

Universidade da Beira Interior

Departamento de Informática



**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**

29 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 <i>Red, Green & Blue</i> (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 Trans- crição para Python	8

3.2.2	<i>Kakadu Software</i>	8
3.3	Materiais Utilizados	9
3.3.1	<i>Hardware</i>	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	19
4.3.1	Breve Estudo do <i>Kakadu Software</i>	20
4.3.2	Implementação e Execução do <i>Script</i> de Compressão	20
4.3.3	Automatização dos <i>Scripts</i>	20
4.4	Determinação das Métricas de Compressão	23
4.5	Conclusões	23
5	Testes e Resultados	25
5.1	Introdução	25
5.2	Resultados	25
5.3	Conclusões	25
6	Conclusões e Trabalho Futuro	27
6.1	Conclusões Principais	27
6.2	Trabalho Futuro	27

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 obten- ção dos resultados.	15
4.2	Diagrama de dependências da função reconst_16views.	16
4.3	Diagrama de dependências da função compress_views.	20

Lista de Tabelas

4.1	Resumo da documentação da função <code>load_hologram</code>	16
4.2	Resumo da documentação da função <code>propagate_asm</code>	17
4.3	Resumo da documentação da função <code>reconst_hologram</code>	18
4.4	Resumo da documentação da função <code>reconst_16views</code>	19
4.5	Resumo da documentação da função <code>compress_views</code>	21
4.6	Resumo da documentação da função <code>cod_jpeg2000</code>	22
4.7	Resumo da documentação da função <code>dec_jpeg2000</code>	23
4.8	Resumo da documentação da função <code>psnr</code>	24

Acrónimos

ASM *Angular Spectrum Method*

JPEG *Joint Photographic Experts Group*

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 reconstruídos 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 onda. 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 princípio de holografia foi descoberto em 1948 pelo físico Dennis Gabor enquanto investigava microscopia de elétrons.

Ao contrário da fotografia convencional, que permite a captura da intensidade da luz, holografia permite guardar a amplitude e a fase da onda 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 cartesianas 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 holográficas 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 perspectivas 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]. Concluíram que são esperadas melhores taxas de compressão quando o holograma digital é codificado em componentes reais e imaginárias 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 família de wavelets Fresnelets. Fresnelets foram também aplicadas em 2003 por Livelin 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 concluíram que os melhores formatos de representação são phase-shifted 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 *software* 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 decodificação 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 transformação 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 reconstruídos 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 respectivas 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).

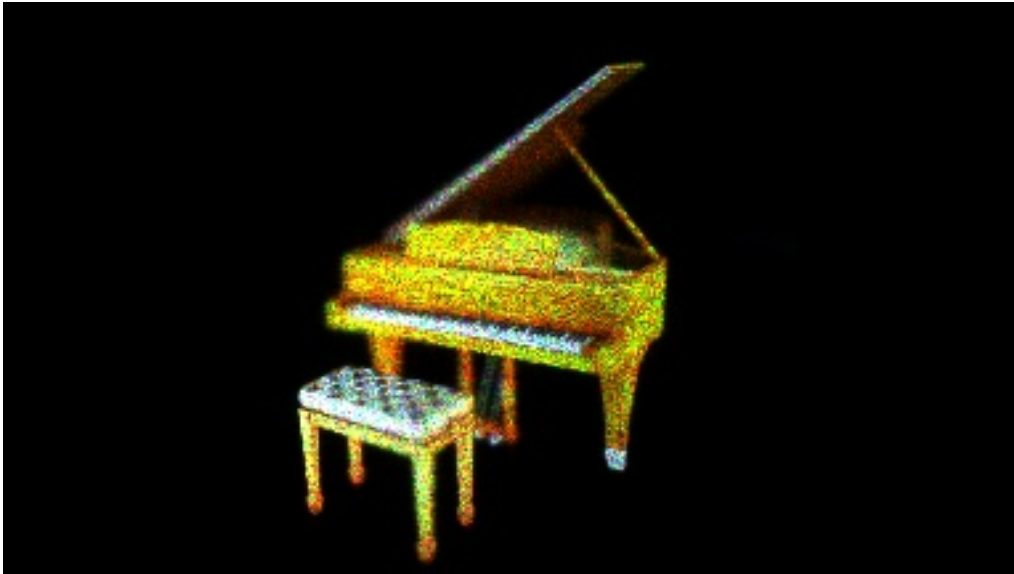


Figura 3.3: Holograma Piano4k (imagem original).

