kdu, a implementação dos *scripts* e o cálculo das métricas.

4.3.1 Breve Estudo do *Kakadu Software*

A fase de estudo do kdu resultou no conhecimento exposto na Secção 3.2.2, onde se encontram descritos os dois comandos utilizados no âmbito do projeto.

4.3.2 Implementação e Execução do *Script* de Compressão

De forma a se poder comparar o holograma original com o holograma comprimido no formato JPEG2000, revela-se necessário realizar uma sequência de compressão e descompressão do primeiro. Tal irá dar origem a um holograma reconstruido após compressão, o qual poderá ser comparado com o original.

Conforme visto anteriormente, a ferramenta kdu é a ferramenta eleita para este processo. Contudo, não é prático executar manualmente a sequência de comandos na *shell*. Por conseguinte, um *script* escrito em Python foi implementado. Este inclui a função `compress_views` (Tabela A.5 e Figura 4.3), a qual executa a seguinte sequência de operações:

1. *Compressão do holograma original, previamente reconstruido.*
   É feita uma chamada ao sistema do comando `kdu_compress`.
   Esta encontra-se encapsulada na função auxiliar `cod_jpeg2000` (Tabela A.6).

2. *Descompressão do holograma agora comprimido.*
   É feita uma chamada ao sistema do comando `kdu_expand`.
   Esta encontra-se encapsulada na função auxiliar `dec_jpeg2000` (Tabela A.7).

3. *Cálculo do débito através de uma métrica de compressão.*
   O holograma original e o holograma agora descomprimido são comparados com recurso à métrica PSNR (Secção 4.4).
   Esta operação é realizada pela função auxiliar `psnr` (Tabela A.8).

De notar os *bitrates* testados: 0.1, 0.3, 0.6, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5 e 5.0. Há ainda a referir a realização dos testes com e sem transformada de cor.

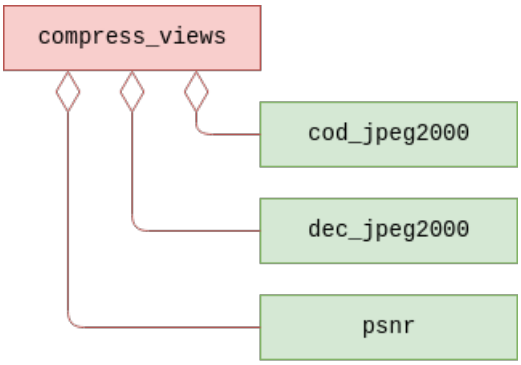

Figura 4.3: Diagrama de dependências da função `compress_views`.

Por fim, de forma a facilitar a extração dos resultados pretendidos segundo o objetivo primário do projeto (Secção 1.3), é gerado um ficheiro JSON por cada holograma. Este tem a seguinte estrutura:

```
<nome_holograma>.json
├── no_ycbcr
│   └── <rate> ..... Número de bits por amostra
│       ├── <vista>.ppm:  valor ..... Posição da janela de reconstrução
│       ├── avg:  valor ..... PSNR médio das vistas para este rate
│       └── std:  valor ..... Desvio padrão
└── ycbcr
    └── <rate>
        ├── <vista>.ppm:  valor
        ├── avg:  valor
        └── std:  valor
```

O ficheiro JSON supra-mencionado será portanto utilizado no processo descrito na Secção 4.4.

