UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CIÊNCIAS DA COMPUTAÇÃO

André Filipe da Silva Fernandes

Algoritmos de Controle de Taxa para Compressão de Light Fields no Padrão JPEG Pleno Parte 2 – 4DTM

Florianópolis
8 de julho de 2024

André Filipe da Silva Fernandes

Algoritmos de Controle de Taxa para Compressão de Light Fields no Padrão JPEG Pleno Parte 2 – 4DTM

Trabalho de Conclusão de Curso submetido ao Curso de Graduação em Ciências da Computação do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito para obtenção do título de Bacharel em Ciência da Computação.
Orientador: Prof. Ismael Seidel, Dr.

Florianópolis
8 de julho de 2024

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Fernandes, André Filipe da Silva
Algoritmos de controle de taxa para compressão de light fields no padrão JPEG Pleno Parte 2 – 4DTM / André Filipe da Silva Fernandes ; orientador, Ismael Seidel, 2024.
72 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2024.

Inclui referências.

1. Ciências da Computação. 2. Controle de taxa. 3. Light Fields. 4. JPEG Pleno. 5. Compressão de sinais visuais. I. Seidel, Ismael. II. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. III. Título.

AGRADECIMENTOS

Agradeço a minha família, por todo o apoio e suporte desde sempre; à Amanda, pela companhia e paciência nos meus dias mais insuportáveis; ao meu orientador Ismael pela ideia de tema, pela confiança e pelas várias das discussões mais confusas e produtivas que eu já tive; aos meus amigos, por me ajudar a não enlouquecer durante estes anos, especialmente ao Djalma, Gabriel, Hans, Josefo, JVM, Samuel e Yoshi; e por fim, a todos que de alguma forma contribuíram com a minha formação.

Obrigado.

“Sometimes I’ll start a sentence and I don’t even know where it’s going. I just hope I find it along the way.”
– Michael Scott

RESUMO

Um problema comum na compressão de sinais visuais com perdas é a necessidade de alcançar um balanço entre a quantidade de informação que pode ser descartada e o impacto disso na qualidade visual do dado. Um arquivo com taxa de bits muito alta pode ocupar uma grande quantidade de memória, enquanto taxas muito baixas resultam em imagens com uma qualidade visivelmente pior. O Modo de Transformada 4D – *4D Transform Mode* (4DTM), utilizado para codificar *Light Field*s (LFs) na Parte 2 do padrão JPEG Pleno, realiza esse balanço selecionando as opções que minimizam um custo, definido através da taxa de compressão, da distorção e de um multiplicador Lagrangiano (λ), que é definido pelo usuário. Esta é uma solução simples e efetiva, mas que dificulta a utilização por usuários que buscam por taxas específicas, como é o caso de quem deseja seguir as especificações contidas nas Condições Comuns de Teste – *Common Test Conditions*s (CTCs) do padrão. Este trabalho propõe dois algoritmos inéditos, criados para encontrar eficientemente um valor λ que corresponda a determinada taxa-alvo. Os algoritmos foram batizados de *Hyperbolic Split* (H. Split) e *Hyperbolic Slope* (H. Slope), e são capazes de atingir as taxas alvo testadas com erro máximo de 1%, sem perda de qualidade.

Palavras-chave: Light fields. JPEG Pleno. Compressão de sinais visuais. Controle de taxa.

ABSTRACT

A common concern in lossy visual signal data compression is balancing the amount of information that can be discarded and the related impact on the signal quality. High bitrates can result in huge files, while small bitrates can result in poor visual quality. The codification of light fields in the 4D Transform Mode, according to the JPEG Pleno Standard Part 2, solves this problem using a user defined Lagrangian multiplier (λ) to calculate a cost that is minimized during the codification. Although this is a simple and effective solution, it is hard for a user to encode a light field aiming for specific rates, as required by the standard CTC. This work proposes two new algorithms to efficiently find a λ parameter such that the rate of the resulting light field matches a user defined target. The algorithms were named Hyperbolic Split and Hyperbolic Slope, and they are able to achieve target rates with maximum error of 1%, without quality loss.

Keywords: Light fields. JPEG Pleno. Visual Signal Compression. Rate Control.

LISTA DE FIGURAS

Figura 1 – Exemplos de câmeras para captura de LFs. 23
Figura 2 – Imagem capturada por câmera plenóptica sem pós processamento. 24
Figura 3 – Padrões que compõem a família *Joint Photographic Experts Group* (JPEG) Pleno. 25
Figura 4 – Arquitetura genérica de decodificação de um LF no padrão JPEG Pleno . . . 26
Figura 5 – Processo de codificação de um LF através do modo de transformada 4D. . . 28
Figura 6 – Ilustração do processo de RDO através de um multiplicador Lagrangiano. . 29
Figura 7 – Vistas de exemplo de LFs capturados pela câmera *Lenslet Lytro Illum*. . . . 30
Figura 8 – Vista de exemplo do LF Set2 2K sub. 31
Figura 9 – Vista de exemplo do LF Poznan Laboratory 1. 32
Figura 10 – Vista de exemplo do LF Tarot Cards. 33
Figura 11 – Vistas de exemplo de LFs sintéticos. 33
Figura 12 – Exemplo ilustrativo do algoritmo de bisseção aplicado ao problema de controle de taxa. 39
Figura 13 – Distribuições de taxa e lambda comparadas com o modelo hiperbólico. . . . 40
Figura 14 – Exemplo ilustrativo do algoritmo de H. Split aplicado ao problema de controle de taxa. 42
Figura 15 – Distribuições de taxa e distorção comparadas com o modelo hiperbólico. . . 44
Figura 16 – Exemplo ilustrativo do algoritmo H. Slope aplicado ao problema de controle de taxa. 48
Figura 17 – Número de iterações necessárias para o LF “Bikes” 51
Figura 18 – Número de iterações necessárias para o controle de taxa dos LFs testados. . 52
Figura 19 – Tempo de execução (s) para cada um dos algoritmos testados. 53
Figura 20 – Speedup médio de cada algoritmo em relação à bisseção. 54
Figura 21 – *Overhead* médio de cada algoritmo em relação a um oráculo. 56

LISTA DE TABELAS

Tabela 1 – Configurações do Dataset. 32
Tabela 2 – Taxas alvo definidas nas CTCs (Pereira et al., 2019). 34
Tabela 3 – Argumentos criados na CLI do JPLM e suas respectivas funções. 49
Tabela 4 – LFs e taxas alvo escolhidas para os testes. 50

LISTA DE ALGORITMOS

Algoritmo 1 – Bisseção 38
Algoritmo 2 – Hyperbolic Split 43
Algoritmo 3 – Hyperbolic Slope 47

LISTA DE ABREVIATURAS E SIGLAS

4D-DCT	Transformada Discreta de Cossenos em 4-Dimensional – *4 dimensional discrete cosine transform* 21, 28
4DPM	Modo de Predição 4D – *4D Prediction Mode* 21, 25, 26, 50
4DTM	Modo de Transformada 4D – *4D Transform Mode* . . 7, 21, 22, 25–28, 31, 36, 37, 40, 49, 50, 57
ABAC	Codificador Aritmético Binário Adaptável – *Adaptive Binary Arithmetic Coding* 29
bpp	bits por pixel – *bits per pixel* 22, 32, 34, 49, 51, 52
BU	Unidade Básica – *Basic Unit* 35
CE	Core Experiment 54, 57
CLI	Interface de Linha de Comando – *Command Line Interface* 13, 49
CTC	Condições Comuns de Teste – *Common Test Conditions* . 7, 9, 13, 21, 29, 31, 34, 50, 57
DCT	Transformada Discreta de Cossenos – *Discrete Cosine Transform* 27, 28
H. Slope	*Hyperbolic Slope* 7, 11, 44, 46, 48, 49, 52, 54, 56, 57
H. Split	*Hyperbolic Split* 7, 11, 42, 45, 46, 49, 51, 52, 54, 56, 57
HDCA	Conjunto de Câmeras de Alta Densidade – *High Density Camera Array* . 30, 31, 50
HEVC	Codificador de Vídeo de Alta Eficiência – *High Efficiency Video Coding* . 26, 28, 29, 35, 40
HVS	Sistema Visual Humano – *Human Visual System* 34
JPEG	*Joint Photographic Experts Group* 11, 21, 22, 24–26, 29, 34, 35, 54, 57
JPL	Formato de Arquivo JPEG Pleno – *JPEG Pleno file format* 29
JPLM	Modelo JPEG Pleno – *JPEG Pleno Model* 13, 22, 36, 40, 49, 57
LF	*Light Field* 7, 11, 13, 21–33, 36–40, 43–45, 47, 50–52, 54–57
MSE	Erro Quadrático Médio – *Mean Squared Error* 31, 32

PCRD-Opt Otimização de Taxa-Distorção Pós-Compressão – *Post-Compression Rate-Distortion Optimization* 34

PGM *Portable Gray Map* 30

PNG *Portable Network Graphics* 31

PPM *Portable Pix Map* 30

PSNR Relação Sinal-Ruído de Pico – *Peak Signal-to-Noise Ratio* 32, 34

QP Parâmetro de Quantização – *Quantization Parameter* 35, 36

RD Taxa-Distorção – *Rate-Distortion* 28, 29, 35, 36, 44, 45, 47, 52, 57

RDO Otimização de Taxa-Distorção – *Rate-Distortion Optimization* 11, 28, 29, 35, 36, 49

VVC Codificador Versátil de Vídeo – *Versatile Video Coding* 28, 35

SUMÁRIO

1 INTRODUÇÃO 21
1.1 OBJETIVOS 22
1.1.1 Objetivo Geral 22
1.1.2 Objetivos Específicos 22
1.2 ORGANIZAÇÃO 22

2 CONTEXTUALIZAÇÃO E TRABALHOS CORRELATOS 23
2.1 LIGHT FIELDS 23
2.2 JPEG PLENO 24
2.2.1 Compressão de light fields 25
2.2.1.1 4D Prediction Mode (4DPM) 26
2.2.1.2 4D Transform Mode (4DTM) 27
2.3 COMMON TEST CONDITIONS (CTC) 29
2.3.1 Dataset 29
2.3.2 Métricas 31
2.4 TRABALHOS CORRELATOS 34

3 PROPOSTAS DE ALGORITMOS 37
3.1 BISSEÇÃO 37
3.2 HYPERBOLIC SPLIT 40
3.3 HYPERBOLIC SLOPE ALGORITHM 44

4 RESULTADOS 49
4.1 CONFIGURAÇÃO EXPERIMENTAL 49
4.2 NÚMERO DE ITERAÇÕES 51
4.3 TEMPO DE EXECUÇÃO 52

5 CONCLUSÕES 57
5.1 TRABALHOS FUTUROS 57

REFERÊNCIAS 59

ANEXO A – CÓDIGO FONTE 63

ANEXO B – ARTIGO 65

1 INTRODUÇÃO

*Light Field*s (LFs) são uma tecnologia de representação de imagens, capaz de retratar simultaneamente a intensidade e o ângulo de raios de luz incidentes em cada ponto de um plano, superando, assim, limitantes de tecnologias mais tradicionais como fotografias e vídeos. Deste modo, LFs proporcionam algumas funcionalidades novas em relação às demais abordagens, como a capacidade de simular o efeito paralaxe[1] através de tipos especiais de telas, permitindo uma experiência mais natural e imersiva do que no cinema 3D tradicional. Também é possível extrair mapas de profundidade de LFs com certa facilidade, realizar pequenas alterações de perspectiva e até alterar o foco da imagem em etapas de pós processamento (Conceição, 2017).