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.

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 final.

4.2 Reconstrução dos Hogramas

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 4.1, 4.2 e 4.3.

Estas funções foram compiladas num único *script* Python. A fim de cumprir o 2º objetivo secundário, uma nova função, `reconst_16views`, foi implementada (Tabela 4.4). 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 JSON cujo conteúdo inclui:

- `holo_size`: Resolução do holograma (em pixeis);
- `pitch`: Distância entre pixeis (em metros);
- `wavelengths`: 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:

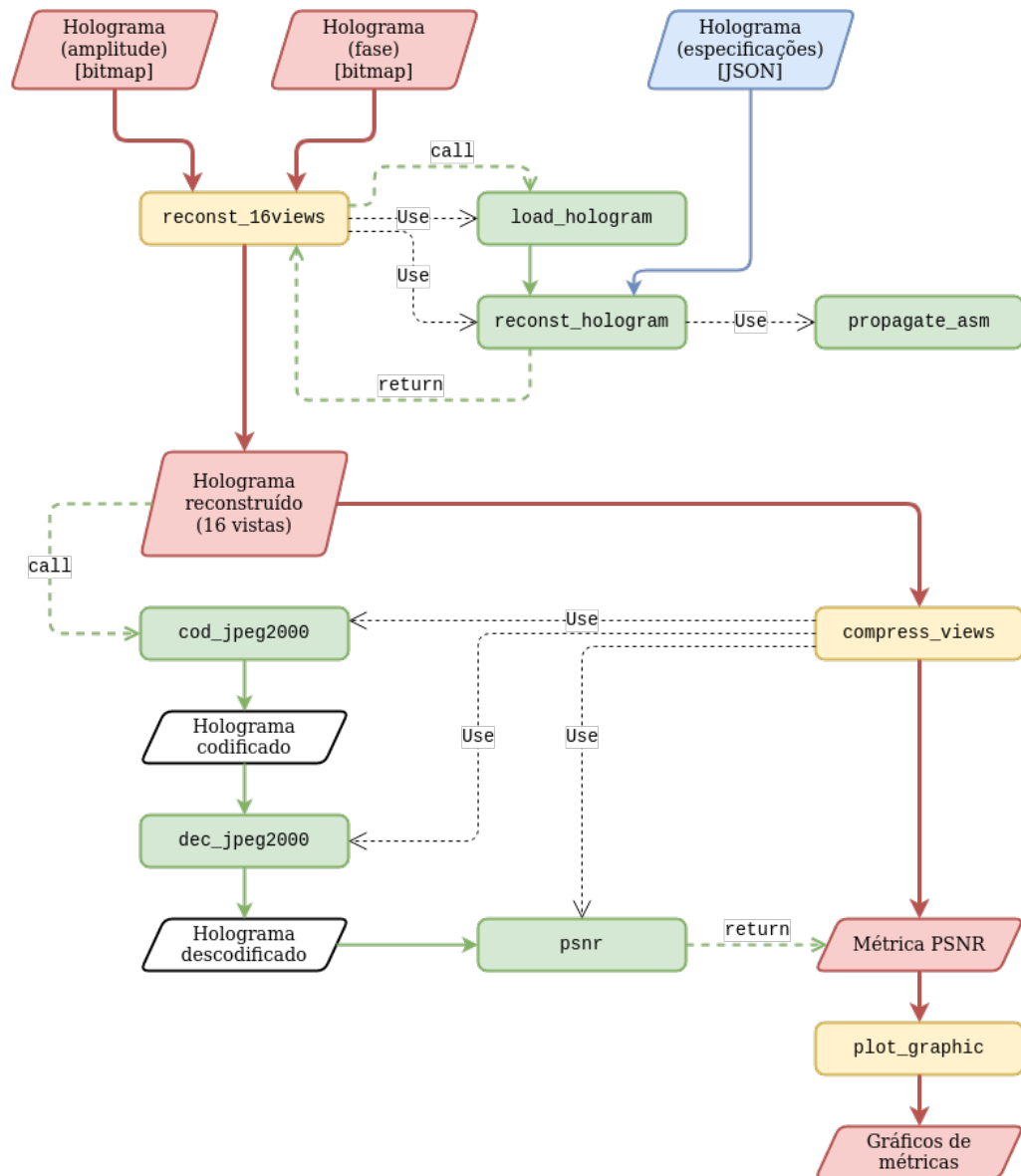


Figura 4.1: Diagrama do processo efetuado sobre um holograma para obtenção dos resultados.

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”.

Tabela 4.1: Resumo da documentação da função `load_hologram`.

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

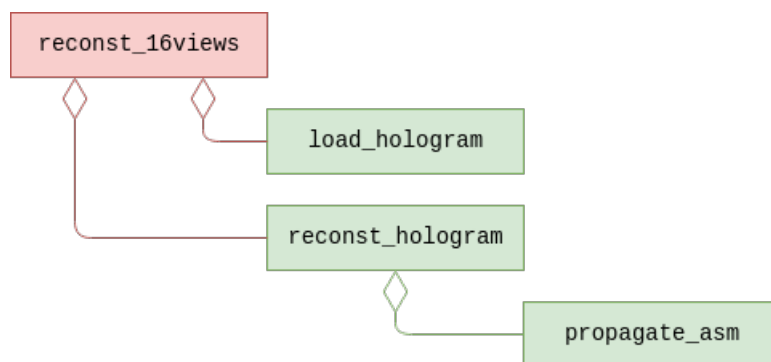
Figura 4.2: Diagrama de dependências da função `reconst_16views`.

Tabela 4.2: Resumo da documentação da função `propagate_asm`.

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

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;

Tabela 4.3: Resumo da documentação da função `reconst_hologram`.

Nome da função	<code>reconst_hologram</code>
Protótipo original em MATLAB	<code>function [recons] = reconsHologram(hologram, pitch, wavelengths, z, pupilPos, pupilSize)</code>
Protótipo transcrito em Python	<code>def reconst_hologram(hologram, pitch, wavelengths, z, pupil_pos, pupil_size)</code>