4.3.3 Automatização dos *Scripts*

Para a otimização do processo de investigação neste projeto, foram criados dois *scripts* adicionais, os quais trabalham em conjunto a fim de automatizar todo este processo. São eles os seguintes:

1. *Makefile*: Introduz as 8 opções infra-apresentadas, as quais invocam o *script* `main.py`:

a) `reconst`: reconstrói o holograma fornecido por argumento em 16 vistas (Capítulo 4.2 e Tabela A.4);

b) `compress`: executa o *script* apresentado na Secção 4.3.2;

c) `plot`: calcula os gráficos com os resultados de comparação da compressão com e sem transformada de cor (Secção 4.4);

d) `all`: executa os 3 itens anteriores;

e) `test`: opção utilizada na fase de *debugging*;

f) `install`: instala pacotes no sistema operativo essenciais à execução dos *scripts*;

g) `clean`: remove os diretórios relativos a: hologramas reconstruidos, *output* do kdu, ficheiros JSON dos resultados do PSNR, e ficheiros de *cache* do Python;

h) `clean-compressed`: remove os diretórios relativos a: *output* do kdu, ficheiros JSON dos resultados do PSNR, e ficheiros de *cache* do Python.

2. `main.py`: *script* Python que executa as primeiras 5 opções supra-mencionadas conforme o argumento passado pelo *Makefile*. Faz uso das funções `reconst_16views` (Tabela A.4) e `compress_views` (Tabela A.5).

4.4 Determinação das Métricas de Compressão

Na sequência do processo de compressão e descompressão no formato JPEG2000, e conforme previamente introduzido na Secção 4.3.2, a determinação das métricas de compressão constituem a última fase da investigação a fim destes resultados poderem ser escrutinados no Capítulo 5.

Variadas métricas estão disponíveis para se poder comparar amostras e, no caso do presente projeto, avaliar a qualidade do holograma após compressão no formato JPEG2000. De entre as métricas existentes, foi eleita a *Peak signal-to-noise ratio* (PSNR) por indicação da professora orientadora, a qual está envolvida no Projeto JPEG Pleno.

Conforme introduzido na Secção 4.3.2, foi utilizada uma função auxiliar `psnr` (Tabela A.8), invocada pela função principal `compress_views` (Tabela A.5). Este método de implementação foi escolhido para fins de eficiência de armazenamento.

Uma vez que não é mandatório armazenar os hologramas rescontruídos após compressão, sendo apenas do nosso interesse as métricas finais a si referentes, a função `dec_jpeg2000` exporta o holograma descomprimido num

ficheiro temporário `temp.ppm`. Este é utilizado de seguida pela função `psnr` para cálculo do PSNR (sendo os resultados armazenados num ficheiro JSON), sendo de seguida eliminado.

4.4.1 Criação de gráficos

O ficheiro JSON gerado (descrito na Secção 4.3.2) é posteriormente carregado num novo *script* Python, especificamente implementado para gerar gráficos em si baseados para mais fácil visualização, análise e discussão (Capítulo 5). Este *script* recorre à biblioteca `matplotlib.pyplot`.

A geração de um gráfico por holograma por parte deste *script* marca o fim do processo de investigação relativo à manipulação dos hologramas e extração de métricas.

4.5 Conclusões

Após a conclusão das 3 fases de investigação segundo a estratégia delineada inicialmente e ilustrada pela Figura 4.1, resta a exposição e análise dos resultados obtidos no Capítulo 5.

A estratégia desenhada revelou-se eficaz e permitiu a obtenção de *scripts* facilmente adaptáveis para trabalhos futuros no âmbito do Projeto JPEG Pleno, assim como os próprios resultados para análise por parte de outros interessados. De igual forma, a escolha de *standards* gratuitos e *open-source* (justificada na Secção 3.2) revelou-se acertada pelos mesmos motivos.

Capítulo

5

Testes e Resultados

5.1 Introdução

TODO

5.2 Resultados

TODO

5.3 Conclusões

TODO

Capítulo

6

Conclusões e Trabalho Futuro

6.1 Conclusões Principais

Esta secção contém a resposta à questão:
Quais foram as conclusões princípais a que o(a) aluno(a) chegou no fim deste trabalho?

6.2 Trabalho Futuro

Esta secção responde a questões como:
O que é que ficou por fazer, e porque?
O que é que seria interessante fazer, mas não foi feito por não ser exatamente o objetivo deste trabalho?
Em que outros casos ou situações ou cenários – que não foram estudados no contexto deste projeto por não ser seu objetivo – é que o trabalho aqui descrito pode ter aplicações interessantes e porque?