Por definição, um LF é composto por um conjunto de pixels dispostos em 4 dimensões (t,s,v,u) (Alves et al., 2020), respectivamente as coordenadas verticais, horizontais, ângulo de incidência perpendicular ao eixo vertical e ângulo de incidência perpendicular ao eixo horizontal. Isso implica que uma imagem de LF com alguma resolução razoável ocupará uma quantidade impraticavelmente grande de memória para armazenamento. O LF "Set2 2K sub", por exemplo, que faz parte das Condições Comuns de Teste – *Common Test Conditions* (CTC) (Pereira et al., 2019), possui uma resolução de 1920×1080 pixels, 33×11 vistas e ocupa 2,62 GiB em memória. A versão completa do LF "Set 2" possui 3840×2160 pixels e 101×21 vistas, ocupando 61,44 GiB.

Para contornar este problema, técnicas de compressão de dados podem ser aplicadas, assim como acontece em outras modalidades de imagens. Neste contexto, o *Joint Photographic Experts Group* (JPEG) criou o grupo JPEG Pleno, a fim de criar e padronizar um *framework* para codificação de LFs e outras imagens plenópticas, como *point clouds* e hologramas. Até o momento, a Parte 2 do padrão JPEG Pleno possui dois modos de codificação de LFs:

1. Modo de Transformada 4D – *4D Transform Mode* (4DTM), que se baseia na Transformada Discreta de Cossenos em 4-Dimensional – *4 dimensional discrete cosine transform* (4D-DCT); e

2. Modo de Predição 4D – *4D Prediction Mode* (4DPM) que é um modelo preditivo, capaz de gerar partes da imagem baseando-se em seu contexto.

Este trabalho se concentra em resolver o problema do controle de taxa apenas para o 4DTM. O problema do controle de taxa, neste contexto, reside na necessidade de que os dados codificados possuam um tamanho próximo a uma taxa previamente especificada, chamada de taxa-alvo. Ter este controle do tamanho dos dados após a codificação é uma característica útil para *codecs*, especialmente se utilizados em aplicações de *streaming*, por exemplo, onde é necessário controlar precisamente a vazão dos dados de acordo com a capacidade da rede.

[1] Efeito em que a movimentação aparente de um objeto varia de acordo com sua distância.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

O objetivo geral deste trabalho é propor um método capaz de controlar a otimização de taxa-distorção de um codificador de LFs compatível com a norma ISO/IEC 21794-2 (JPEG Pleno Parte 2) no modo 4DTM, e implementá-lo em seu software de referência, chamado de Modelo JPEG Pleno – *JPEG Pleno Model* (JPLM), tal que seja possível atingir uma taxa-alvo, medida em bits por pixel – *bits per pixel* (bpp), com pouca ou nenhuma redução de qualidade objetiva.

1.1.2 Objetivos Específicos

- Demonstrar que os algoritmos propostos atingem a taxa-alvo conforme o esperado.
- Manter a eficiência de codificação dos algoritmos propostos próxima à do algoritmo original.
- Avaliar o desempenho, em termos de tempo de codificação, dos novos algoritmos em relação ao software original.

1.2 ORGANIZAÇÃO

Este trabalho está organizado como segue. No Capítulo 2 será feita uma contextualização, para que o leitor entenda o escopo deste trabalho, além de uma revisão de algumas técnicas de controle de taxa empregadas em codificadores de imagens e vídeo. No Capítulo 3 o problema é discutido com mais profundidade, e as soluções propostas são apresentadas com detalhes. Finalmente, no Capítulo 4, os resultados deste trabalho são avaliados, e as conclusões são apresentadas no Capítulo 5.

2 CONTEXTUALIZAÇÃO E TRABALHOS CORRELATOS

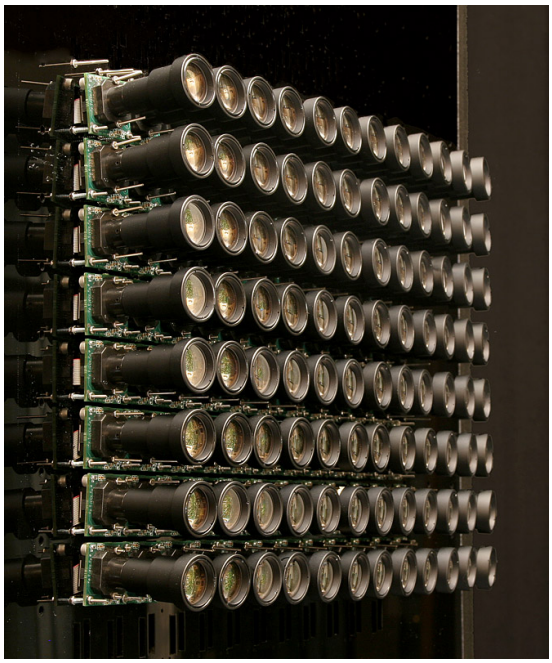
2.1 LIGHT FIELDS

A função plenóptica é um modelo idealizado de se representar informações visuais que define a luz como uma função de 7 dimensões (Adelson; Bergen, 1991). Uma função plenóptica $P(\vartheta, \varphi, Vx, Vy, Vz, \lambda, t)$ representa a intensidade dos raios de luz que incidem em uma posição do espaço 3D (Vx, Vy, Vz), a partir de um ângulo (ϑ, φ), com um comprimento de onda (λ) em um determinado instante de tempo (t). Este modelo fornece uma descrição muito precisa da luz, e poderia representar a realidade de maneira muito fiel. Porém, a aquisição destes dados não é praticável a partir de tecnologias atuais.

Um LF pode ser compreendido como uma versão simplificada da função plenóptica, de maneira a viabilizar sua aquisição, e é descrito como uma função quadridimensional $L(\vartheta, \varphi, Vx, Vy)$. Esta simplificação é possível limitando a imagem a cenas estáticas, que incidem em um plano (em vez de todo o espaço tridimensional) e que possui apenas 3 canais de cores. A partir desta definição é possível adquirir LFs de algumas formas diferentes. As mais comuns são através de matrizes de câmeras (Wilburn et al., 2005a) e câmeras plenópticas (Ng et al., 2005), mostradas na Figura 1.

Figura 1 – Exemplos de câmeras para captura de LFs.

(a) Matriz de câmeras de Stanford.



(b) Camera Raytrix.

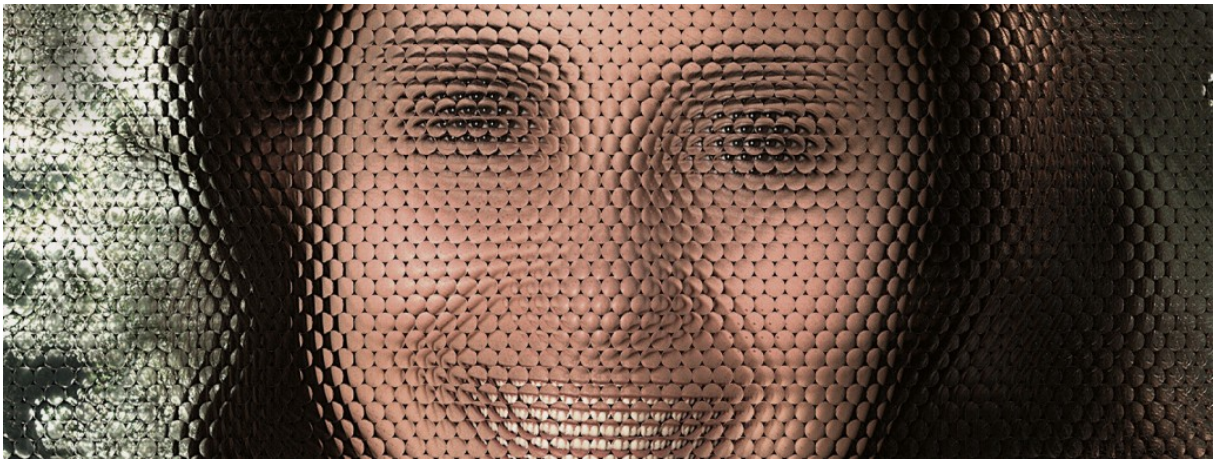
Fonte: (a) Wilburn et al. (2005b); (b) Raytrix (2019).

A matriz de câmeras é uma das formas mais simples de se obter um LF, pois basta capturar, simultaneamente, diversas fotografias utilizando um conjunto de câmeras dispostas em uma matriz. Este conjunto de imagens é equivalente a um LF, pois cada câmera é capaz de representar uma vista diferente (u, v) e cada fotografia codifica as coordenadas espaciais (t, s)

de cada vista. Neste contexto, uma vista é o equivalente a uma fotografia tradicional. Esta modalidade, apesar da simplicidade, possui problemas como: a necessidade de sincronizar as câmeras; e o grande volume ocupado pelo dispositivo, como é possível observar na Figura 1a.

Por sua vez, as câmeras plenópticas são muito mais simples de operar e possuem tamanhos equivalentes aos de câmeras fotográficas comuns. Um exemplo de câmera plenóptica de uso industrial, fabricada pela empresa Raytrix, é apresentado na Figura 1b. Para capturar as 4 dimensões necessárias de um LF, estas câmeras possuem uma matriz de micro lentes posicionada entre a lente principal e seu sensor (Perwaß; Wietzke, 2012). Cada micro lente, de acordo com sua posição, causa distorções na imagem gerada pela lente principal, formando pequenas micro imagens como demonstra a Figura 2. Esta representação é menos intuitiva, mas como seus resultados são equivalentes, é possível convertê-la em um formato análogo ao gerado por matrizes de câmeras.

Figura 2 – Imagem capturada por câmera plenóptica sem pós processamento.



Fonte: Raytrix (2015a)

A praticidade das câmeras plenópticas possibilita a utilização de LFs em diversos cenários (Raytrix, 2015b), como em pesquisas científicas onde são necessárias formas de medir movimentos tridimensionais de animais ou plantas, mas não é viável a utilização de sensores físicos como acelerômetros. Também é possível adaptar câmeras plenópticas em microscópios, possibilitando a captura de informações mais ricas, utilizando o mesmo equipamento (Levoy et al., 2006). Além disso, a capacidade de extrair mapas de profundidade pode ser útil em imagens médicas, reconhecimento facial, e até em inspeções automatizadas na indústria.