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	<p><code>hologram</code>: Holograma de modulação complexa (3 canais: RGB).</p> <p><code>pitch</code>: Distância entre pixels (em metros).</p> <p><code>wavelengths</code>: Comprimentos de onda de luz (em metros, 3 canais: RGB).</p> <p><code>z</code>: Distância de reconstrução (em metros).</p> <p><code>pupilPos</code>: Posição da janela (em pixels, canto superior direito).</p> <p><i>Valor por defeito.</i> <code>[0, 0]</code>.</p> <p><code>pupilSize</code>: Tamanho da janela (em pixels, altura \times largura).</p> <p><i>Valor por defeito.</i> <code>None</code>.</p>
Output	Reconstrução numérica do holograma (3 canais: RGB).
Efeitos colaterais	Não aplicável.
Dependências	Função <code>propagate_asm</code> (Tabela 4.2).

Tabela 4.4: Resumo da documentação da função `reconst_16views`.

Nome da função	<code>reconst_16views</code>
Protótipo em Python	<code>def reconst_16views(hologram)</code>
Descrição	<p>Reconstrói o holograma fornecido por argumento em 16 vistas.</p> <p><i>Processo.</i> Carrega o holograma (dois ficheiros <i>bitmap</i>: amplitude e fase) e o respetivo ficheiro de especificações (formato JSON). Calcula as posições das 16 vistas tendo em conta o tamanho do holograma.</p>
Inputs	<p><code>hologram</code>: Nome do holograma a ser reconstruído.</p> <p><i>Valor por defeito.</i> “dices4k”.</p>
Output	Não aplicável.
Efeitos colaterais	<p>Produz, dentro da pasta <code>./reconst/</code>, uma nova pasta com o nome do holograma. O respetivo conteúdo incluirá as 16 vistas (ficheiros <code>*.ppm</code>) correspondentes às reconstruções produzidas pela função.</p>
Dependências	<p>Função <code>load_hologram</code> (Tabela 4.1);</p> <p>Função <code>reconst_hologram</code> (Tabela 4.3).</p> <p><i>Ver Figura 4.2.</i></p>