Apêndice

A

Documentação de Funções

A.1 Função `load_hologram`

Tabela A.1: Documentação da função `load_hologram`.

Nome da função `load_hologram`
Protótipo original em MATLAB `function [hologram] = load_hologram(ampli_path, phase_path)`
Protótipo transcrito em Python `def load_hologram(ampli_path, phase_path)`
Descrição Esta função carrega um holograma da base de dados b<>com a partir dos seus ficheiros de amplitude e fase.
Inputs `ampli_path`: Diretório do ficheiro da imagem da amplitude (caminho relativo ou absoluto). `phase_path`: Diretório do ficheiro da imagem da fase (caminho relativo ou absoluto).
Output Modulação complexa do holograma (3 canais: RGB).
Efeitos colaterais Não aplicável.
Dependências Não aplicável.

A.2 Função `propagate_asm`

Tabela A.2: Documentação da função `propagate_asm`.

Nome da função `propagate_asm`
Protótipo original em MATLAB `function [v] = propagate_asm(u, pitch, wavelength, z)`
Protótipo transcrito em Python `def propagate_asm(u, pitch, wavelength, z)`
Descrição Esta função simula a propagação no plano complexo `u` sobre a distância `z` utilizando o *Angular Spectrum Method* (ASM).
Inputs `u`: Campo de onda de luz do plano de *input* (um canal). `pitch`: Distância entre pixeis (em metros). `wavelength`: Comprimento de onda do canal de cor a propagar (em metros). `z`: Distância de propagação ao longo do eixo ótico (em metros).
Output Campo de onda de luz no plano de destino (um canal).
Efeitos colaterais Não aplicável.
Dependências Não aplicável.

A.3 Função `reconst_hologram`

Tabela A.3: Documentação da função `reconst_hologram`.

Nome da função `reconst_hologram`
Protótipo original em MATLAB `function [recons] = reconsHologram(hologram, pitch, wavelengths, z, pupilPos, pupilSize)`
Protótipo transcrito em Python `def reconst_hologram(hologram, pitch, wavelengths, z, pupil_pos, pupil_size)`
Descrição Esta função reconstrói o holograma a uma distância `z`, utilizando o ASM. Permite o uso de uma janela para obter reconstruções de diferentes pontos de vista.
Inputs `hologram`: Holograma de modulação complexa (3 canais: RGB). `pitch`: Distância entre pixeis (em metros). `wavelengths`: Comprimentos de onda de luz (em metros, 3 canais: RGB). `z`: Distância de reconstrução (em metros). `pupilPos`: Posição da janela (em pixeis, canto superior direito). *Valor por defeito.* `[0, 0]`. `pupilSize`: Tamanho da janela (em pixeis, altura × largura). *Valor por defeito.* `None`.
Output Reconstrução numérica do holograma (3 canais: RGB).
Efeitos colaterais Não aplicável.
Dependências Função `propagate_asm` (Tabela A.2).

A.4 Função `reconst_16views`

Tabela A.4: Documentação da função `reconst_16views`.

Nome da função

`reconst_16views`

Protótipo em Python

```
def reconst_16views(hologram)
```

Descrição

Reconstrói o holograma fornecido por argumento em 16 vistas.
Processo. Carrega o holograma (dois ficheiros *bitmap*: amplitude e fase) e o respetivo ficheiro de especificações (formato JSON). Calcula as posições das 16 vistas tendo em conta o tamanho do holograma.

Inputs

`hologram`: Nome do holograma a ser reconstruído.
Valor por defeito. "`dices4k`".

Output

Não aplicável.

Efeitos colaterais

Produz, dentro da pasta `./reconst/`, uma nova pasta com o nome do holograma. O respetivo conteúdo incluirá as 16 vistas (ficheiros `*.ppm`) correspondentes às reconstruções produzidas pela função.