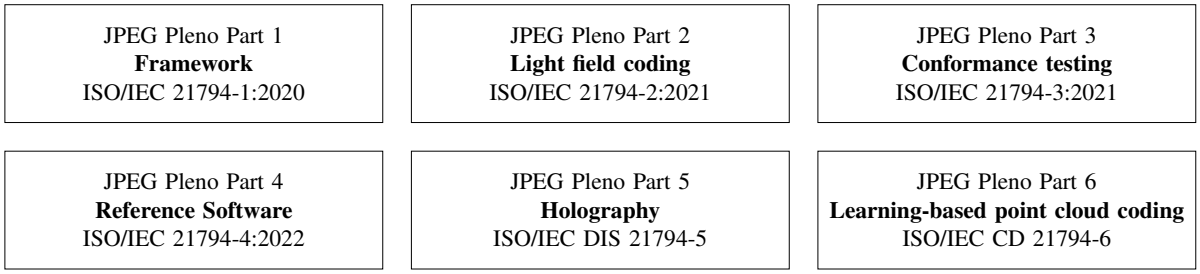
2.2 JPEG PLENO

Dada a capacidade representativa da função plenóptica, o grupo JPEG iniciou o projeto JPEG Pleno, para a criação de um *framework* capaz de representar, comprimir e manipular mídias baseadas nesta função (Astola et al., 2020). Um dos requisitos do projeto é a possibilidade de representação de múltiplas modalidades de dados em um único arquivo, de modo que

os dados se complementem e sejam efetivamente mais representativos em conjunto (Ebrahimi et al., 2016; Astola et al., 2020).

As modalidades previstas até o momento, além de LFs, são: *Point Clouds* (nuvens de pontos) e Hologramas. Os padrões que constituem o projeto JPEG Pleno são apresentados na Figura 3. No momento, apenas as Partes 1 até 4 estão finalizadas (ISO Central Secretary, 2020; ISO Central Secretary, 2021a; ISO Central Secretary, 2021b; ISO Central Secretary, 2022), enquanto as demais permanecem em processo de padronização (ISO Central Secretary, 2023a; ISO Central Secretary, 2023b).

Figura 3 – Padrões que compõem a família JPEG Pleno.



Fonte: Adaptado de Astola et al. (2020).

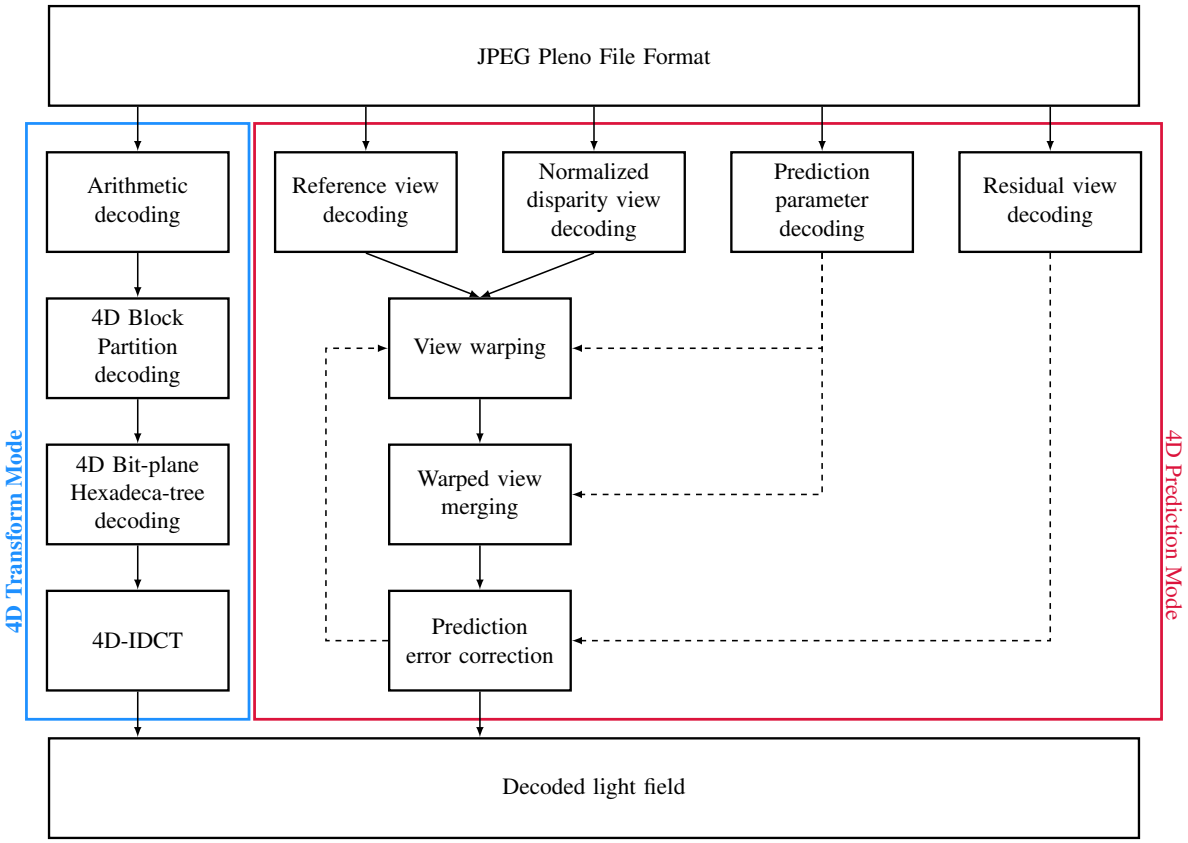
Point Clouds são uma forma de representar informações tridimensionais em que os dados são guardados como conjuntos de pontos no espaço que podem ou não ter alguma informação atrelada a eles (Perry, 2019), como cores, refletância, temperatura, ou simplesmente referências para bases de dados externas. *Point Clouds* possuem inúmeras aplicações práticas como em computação gráfica, impressões 3D, imagens médicas, análises urbanas, etc.

Hologramas são uma forma de armazenar frentes de onda de luz através da geração de padrões de interferometria (Oh, 2019). Estes padrões são igualmente capazes de codificar imagens tridimensionais. Dentre as aplicações da holografia digital, é possível citar a microscopia holográfica, tomografias holográficas, visualização tridimensional em impressões e telas e também pode ser muito útil no campo da metrologia, que utiliza amplamente padrões de interferometria.

2.2.1 Compressão de light fields

Devido à forma como os dados se organizam nos LFs, baseando-se em pixels, técnicas de compressão utilizadas em outros meios puderam ser adaptadas, e a modalidade evoluiu mais rapidamente que as demais (Alves et al., 2020). As duas formas que mais se destacaram e foram aceitas no padrão JPEG Pleno foram o modo preditivo, chamado de 4DPM, e o modo de transformadas 4D, chamado de 4DTM (Astola et al., 2020; ISO Central Secretary, 2021a). A Figura 4 ilustra como acontece a decodificação de um LF através de ambos os modos.

Figura 4 – Arquitetura genérica de decodificação de um LF no padrão JPEG Pleno. A seção em azul corresponde ao 4DTM, e a seção em vermelho ao 4DPM.



Fonte: Alves et al. (2020).

2.2.1.1 4D Prediction Mode (4DPM)

O codificador preditivo utiliza mapas de profundidade e parâmetros de câmeras para representar e codificar os LFs de maneira eficiente. Este modo utiliza algumas vistas de referência, de modo que seja possível estimar as demais vistas de maneira coerente. Desta forma, basta codificar as vistas de referência como imagens 2D tradicionais, juntamente com o resíduo das predições (pequenas correções às vistas geradas).

O primeiro passo do codificador preditivo é hierarquizar as vistas em múltiplos níveis, onde os níveis mais baixos servem como referência para a geração dos níveis superiores (Astola; Tabus, 2018). Esta configuração é feita de forma semelhante à configuração hierárquica de quadros de referência em codificadores preditivos de vídeo como o Codificador de Vídeo de Alta Eficiência – *High Efficiency Video Coding* (HEVC) (Astola et al., 2020).

As vistas do primeiro nível, assim como seus mapas de profundidade, são codificadas através de um codificador de imagens 2D. Normalmente, utiliza-se o formato JPEG 2000 (ISO Central Secretary, 2019) para este fim, apesar do codificador ser flexível o suficiente para permitir a utilização de outros formatos.

As vistas que pertencem aos demais níveis são codificadas através de uma estratégia diferente, realizada em três etapas: *warping*, *merging* e *sparse prediction*. A etapa de *warping*

utiliza cada uma das M vistas do nível anterior, juntamente com seus mapas de profundidade e distância entre câmeras, para gerar M estimativas de cada vista subsequente. Naturalmente, cada estimativa é incompleta, pois de acordo com a cena, é provável que ocorra a oclusão de algum objeto, dependendo do ângulo.

A etapa de *merging* utiliza as múltiplas estimativas de cada vista para fazer uma fusão com as melhores regiões de cada uma delas, reduzindo as imperfeições da estimativa. Por último, a vista gerada passa por uma etapa de convolução em 2D com um filtro escolhido adaptativamente a fim de melhorar o resultado em termos de distorção. Finalizadas estas etapas, é feita uma subtração entre as vistas reais e suas respectivas estimativas, e o resíduo resultante é codificado, caso necessário.

2.2.1.2 4D Transform Mode (4DTM)

A Transformada Discreta de Cossenos – *Discrete Cosine Transform* (DCT), é uma forma de representar um sinal finito como uma combinação de cossenos. O cálculo da DCT pode ser feito através da Equação 2.1, onde $G_x(k)$ é o valor da transformada na posição k para o sinal X, $X(m)$ é o sinal que está sendo transformado na posição m e M é o número de amostras deste sinal. Esta transformada é muito útil na codificação de imagens naturais, pois a maioria dos dados podem ser representados nos primeiros coeficientes do resultado, os quais representam baixas frequências, permitindo o descarte dos demais com impacto reduzido na qualidade da informação (Ahmed; Natarajan; Rao, 1974).