- 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 Hogramas Reconstruídos

TODO

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

TODO

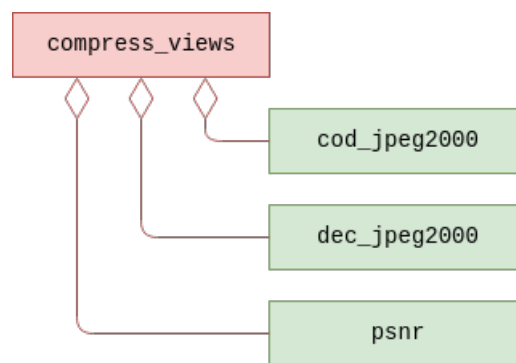


Figura 4.3: Diagrama de dependências da função `compress_views`.

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 afim de automatizar todo este processo. São eles os seguintes:

1. *Makefile*: Introduce 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 4.4);
 - 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*;

Tabela 4.5: Resumo da documentação da função `compress_views`.

Nome da função	<code>compress_views</code>
Protótipo em Python	<code>def compress_views(hologram, ycbcr, rate)</code>
Descrição	<p>Comprimir os hologramas e calcular o PSNR.</p> <p><i>Processo.</i> Para cada vista, o holograma é codificado, decodificado e analisado em termos do PSNR. São criados 16 ficheiros *.jp2 por pasta <code>rate_n</code>, correspondentes às 16 vistas reconstruídas, assim como ficheiros JSON (um por holograma) na pasta <code>kduOutput</code> com as métricas de compressão (PSNR).</p>
Inputs	<p><code>hologram</code>: Nome do holograma a comprimir. <i>Valor por defeito.</i> “dices4k”.</p> <p><code>ycbcr</code>: <i>Flag</i> que indica ao kdu se deve ser efetuada uma transformada de cor. <i>Valor por defeito.</i> False.</p> <p><code>rate</code>: número de <i>bits</i> por amostra. <i>Valor por defeito.</i> 1.0.</p>
Output	Não aplicável.
Efeitos colaterais	<p>Armazena as imagens no formato *.jp2 nas pastas <code>rate_n</code>, assim como os ficheiros JSON na pasta <code>kduOutput</code>, segundo a seguinte árvore de diretórios:</p> <pre> ./ ├── kduOutput │ ├── dices4k │ │ ├── no_ycbcr │ │ │ ├── rate_n n: número de bits │ │ │ └── ... │ │ └── ycbcr │ ├── diffuseCar4k idem │ └── piano4k </pre>
Dependências	<p>Função <code>cod_jpeg2000</code> (Tabela 4.6);</p> <p>Função <code>dec_jpeg2000</code> (Tabela 4.7);</p> <p>Função <code>psnr</code> (Tabela 4.8).</p> <p><i>Ver Figura 4.3.</i></p>

Tabela 4.6: Resumo da documentação da função `cod_jpeg2000`.

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

- f) `install`: instala pacotes no sistema operativo essenciais à execução dos *scripts*;
 - g) `clean`: remove os diretórios relativos a: hologramas reconstruídos, *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 4.4) e `compress_views` (Tabela 4.5).

Tabela 4.7: Resumo da documentação da função `dec_jpeg2000`.

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

4.4 Determinação das Métricas de Compressão

TODO

4.5 Conclusões

TODO

Tabela 4.8: Resumo da documentação da função `psnr`.

Nome da função
<code>psnr</code>
Protótipo em Python
<code>def psnr(p1, p2)</code>
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.

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 principais 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?