Dependências

Função `load_hologram` (Tabela A.1);
Função `reconst_hologram` (Tabela A.3).
Ver Figura 4.2.

A.5 Função `compress_views`

Tabela A.5: Documentação da função `compress_views`.

Nome da função

`compress_views`

Protótipo em Python

```
def compress_views(hologram, ycbcr, rate)
```

Descrição

Comprimir os hologramas e calcular o PSNR.
Processo. Para cada vista, o holograma é codificado, descodificado e analisado em termos do PSNR. São criados 16 ficheiros `*.jp2` por pasta `rate_n`, correspondentes às 16 vistas reconstruídas, assim como ficheiros JSON (um por holograma) na pasta `kduOutput` com as métricas de compressão (PSNR).

Inputs

`hologram`: Nome do holograma a comprimir.
Valor por defeito. "`dices4k`".
`ycbcr`: *Flag* que indica ao kdu se deve ser efetuada uma transformada de cor.
Valor por defeito. `False`.
`rate`: número de *bits* por amostra.
Valor por defeito. `1.0`.

Output

Não aplicável.

Efeitos colaterais

Armazena as imagens no formato `*.jp2` nas pastas `rate_n`, assim como os ficheiros JSON na pasta `kduOutput`, segundo a seguinte árvore de diretórios:

```
./kduOutput
└── <nome_holograma>
    ├── no_ycbcr
    │   └── rate_n ..... n: número de bits
    └── ycbcr
```

Dependências

Função `cod_jpeg2000` (Tabela A.6);
Função `dec_jpeg2000` (Tabela A.7);
Função `psnr` (Tabela A.8).
Ver Figura 4.3.

A.6 Função `cod_jpeg2000`

Tabela A.6: Documentação da função `cod_jpeg2000`.

Nome da função
`cod_jpeg2000`
Protótipo em Python
`def cod_jpeg2000(in_path, out_path, ycbcr, rate)`
Descrição
Invoca ao sistema a execução do comando `kdu_compress`, conforme descrito na Secção 3.2.2.
Inputs
`in_path`: caminho do ficheiro de *input*. `out_path`: caminho do ficheiro de *output*. *Valor por defeito.* "`out.jp2`". `ycbcr`: *flag* que indica ao kdu se deve ser efetuada uma transformada de cor. *Valor por defeito.* `False`. `rate`: número de *bits* por amostra. *Valor por defeito.* `1.0`.
Output
Não aplicável.
Efeitos colaterais
Gera uma imagem no formato `*.jp2` no diretório definido por `out_path`.
Dependências
Não aplicável.

A.7 Função dec_jpeg2000

Tabela A.7: Documentação da função dec_jpeg2000.

Nome da função	dec_jpeg2000
Protótipo em Python	`def dec_jpeg2000(in_path, out_path, ycbcr, rate)`
Descrição	Invoca ao sistema a execução do comando kdu_expand, conforme descrito na Secção 3.2.2.
Inputs	in_path: caminho do ficheiro de *input*. out_path: caminho do ficheiro de *output*. *Valor por defeito.* "out.jp2". ycbcr: *flag* que indica ao kdu se deve ser efetuada uma transformada de cor. *Valor por defeito.* False. rate: número de *bits* por amostra. *Valor por defeito.* 1.0.
Output	Não aplicável.
Efeitos colaterais	Gera uma imagem no formato *.tmp no diretório definido por out_path.
Dependências	Não aplicável.

A.8 Função `psnr`

Tabela A.8: Documentação da função `psnr`.

Nome da função
`psnr`
Protótipo em Python
`def psnr(p1, p2)`
Descrição
Calcula a métrica de compressão PSNR entre duas imagens.
Inputs
`p1`: caminho para a primeira imagem. `p2`: caminho para a segunda imagem.
Output
Valor calculado do PSNR.
Efeitos colaterais
Não aplicável.
Dependências
Não aplicável.