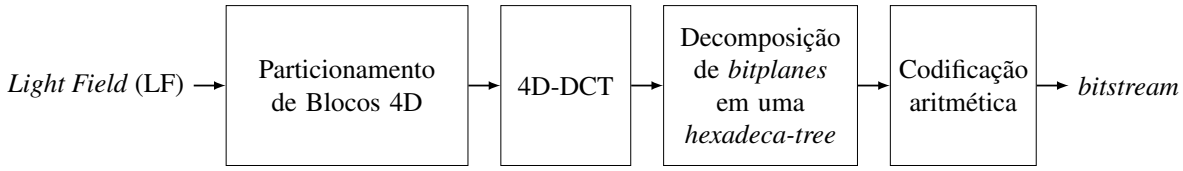
$$G_x(0) = \frac{\sqrt{2}}{M} \sum_{m=0}^{M-1} X(m)$$
$$G_x(k) = \frac{2}{M} \sum_{m=0}^{M-1} X(m) cos \frac{(2m+1)k\pi}{2M} \tag{2.1}$$

Para que a transformada fosse útil na codificação de imagens 2D foi necessário estender sua definição para duas dimensões, dado que cada pixel da imagem possui relação tanto com sua vizinhança horizontal quanto vertical. Para isso, basta executar o algoritmo unidimensional para cada linha da imagem, e em seguida, para cada coluna (ou vice-versa). O 4DTM se baseia no mesmo princípio de separabilidade, porém como cada pixel de um LF possui uma relação direta com seus vizinhos em 4 dimensões, é necessária a utilização de uma DCT quadridimensional, que pode ser definida de maneira semelhante ao caso 2D.

O processo de codificação do 4DTM é realizado em 4 etapas principais, como representado na Figura 5. A primeira etapa subdivide todo o LF em blocos 4D de mesmo tamanho. Durante todas as demais etapas da codificação, estes blocos são processados de forma totalmente independente entre si. Assim, a codificação de um deles não afeta os demais. Aplicar a DCT diretamente nestes blocos seria uma estratégia possível, mas, para melhorar a eficiência de codificação, o módulo de particionamento de blocos os subdivide recursivamente, assim como

acontece em padrões como o HEVC, o Codificador Versátil de Vídeo – *Versatile Video Coding* (VVC) e outros codificadores de vídeo.

Figura 5 – Processo de codificação de um LF através do modo de transformada 4D.



Fonte: o autor.

No 4DTM é possível dividir cada sub-bloco de duas formas:

- Através das dimensões espaciais (u, v);
- Através das dimensões de vistas (t, s).

Os sub-blocos podem ser recursivamente subdivididos, se necessário, até que alcancem um tamanho mínimo previamente configurado (Alves et al., 2020). Estas divisões são representadas por um conjunto de símbolos que serão, em seguida, escritos no arquivo comprimido (*bitstream*).

Estes blocos serão transformados através de uma 4D-DCT, e em seguida, representados através de uma *hexadeca-tree*, ou seja, uma árvore cujos nodos possuem até 16 nodos filhos. Esta representação explora o fato de que blocos transformados pela DCT costumam ser bastante esparsos e alguns poucos coeficientes são consideravelmente maiores do que os demais. Para explorar estas características, conjuntos diferentes de símbolos representam os blocos como *bitplanes* utilizando alguns destes símbolos para representar grandes áreas vazias (Alves et al., 2020). Um *bitplane*, nesse caso, representa o conjunto de bits no mesmo nível dentre as amostras do bloco. Por exemplo, é possível obter um *bitplane* extraindo todos os bits do nível menos significativo de cada uma das amostras de um bloco.

É nesta etapa que ocorre o descarte de informação, ou quantização. Isto acontece porque as decisões de particionamento de blocos e flags que serão utilizadas dependem de um custo Lagrangiano (J). Desta forma, a árvore resultante pode não representar perfeitamente o LF original se isso for vantajoso em termos de Taxa-Distorção – *Rate-Distortion* (RD).

O custo Lagrangiano (J) é calculado dinamicamente através da Equação 2.2, onde D é a distorção em relação ao bloco original, R é a taxa e λ é o multiplicador Lagrangiano, que pode ser configurado e determina a relação entre compressão e distorção desejada. Quando o codificador precisa tomar uma decisão, basta calcular os custos de cada opção e optar pelo menor. Este processo é chamado de Otimização de Taxa-Distorção – *Rate-Distortion Optimization* (RDO) (Ortega; Ramchandran, 1998; Sullivan; Wiegand, 1998).

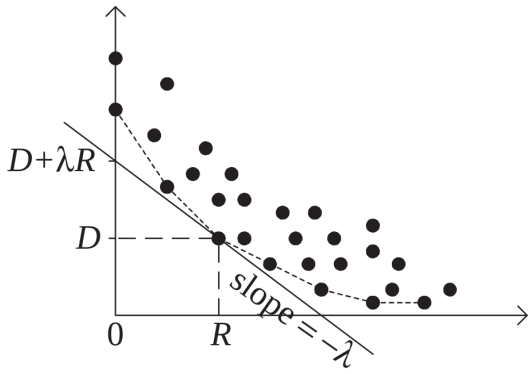
$$J = D + \lambda R \tag{2.2}$$

As *flags* de decisão que compõem a *hexadeca-tree* são armazenadas em *bitstreams* independentes. Estes *bitstreams* são finalmente codificados através de um Codificador Aritmético Binário Adaptável – *Adaptive Binary Arithmetic Coding* (ABAC), e os dados são escritos em um arquivo Formato de Arquivo JPEG Pleno – *JPEG Pleno file format* (JPL).

Apesar da independência na codificação dos blocos 4D, o mesmo valor de λ pode ser utilizado na codificação de todos eles. A adoção de um único valor de λ para todos os blocos garante a alocação ótima de bits que minimiza a distorção (Ortega; Ramchandran, 1998).

Outra informação importante a ser destacada, e que será útil no decorrer deste texto, é que o λ representa o inverso da inclinação de uma reta tangente à curva de RD (Ortega; Ramchandran, 1998). Isso também implica que o custo J representa o ponto em que essa reta encontra-se com o ponto $R = 0$. Estas informações podem ser visualizadas na Figura 6.

Figura 6 – Ilustração do processo de RDO através de um multiplicador Lagrangiano. O eixo X representa a taxa (R), enquanto o eixo Y representa a distorção (D). Cada círculo representa uma possível codificação. A linha pontilhada apresenta a fronteira de Pareto, com decisões de codificação ótimas em termos de RD, obtidas através da RDO com valores diferentes de λ.



Fonte: (Chou; Miao, 2001)

2.3 COMMON TEST CONDITIONS (CTC)

Para fornecer um conjunto de dados que sirva de base para pesquisadores interessados no desenvolvimento de técnicas de codificação de LFs, bem como padronizar testes e experimentos, o grupo JPEG Pleno criou as Condições Comuns de Teste – *Common Test Conditions* (CTC) para codificação de LFs (Pereira et al., 2019). As CTC contêm, além dos LFs, definições de métricas de qualidade, convenções de nomenclatura e até um método base de codificação, que interpreta o LF como uma sequência de imagens 2D e as comprime através de um codificador de vídeo conforme o padrão HEVC.

2.3.1 Dataset

Os LFs presentes no conjunto de testes são diversificados em termos de aquisição, geometria da cena, resolução, número de vistas, profundidade de cor e textura. Em relação

à aquisição, o *dataset* inclui LFs capturados pela câmera plenóptica *Lenslet Lytro Illum*, que foram chamados de *lenslets*, LFs capturados por Conjunto de Câmeras de Alta Densidade – *High Density Camera Array* (HDCA) e LFs sintéticos, gerados digitalmente. Todos os LFs, independente do formato em que foram publicados originalmente, são disponibilizados como um conjunto de vistas no formato *Portable Pix Map* (PPM) e possuem mapas de profundidade no formato *Portable Gray Map* (PGM).

As imagens capturadas pela câmera *Lenslet Lytro Illum* possuem resolução espacial de 625×434 pixels, 15×15 vistas (onde apenas 13×13 são de fato utilizadas devido às aberrações provocadas na periferia das microlentes), profundidade de cor de 10 bits por canal, 3 canais de cores e possuem um único mapa de profundidade para a vista central. Os LFs disponíveis são "Bikes", "Danger de Mort", "Stone Pillars Outside" e "Fountain&Vincent 2". Um exemplo de vista de cada LF *lenslet* está disponível na Figura 7.

Figura 7 – Vistas de exemplo de LFs capturados pela câmera *Lenslet Lytro Illum*.

(a) Bikes

(b) Fountain&Vincent 2

(c) Stone Pillars Outside

(d) Danger de Mort

Fonte: (Pereira et al., 2019)

Já a captura por HDCA foi feita através de diferentes configurações. O LF "Set2 2K" foi capturado pelo Fraunhofer HDCA e foi gerado utilizando uma câmera de alta qualidade com movimentos robotizados, que se moveu em uma grade com espaçamento de 4 mm verticalmente e 6 mm horizontalmente. A versão completa deste LF possui resolução espacial de 3840×2160 pixels, 101×21 vistas, profundidade de cor de 10 bits por canal e 3 canais de cores. Porém, por questões de complexidade, foi criada uma versão subamostrada deste LF, com resolução

espacial de 1920×1080 pixels, 33×11 vistas, chamada de “Set2 2K sub”. A Figura 8 mostra uma vista de exemplo do LF“Set2 2K sub”.

Figura 8 – Vista de exemplo do LF Set2 2K sub.

Fonte: (Pereira et al., 2019)

Já o LF “Poznan Laboratory 1” foi obtido através do Poznan HDCA, utilizando uma matriz de câmeras com espaçamento horizontal e vertical de 10 mm. O LF possui resolução espacial de 1936×1288 com 31×31 vistas e sua profundidade de cor é de 8 bits por canal e possui 3 canais de cores. A Figura 9 mostra uma vista de exemplo do LF “Poznan Laboratory 1”.

Finalizando a categoria, o LF “Tarot Cards” foi capturado pelo Stanford HDCA. Os dados estão disponíveis como uma sequência de imagens no formato *Portable Network Graphics* (PNG) com resolução de 1024×1024 representando 17×17 vistas, com profundidade de cor de 8 bits por canal e possui 3 canais de cores. Uma vista de exemplo está disponível na Figura 10.

Além destes exemplos, que incluem imagens capturadas no mundo real, os LFs “Greek” e “Sideboard” são sintéticos, gerados no *Heidelberg Collaboratory*. Ambos os LFs possuem resolução espacial de 512×512, 9×9 vistas, profundidade de cor de 8 bits por canal e 3 canais de cores. Vistas de exemplo de ambos os LFs estão disponíveis na Figura 11.

Os tamanhos dos LFs presentes nas CTCs, juntamente com os tamanhos máximos e mínimos de transformadas, usados para codificação com o 4DTM, estão compilados na Tabela 1.

2.3.2 Métricas

Para padronizar as análises de LFs as CTC definem algumas métricas. As mais frequentemente utilizadas no decorrer deste trabalho são Erro Quadrático Médio – *Mean Squared*

Figura 9 – Vista de exemplo do LF Poznan Laboratory 1.

Fonte: (Pereira et al., 2019)

Tabela 1 – Configurações do Dataset.

Light Field	**Tamanho do Light field (t×s×v×u)**	**Tamanho de Transformada (t×s×v×u)**	
		Mínimo	**Máximo**
Todos os Lenslets	13×13×625×434	13×13×25×31	13×13×4×4
Greek	9×9×512×512	9×9×32×32	9×9×4×4
Tarot cards	17×17×1024×1024	17×17×32×32	17×17×4×4

Fonte: (Seidel; Fernandes; Güntzel, 2024)

Error (MSE) e bpp. O MSE mede a diferença entre duas imagens, normalmente utilizado para comparar a imagem original e a versão com os erros gerados pelo processo de codificação com perdas. O cálculo do MSE é definido para imagens de tamanho M×N e é dado pela Equação 2.3, onde $I(i,j)$ e $I'(i,j)$ são, respectivamente, a amostra original e a amostra com erro, ambas localizadas na linha i e coluna j.

$$MSE(I,I') = \frac{1}{N*M} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} (I(i,j) - I'(i,j))^2 \tag{2.3}$$

O cálculo da Relação Sinal-Ruído de Pico – *Peak Signal-to-Noise Ratio* (PSNR) utiliza a definição da MSE juntamente com n, o número de bits utilizados na representação do dado, e é descrito pela Equação 2.4.

$$PSNR(I,I') = 10log_{10}\left(\frac{2^n - 1}{MSE(I,I')}\right) \tag{2.4}$$

Figura 10 – Vista de exemplo do LF Tarot Cards.

Fonte: (Pereira et al., 2019)

Figura 11 – Vistas de exemplo de LFs sintéticos.

(a) Greek (b) Sideboard

Fonte: (Pereira et al., 2019)

Devido à maior sensibilidade do Sistema Visual Humano – *Human Visual System* (HVS) ao componente de luminância das imagens, em relação à crominância, o cálculo do PSNR para imagens coloridas é feito no espaço de cores *YCbCr*, de acordo com a recomendação BT.709-6 (ITU-T, 2008), tal que a luminância (Y) e a crominância (Cb, Cr) são ponderadas conforme a Equação 2.5.

$$PSNR_{YCbCr} = \frac{6 * PSNR_y + PSNR_{Cb} + PSNR_{Cr}}{8} \tag{2.5}$$

A principal métrica relacionada à taxa é o número de bits por pixel – *bits per pixel* (bpp), que é calculado utilizando o número total de bits N_TOT_BITS e o número total de pixels N_TOT_PIXELS de acordo com a Equação 2.6.

$$bpp = \frac{N\_TOT\_BITS}{N\_TOT\_PIXELS} \tag{2.6}$$

As CTCs também definem as taxas alvo de bits que devem ser utilizadas em experimentos realizados com este *dataset*, estes dados estão descritos na Tabela 2.

Tabela 2 – Taxas alvo definidas nas CTCs (Pereira et al., 2019).

Light Field	Target Bitrate (bpp)					
Todos os Lenslets	-	0.001	0.005	0.02	0.01	0.75
Greek e Sideboard	-	0.001	0.005	0.02	0.01	0.75
Tarot Cards	-	0.001	0.005	0.02	0.01	0.75
Set2 2K sub	0.0005	0.001	0.005	0.01	0.05	0.1
Poznan Laboratory 1	0.0005	0.001	0.005	0.01	0.05	0.1

Fonte: (Pereira et al., 2019).

2.4 TRABALHOS CORRELATOS

Algoritmos de controle de taxa são comuns em codificadores que aceitam perdas. Este recurso é particularmente útil quando a mídia precisa adaptar-se ao seu meio, como aos dispositivos com espaço limitado em que serão armazenadas, ou à banda de rede necessária para transmissões em tempo real. Nestes casos, a perda de qualidade é aceitável mediante às restrições.

Diversos algoritmos já foram propostos para realizar o controle de taxa de variadas mídias e métodos de compressão. O método de Otimização de Taxa-Distorção Pós-Compressão – *Post-Compression Rate-Distortion Optimization* (PCRD-Opt) (Balster; Fortener; Turri, 2010) é o algoritmo recomendado pelo padrão de imagens estáticas JPEG 2000 (ISO Central Secretary, 2019). Outros trabalhos, como o de Amor, Bruns e Sparenberg (2017), propõem alternativas mais eficientes em termos de velocidade e complexidade de algoritmo. Tanto no caso do PCRD-Opt, quanto no trabalho de Amor, Bruns e Sparenberg (2017), os métodos são integrados ao codificador de entropia.

No caso do padrão JPEG XR (ISO Central Secretary, 2019) o trabalho de Chan, Liang e Tu (2010) propôs um método baseado no domínio ρ, que possui uma forte correlação linear com o tamanho da imagem após a codificação. São chamadas de análises no domínio ρ aquelas que correlacionam a taxa de compressão à porcentagem de coeficientes iguais a zero em uma imagem quantizada, onde ρ é o valor desta porcentagem. O trabalho de Chan, Liang e Tu (2010) utiliza esta correlação, que é linear, juntamente com duas codificações anteriores para estimar o valor de quantização necessário para alcançar determinada taxa.

Nos padrões de codificação de vídeos HEVC e VVC, os métodos de controle de taxa e distorção normalmente estão relacionados aos processos de alocação de bits, otimização de taxa e distorção e à escolha de parâmetro de quantização (Ahmad et al., 2022). Um dos principais aspectos de um algoritmo de controle de taxa-distorção para estes padrões envolve a predição do comportamento da RDO. Para isso, os seguintes modelos são comumente utilizados:

1. Q-Domain RD model, que prediz a taxa com base no parâmetro de quantização (Ma; Gao; Lu, 2005; Choi et al., 2013);

2. ρ-Domain RD model, que, assim como no padrão JPEG XR, estima a taxa através da porcentagem de zeros dos coeficientes quantizados (He; Kim; Mitra, 2001; Milani; Celetto; Mian, 2008); e

3. λ-Domain RD model que utiliza a inclinação da curva de taxa e distorção para estimar a taxa (Li et al., 2014; Yang et al., 2019).

A partir da alocação de bits e do modelo de taxa de distorção escolhido, um valor diferente de Parâmetro de Quantização – *Quantization Parameter* (QP) é utilizado. De acordo com a eficácia na obtenção da taxa desejada, o modelo para predição da RDO é atualizado.

No trabalho de Li et al. (2014), os autores propõem um novo algoritmo baseado no modelo R-λ para realizar o controle de taxa em codificadores que seguem o padrão HEVC. O algoritmo explora a forte correlação que o modelo R-λ possui com a função hiperbólica apresentada na Equação 2.7, onde α e β são parâmetros relacionados à fonte de vídeo.

$$\lambda = \alpha * R^{\beta} \tag{2.7}$$

Os autores também consideram a premissa de que Unidades Básicas – *Basic Units* (BUs) próximas, que neste caso são os blocos de codificação de vídeo, possuem valores semelhantes de α e β. Portanto, a primeira BU utiliza valores fixos de α e β, e o modelo é atualizado a cada codificação através das Equações 2.8 até 2.10 para ser utilizado nas próximas BUs. Nestas equações, α_{old} e β_{old} são os valores de α e β encontrados anteriormente, enquanto α_{new} e β_{new} são os que estão sendo calculados na iteração atual. Além disso, λ_{comp} é o valor computado como estimativa para λ, enquanto λ_{real} é o valor real de λ utilizado no processo interno de RDO. Por fim, R_{real} é o valor de taxa resultante de uma codificação.

$$\lambda^{\text{comp}} = \alpha^{\text{old}} \left(\text{R}^{\text{real}}\right)^{\beta^{\text{old}}} \tag{2.8}$$

$$\alpha^{\text{new}} = \alpha^{\text{old}} + \delta^{\alpha} * (ln(\lambda^{\text{real}}) - ln(\lambda^{\text{comp}})) * \alpha^{\text{old}} \tag{2.9}$$

$$\beta^{\text{new}} = \beta^{\text{old}} + \delta^{\beta} * (ln(\lambda^{\text{real}}) - ln(\lambda^{\text{comp}})) * ln(\text{R}^{\text{real}}) \tag{2.10}$$

Não foram encontradas propostas de algoritmos de controle de taxa criados especificamente para LFs. Dependendo das técnicas empregadas na codificação de um LF, é possível que alguns dos algoritmos citados demonstrem-se adequados, mas este não é o caso para o 4DTM adotado no padrão JPEG Pleno. Isso acontece, principalmente, porque a quantização neste padrão é feita através de bitplanes, e porque a única forma de controlar a relação entre taxa e distorção no JPLM é através de um multiplicador Lagrangiano (λ, apresentado na Equação 2.2), que é usado no processo de RDO. Por outro lado, codificadores de imagem e vídeo costumam utilizar um QP para fazer o controle de RD, e a quantização é feita através de divisões inteiras.

Apesar destas diferenças, trabalhos que realizam análises no domínio λ, como é o caso de Li et al. (2014), podem fornecer informações úteis para criação de um algoritmo nativo de controle de taxa para codificação de LFs no 4DTM, conforme será apresentado nos capítulos seguintes.

3 PROPOSTAS DE ALGORITMOS

Este capítulo apresenta diferentes soluções para o controle de taxa em LFs para o 4DTM. As propostas foram criadas iterativamente, e cada uma delas apresenta novidades a fim de corrigir problemas da proposta anterior.

Para encontrar um algoritmo efetivo de controle de taxa, é possível tratar o codificador como uma função $C : \mathbb{R} \to \mathbb{R}^2$, que mapeia um multiplicador Lagrangiano (λ) para um par de taxa (R) e distorção (D). Deste modo, o problema do controle de taxa pode ser definido como uma tentativa de encontrar um valor λ^{target} tal que $C(\lambda^{\text{target}}) = (R_\text{i}, D_\text{i})$ e $|R_\text{i} - R^{\text{target}}| \leq \varepsilon$, onde R^{target} é a taxa desejada para o arquivo codificado, e ε é o erro máximo aceitável. Ou seja, λ^{target} é o valor de λ que deve ser usado para atingir a taxa-alvo (R^{target}).

3.1 BISSEÇÃO

Um bom ponto de partida para resolver este tipo de problema é utilizando um algoritmo de busca binária, tal qual o método da bisseção (Burden; Faires; Burden, 2015). A bisseção é um dos métodos mais simples usados para encontrar, dado um erro máximo (ε), a raiz de uma função ($f(x) = 0$). Podemos adaptar o método para resolver o problema do controle de taxa como segue.

A bisseção inicia com um intervalo que garantidamente contém o valor que está sendo buscado (λ^{target}). Neste caso, será utilizado um intervalo grande o suficiente para englobar todos os lambdas possíveis da região de interesse. Em seguida, o algoritmo divide o intervalo ao meio, verifica qual das duas partes ainda contém a taxa buscada e utiliza este subintervalo para o próximo passo de busca. Este processo é, então, repetido até encontrar o valor desejado, uma aproximação boa o suficiente, ou até que um limite de iterações seja atingido.

Parafraseando de maneira mais precisa, o algoritmo começa com os valores λ_0^{start} e λ_0^{end}, representando, respectivamente, o início e o fim do intervalo que contém o valor λ^{target}. É necessário codificar o LF tanto com λ_0^{start} quanto com λ_0^{end}, para obter a taxa relacionada com cada um deles. Então, um λ intermediário (λ^{half}) é calculado conforme a Equação 3.1 e uma nova codificação é executada com λ^{half}, tal que se obtém uma taxa R^{half}. Em cada uma das iterações subsequentes, o intervalo é atualizado de acordo com a Equação 3.2. O valor de n nas Equações 3.1 e 3.2 representa o número da iteração atual.

$$\lambda_n^{\text{half}} = \frac{\lambda_n^{\text{start}} + \lambda_n^{\text{end}}}{2} \tag{3.1}$$

$$(\lambda_{n+1}^{\text{start}}, \lambda_{n+1}^{\text{end}}) = \begin{cases} (\lambda_n^{\text{half}}, \lambda_n^{\text{end}}) & \text{se } R_n^{\text{half}} > R^{\text{target}} \\ (\lambda_n^{\text{start}}, \lambda_n^{\text{half}}) & \text{caso contrário} \end{cases} \tag{3.2}$$

O Algoritmo 1 apresenta uma possível implementação para o algoritmo da bisseção no contexto de um codificador de LFs. O algoritmo descrito tem como entrada a taxa-alvo (R^{target}),

a precisão desejada (ε) e o número máximo de iterações desejado (max_iter) para atingir a precisão. O retorno será o λ encontrado; caso seja retornado antes de atingir o número máximo de iterações, deve ser igual a λ^{target}.

Algoritmo 1: Bisseção

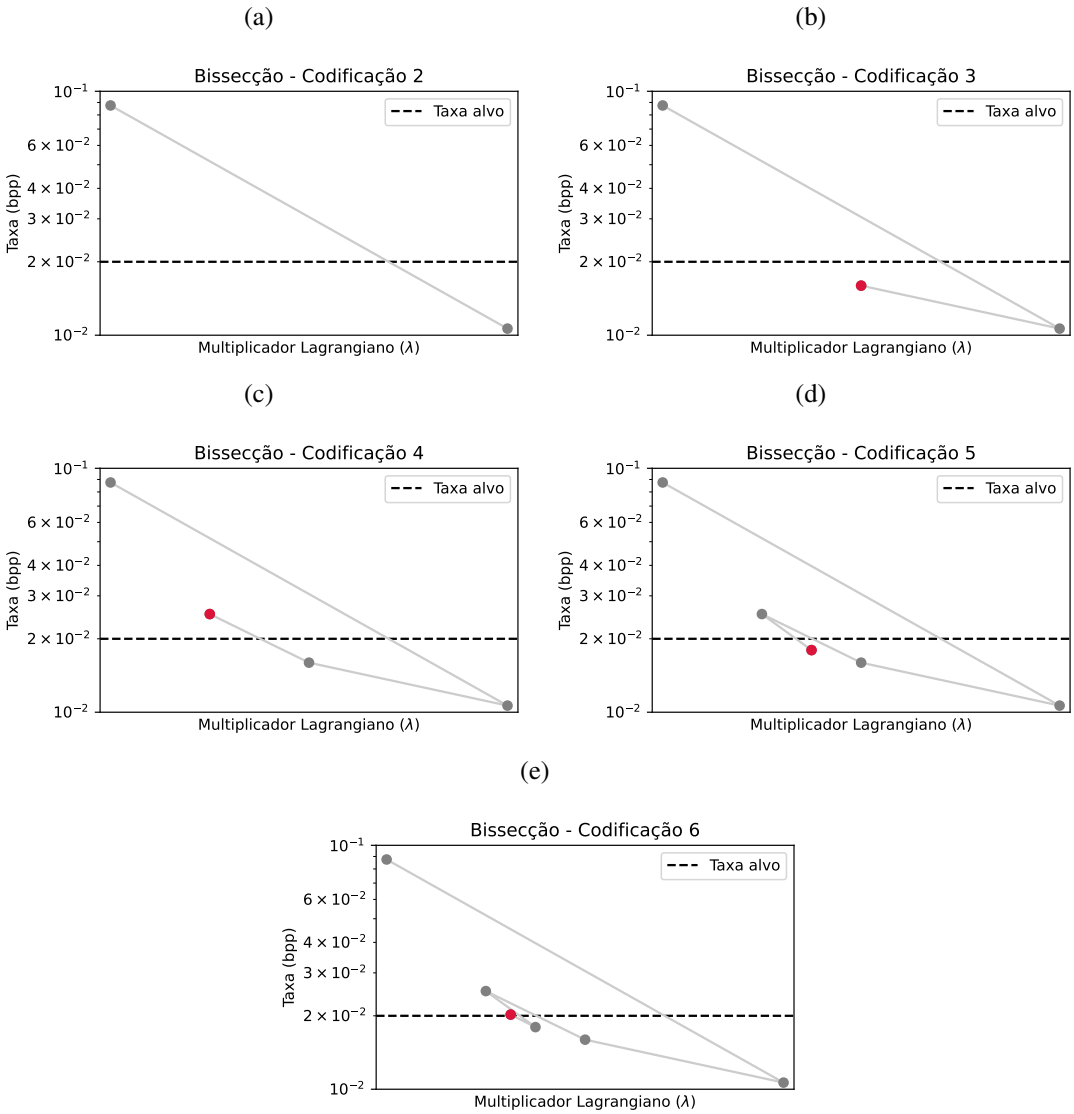
Entrada: R^{target}, ε, max_iter
Saída: λ

$\lambda^{\text{start}} \leftarrow 1$
$\lambda^{\text{end}} \leftarrow 10^9$
$R^{\text{start}}, \_ \leftarrow C(\lambda^{\text{start}})$ // codifica o LF com λ e retorna a taxa e a distorção (esta última não é usada)
se $|R^{start} - R^{target}| < \varepsilon$ **então**
    retorna λ^{start}
fim
$R^{\text{end}}, \_ \leftarrow C(\lambda^{\text{end}})$
se $|R^{end} - R^{target}| < \varepsilon$ **então**
    retorna λ^{end}
fim
para $n = 0$ **até** $max\_iter - 1$ **faça**
    $\lambda^{\text{half}} \leftarrow (\lambda^{\text{start}} - \lambda^{\text{end}})/2$ // Equação 3.1.
    $R, \_ \leftarrow C(\lambda^{\text{half}})$
    se $|R - R^{target}| < \varepsilon$ **então**
        retorna λ^{half}
    fim
    /* Equação 3.2: */
    se $R > R^{target}$ **então**
        $\lambda^{\text{start}} \leftarrow \lambda^{\text{half}}$
    senão
        $\lambda^{\text{end}} \leftarrow \lambda^{\text{half}}$
    fim
fim
/* O algoritmo não convergiu em até max_iter iterações. Retorna a melhor estimativa de λ disponível até o momento. */
retorna $(\lambda^{start} - \lambda^{end})/2$

A Figura 12 ilustra, através de um exemplo, o processo de convergência do algoritmo de bisseção.

O algoritmo da bisseção é conceitualmente simples e sempre converge para uma solução (Burden; Faires; Burden, 2015). Porém, em determinados casos ele pode demorar para

Figura 12 – Exemplo ilustrativo do algoritmo de bisseção aplicado ao problema de controle de taxa. Os pontos mostram a relação entre λ e taxa em uma codificação. O ponto vermelho representa a codificação atual. A reta tracejada preta marca o valor da taxa-alvo (R^{target}).



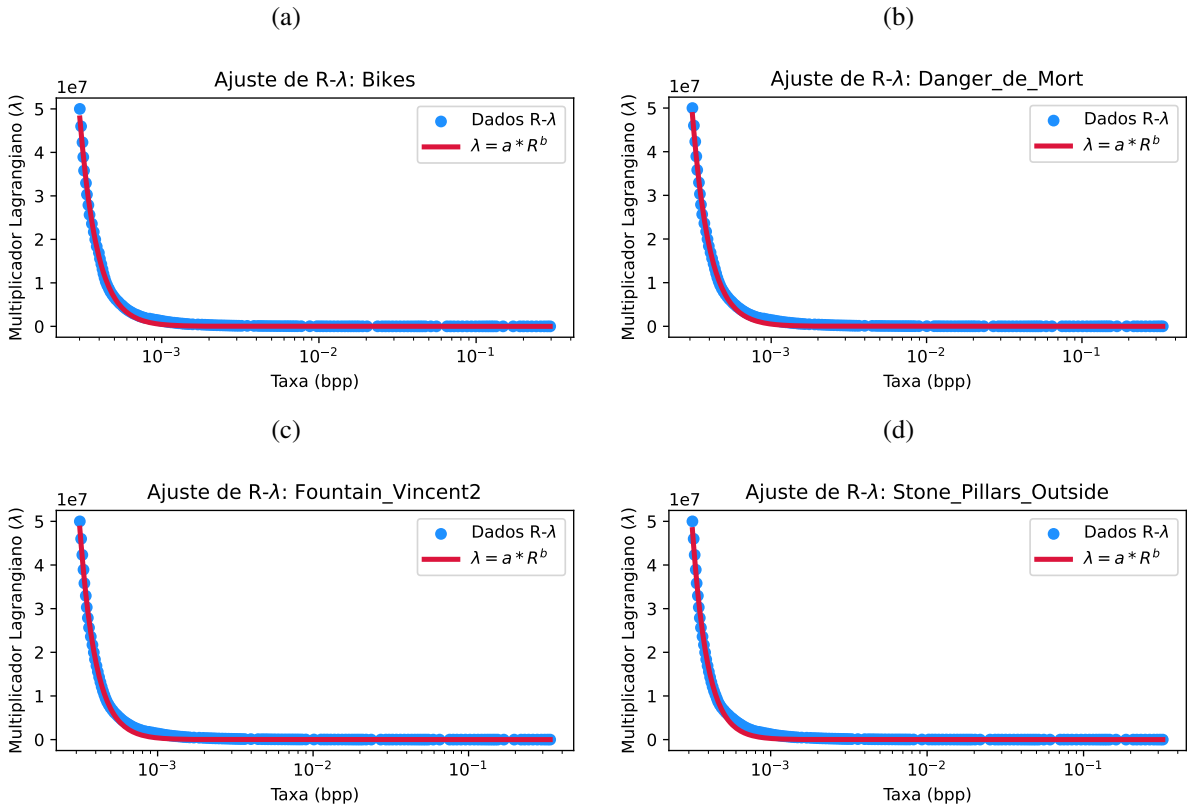
Fonte: o autor.

convergir (Conte; Boor, 1980; Burden; Faires; Burden, 2015), dependendo muito do erro máximo desejado (ε) e do comportamento dos dados que estamos aproximando. Note que um número alto de iterações causará um aumento significativo no tempo de codificação, uma vez que é necessário codificar o LF para cada iteração. Portanto, os próximos métodos tentam levar em consideração o comportamento dos dados de codificação para acelerar a convergência, tal qual é feito em aproximações numéricas para encontrar as raízes de uma função (Conte; Boor, 1980).

3.2 HYPERBOLIC SPLIT

A fim de explorar a não linearidade dos dados, verificamos que a função hiperbólica é uma boa candidata para predição de LFs codificados através do 4DTM, de forma semelhante ao que acontece no trabalho de Li et al. (2014) em relação a vídeos codificados com o HEVC. Esta característica pode ser observada através da Figura 13 para os LFs *lenslets*.

Figura 13 – Distribuições de taxa e lambda comparadas com o modelo hiperbólico. Os pontos representam dados reais de codificação utilizando o JPLM, enquanto a linha apresenta a curva hiperbólica que melhor se aproxima da distribuição. Note que a taxa está representada no eixo das abscissas, pois no contexto do controle de taxa, R^{target} é o valor conhecido e busca-se o valor de λ^{target}. (a) Bikes. (b) Danger de Mort. (c) Fountain Vincent 2. (d) Stone Pillars Outside.



Fonte: o autor.

A primeira forma encontrada para explorar o modelo hiperbólico dentro do JPLM foi utilizando este modelo para otimizar o particionamento do intervalo utilizado na bisseção. Em vez de dividir o intervalo exatamente na metade, podemos usar as informações já computadas de R e o λ correspondente para estimar, através do modelo hiperbólico, uma aproximação da taxa-alvo. Esta aproximação será, então, utilizada para dividir o intervalo em uma posição mais próxima do alvo. Com esta pequena mudança, é esperado que a convergência aconteça mais rapidamente.

Para encontrar uma boa aproximação para o λ^{target} utilizando o modelo hiperbólico, iremos definir uma função λ^{hsplit} como demonstrado na Equação 3.3.

$$\lambda(R) \approx \lambda^{\text{hsplit}}(R) := \alpha * R^{\beta} \tag{3.3}$$

Em seguida, é necessário achar valores de α_n e β_n tais que $\lambda^{\text{hsplit}}(R_n^{\text{start}}) = \lambda_n^{\text{start}}$ e $\lambda^{\text{hsplit}}(R_n^{\text{end}}) = \lambda_n^{\text{end}}$. Estes requisitos implicam nas Equações 3.4 e 3.5, as quais precisam ser satisfeitas em cada iteração do algoritmo.

$$\lambda_n^{\text{start}} = \alpha_n * (R_n^{\text{start}})^{\beta_n} \tag{3.4}$$

$$\lambda_n^{\text{end}} = \alpha_n * (R_n^{\text{end}})^{\beta_n} \tag{3.5}$$

Precisamos encontrar α_n e β_n para obtermos a aproximação apresentada na Equação 3.3. Os demais valores das Equações 3.4 e 3.5 são conhecidos (os valores de λ foram usados para obter as taxas). Podemos isolar o termo β_n dividindo a Equação 3.4 pela Equação 3.5, assim eliminando o termo α_n, como apresentado na Equação 3.6.

$$\frac{\lambda_n^{\text{start}}}{\lambda_n^{\text{end}}} = \frac{\cancel{\alpha_n}}{\cancel{\alpha_n}} * \left(\frac{R_n^{\text{start}}}{R_n^{\text{end}}}\right)^{\beta_n} \tag{3.6}$$

O fato do termo β_n aparecer no expoente da equação dificulta sua manipulação algébrica. Isso pode ser resolvido utilizando a propriedade do logaritmo da potência ($log(a^b) = b * log(a)$) para que β_n apareça na forma de um produto. Para isto, basta calcular o logaritmo de ambos os lados da equação. A base deste logaritmo não é particularmente importante neste caso, desde que a mesma seja utilizada em ambos os lados. Para fins de implementação consideramos que log refere-se à base dois, pois assim o cálculo do logaritmo pode ser feito de maneira mais eficiente em processadores modernos, seja através de algoritmos rápidos (Turner, 2010) ou de instruções específicas (Intel, 2024). Estes passos são demonstrados nas Equações 3.7 e 3.8. Finalmente, a Equação 3.9 apresenta o termo β_n isolado.

$$log\left(\frac{\lambda_n^{\text{start}}}{\lambda_n^{\text{end}}}\right) = log\left(\left(\frac{R_n^{\text{start}}}{R_n^{\text{end}}}\right)^{\beta_n}\right) \qquad \text{(log em ambos os lados da Eq. 3.6)} \tag{3.7}$$

$$log\left(\frac{\lambda_n^{\text{start}}}{\lambda_n^{\text{end}}}\right) = \beta_n * log\left(\frac{R_n^{\text{start}}}{R_n^{\text{end}}}\right) \qquad \text{(aplica } log(a^b) = b * log(a)\text{)} \tag{3.8}$$

$$\beta_n = \frac{log\left(\lambda_n^{\text{start}}\right) - log(\lambda_n^{\text{end}})}{log(R_n^{\text{start}}) - log(R_n^{\text{end}})} \qquad \text{(isola } \beta_n\text{)} \tag{3.9}$$

Através do termo β_n, obtido conforme a Equação 3.9, e da Equação 3.4, o termo α_n pode ser obtido conforme apresentado na Equação 3.10.

$$\alpha_n = \lambda_n^{\text{start}} * (R_n^{\text{start}})^{-\beta_n} \tag{3.10}$$

Com estes valores calculados, o intervalo de lambdas será atualizado de maneira semelhante à bisseção, através das Equações 3.11 e 3.12.

$$\lambda_n^{\text{half}} = \alpha_n * (R^{\text{target}})^{\beta_n} \qquad (\alpha_n \text{ definido na Eq. 3.10 e } \beta_n \text{ na Eq. 3.9}) \tag{3.11}$$

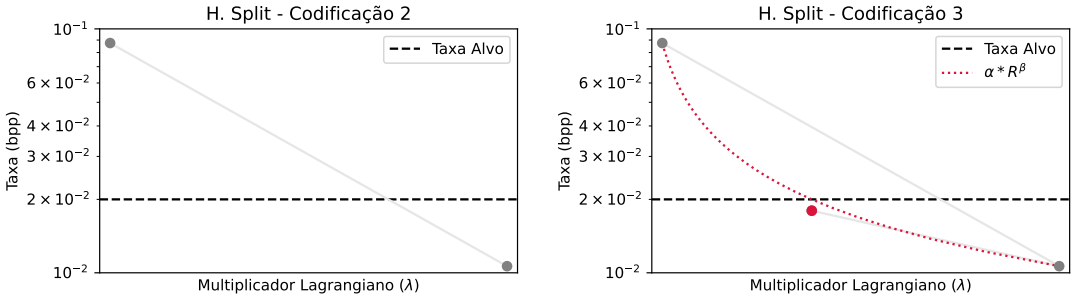
$$(\lambda_{n+1}^{\text{start}}, \lambda_{n+1}^{\text{end}}) = \begin{cases} (\lambda_n^{\text{half}}, \lambda_n^{\text{end}}) & \text{if } R_n^{\text{half}} > R^{\text{target}} \\ (\lambda_n^{\text{start}}, \lambda_n^{\text{half}}) & \text{otherwise} \end{cases} \tag{3.12}$$

Caso haja algum problema no cálculo dos valores de α ou β, o valor de λ_n^{half} será calculado do mesmo modo feito para o algoritmo de bisseção, a fim de evitar erros. O processo pode ser mais facilmente compreendido através do Algoritmo 2, que possui as mesmas entradas e saída que o Algoritmo 1. Ainda, ambos os algoritmos são equivalentes entre as linhas 1 e 10.

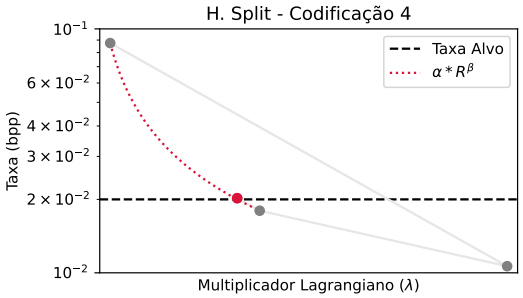
A Figura 14 apresenta um exemplo das iterações necessárias para a convergência do algoritmo *Hyperbolic Split* (H. Split). É possível perceber na figura a curva utilizada na predição de cada etapa, e que o valor escolhido de λ localiza-se exatamente na intersecção entre a predição e a taxa-alvo. Também é claro como esta predição, apesar de não ser exata, aproxima-se bastante do resultado esperado. Comparado com a Figura 12, nota-se que a única diferença entre os algoritmos é a utilização de estimativas mais elaboradas para a divisão do intervalo, e como estas estimativas resultam em um número menor de codificações.

Figura 14 – Exemplo ilustrativo do algoritmo de H. Split aplicado ao problema de controle de taxa. Os pontos mostram a relação entre λ e taxa em uma codificação. O ponto vermelho representa a codificação atual. A reta tracejada preta marca o valor da taxa-alvo (R^{target}). A linha pontilhada vermelha representa a predição utilizada para encontrar o λ utilizado na codificação atual.

(b)



(c)



Fonte: o autor.

Apesar das melhorias oferecidas pelo algoritmo H. Split, em relação à bisseção, ele ainda possui uma característica que pode ser aprimorada: a necessidade de codificações iniciais.

Algoritmo 2: Hyperbolic Split

Entrada: R^{target}, ε, max_iter
Saída: λ

1 $\lambda^{\text{start}} \leftarrow 1$
2 $\lambda^{\text{end}} \leftarrow 10^9$
3 $R^{\text{start}}, \_ \leftarrow C(\lambda^{\text{start}})$ // codifica o LF com λ e retorna a taxa e distorção.
4 **se** $|R^{start} - R^{target}| < \varepsilon$ **então**
5 | **retorna** λ^{start}
6 **fim**
7 $R^{\text{end}}, \_ \leftarrow C(\lambda^{\text{end}})$
8 **se** $|R^{end} - R^{target}| < \varepsilon$ **então**
9 | **retorna** λ^{end}
10 **fim**
11 **para** $i = 0$ **até** $max\_iter - 1$ **faça**
12 | **se** $R^{start} \neq R^{end}$ **então**
13 | | $\beta \leftarrow (log_2\lambda^{\text{start}} - log_2\lambda^{\text{end}})/(log_2 R^{\text{start}} - log_2 R^{\text{end}})$ // Equação 3.9.
14 | | $\alpha \leftarrow \lambda^{\text{start}} \times (R^{\text{start}})^{-\beta}$ // Equação 3.10.
15 | | $\lambda^{\text{half}} \leftarrow \alpha \times (R^{\text{target}})^{\beta}$ // Equação 3.11.
16 | **senão**
17 | | $\lambda^{\text{half}} \leftarrow (\lambda^{\text{start}} - \lambda^{\text{end}})/2$
18 | **fim**
19 | $R, \_ \leftarrow C(\lambda^{\text{half}})$
20 | **se** $|R - R^{target}| < \varepsilon$ **então**
21 | | **retorna** λ^{half}
22 | **fim**
23 | **se** $R > R^{target}$ **então**
24 | | $\lambda^{\text{start}} \leftarrow \lambda^{\text{half}}$
25 | **senão**
26 | | $\lambda^{\text{end}} \leftarrow \lambda^{\text{half}}$
27 | **fim**
28 **fim**
/* O algoritmo não convergiu em até max_iter iterações. Retorna a melhor estimativa de λ disponível até o momento. */
29 **se** $R^{start} \neq R^{end}$ **então**
30 | $\beta \leftarrow (log_2\lambda^{\text{start}} - log_2\lambda^{\text{end}})/(log_2 R^{\text{start}} - log_2 R^{\text{end}})$ // Equação 3.9.
31 | $\alpha \leftarrow \lambda^{\text{start}} \times (R^{\text{start}})^{-\beta}$ // Equação 3.10.
32 | $\lambda^{\text{half}} \leftarrow \alpha \times (R^{\text{target}})^{\beta}$ // Equação 3.11.
33 **senão**
34 | $\lambda^{\text{half}} \leftarrow (\lambda^{\text{start}} - \lambda^{\text{end}})/2$
35 **fim**
36 **retorna** λ^{half}

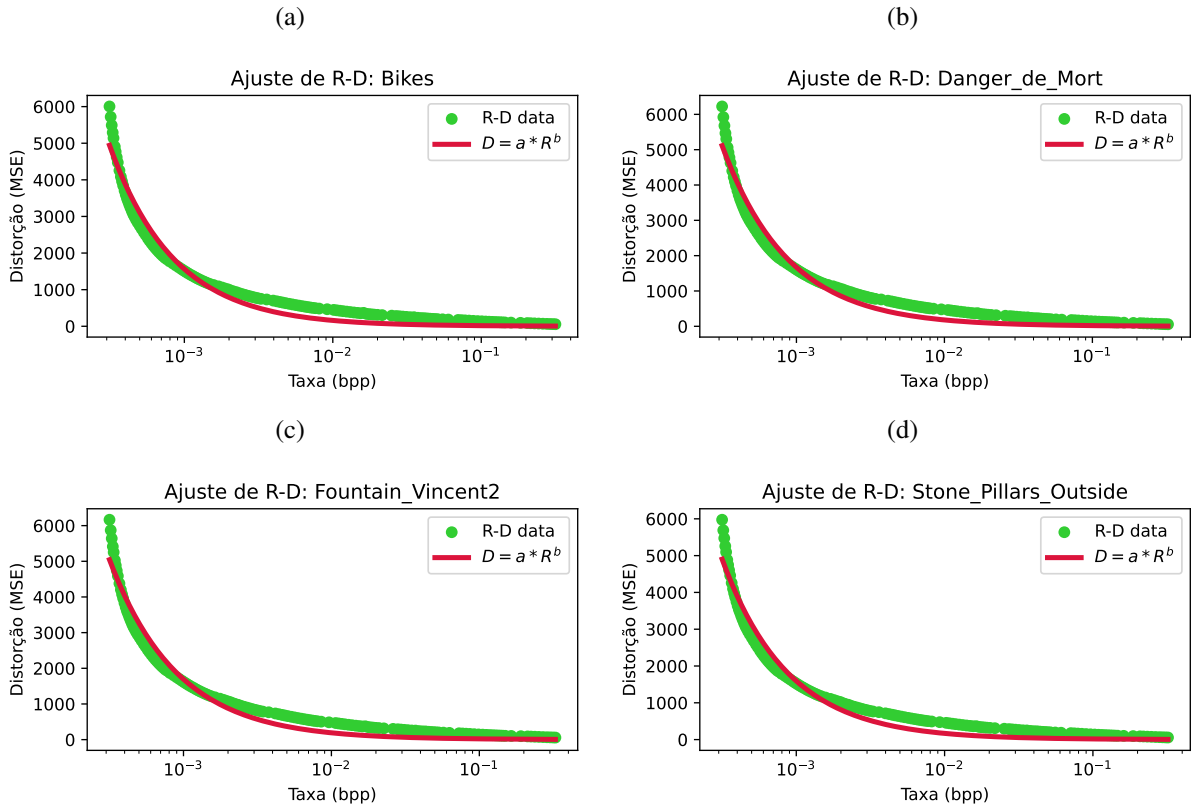
Devido à necessidade de que o intervalo inicial inclua a resposta, um dos valores iniciais precisa ser muito pequeno e o outro muito grande. Nas implementações de teste foram definidos, respectivamente, os valores 1 e 10^9 para λ_0^{start} e λ_0^{end} (linhas 1 e 2 dos Algoritmos 1 e 2). O

problema desta necessidade, é que o codificador normalmente gasta mais tempo para codificar LFs com λ's muito pequenos (altas taxas) do que com λ's maiores (baixas taxas). Portanto, é desejável que os algoritmos trabalhem sempre com codificações tão próximas quanto possível da região de interesse.

3.3 HYPERBOLIC SLOPE ALGORITHM

Para solucionar o problema da necessidade de duas codificações iniciais (uma com valor baixo de λ e outra com valor alto de λ) foi criado um novo algoritmo chamado *Hyperbolic Slope* (H. Slope). Esse algoritmo explora o fato de que, conforme mostra a Figura 15, o modelo hiperbólico também se adapta relativamente bem à curva de RD. Este comportamento também é previsto no contexto do trabalho de Li et al. (2014).

Figura 15 – Distribuições de RD comparadas com o modelo hiperbólico. (a) Bikes. (b) Danger de Mort. (c) Fountain Vincent 2. (d) Stone Pillars Outside.



Fonte: o autor.

Outra característica explorada foi o fato de que, em uma curva de RD, o valor de λ representa o inverso da inclinação desta curva em determinado ponto, conforme explicado na Seção 2.2 (Figura 6). Esta relação é expressa na Equação 3.13.

$$\lambda = -\frac{\mathrm{d}}{\mathrm{d}R} D(R) \tag{3.13}$$

Utilizando o modelo hiperbólico para aproximar a curva de taxa e distorção temos que:

$$D(R) \approx D^{\text{hslope}}(R) := \alpha * R^{\beta} \tag{3.14}$$

Deste modo, considerando as Equações 3.13 e 3.14, podemos definir uma nova aproximação para $\lambda(R)$, chamada de $\lambda^{\text{hslope}}(R)$, através da derivada de $D^{\text{hslope}}(R)$, conforme mostrado nas Equações 3.15 e 3.16. Derivando a Equação 3.16 obtém-se a Equação 3.17, que será efetivamente utilizada para calcular as estimativas de λ neste algoritmo.

$$\lambda(R) \approx \lambda^{\text{hslope}}(R) := -\frac{\mathrm{d}}{\mathrm{d}R} D^{\text{hslope}}(R) \qquad \text{(define a } \lambda^{\text{hslope}} \text{ a partir de 3.13 e 3.14)} \tag{3.15}$$

$$\lambda^{\text{hslope}}(R) = -\frac{\mathrm{d}}{\mathrm{d}R} \alpha * R^{\beta} \qquad \text{(substitui } D^{\text{hslope}}(R) \text{ por sua definição)} \tag{3.16}$$

$$\lambda^{\text{hslope}}(R) = -\alpha * \beta * R^{\beta-1} \qquad \text{(resolve a derivada)} \tag{3.17}$$

Estas relações permitem que o comportamento das curvas de RD e R-λ sejam modeladas juntamente através dos termos α_n e β_n. Para isso é necessário encontrar valores de α_n e β_n, de modo que as equações $D^{\text{hslope}}(R_n) = D_n$ e $\lambda^{\text{hslope}}(R_n) = \lambda_n$ sejam satisfeitas. Onde D_n, R_n e λ_n são, respectivamente, a distorção e a taxa de um LF e o multiplicador Lagrangiano utilizado.

Para achar os valores que satisfazem estas equações a cada passo de iteração do algoritmo, é útil reescrevê-las da seguinte forma, onde n é o número atual de iteração:

$$D_n = \alpha_n * (R_n)^{\beta_n} \tag{3.18}$$

$$\lambda_n = -\alpha_n * \beta_n * (R_n)^{\beta_n - 1} \tag{3.19}$$

O expoente (-1) presente na Equação 3.19 pode ser removido ao dividir o lado direito da equação por R_n, como demonstrado na Equação 3.20. Este passo é importante, pois destaca o termo $\alpha_n(R_n)^{\beta_n}$, que é justamente a definição de D_n apresentado em 3.18. Portanto, a Equação 3.21 substitui este trecho por D_n, de modo que λ_n seja expresso sem utilizar o termo α_n.

$$\lambda_n = -\frac{\beta_n}{R_n} * \alpha_n (R_n)^{\beta_n} \tag{3.20}$$

$$\lambda_n = -\frac{\beta_n}{R_n} * D_n \tag{3.21}$$

Deste modo, com os valores já conhecidos de λ_n, R_n e D_n (os valores de λ foram usados para obter as taxas e as distorções), β_n é calculado através da Equação 3.22.

$$\beta_n = -\frac{\lambda_n * R_n}{D_n} \tag{3.22}$$

Já o valor de α_n é obtido utilizando os valores de β_n, R_n e D_n de maneira semelhante à forma apresentada no algoritmo H. Split e demonstrado na Equação 3.23.

$$\alpha_n = D_n * (R_n)^{-\beta_n} \tag{3.23}$$

Através destes cálculos, assumindo que uma única codificação já tenha sido feita, uma aproximação de λ^{target} pode ser calculada conforme a Equação 3.24.

$$\lambda^{\text{est}} = -\alpha_n * \beta_n * (R_n)^{\beta_n - 1} \tag{3.24}$$

Esta aproximação tende a resultar em valores cada vez mais próximos à taxa-alvo. Porém, dependendo da precisão desejada, é possível que a aproximação entre em *loop*, oscilando em torno da taxa-alvo e nunca convergindo efetivamente. Para garantir a convergência do algoritmo é necessário que a cada passo a taxa obtida esteja mais próxima da taxa-alvo do que no passo anterior. Portanto, podemos utilizar um fator de convergência (f), que faz uma interpolação entre o valor atual de λ e a aproximação obtida, evitando que a aproximação fique oscilando em torno da taxa-alvo sem convergir.

O fator de convergência f utilizado inicia com o valor 1. Cada vez que a estimativa ultrapassa a taxa desejada, seja acima ou abaixo, o fator é reduzido pela metade. Este comportamento está expresso na Equação 3.25.

$$f_{n+1} = \begin{cases} f_n/2 & \text{if } \min(R_n, R_{n-1}) < R^{\text{target}} < \max(R_n, R_{n-1}) \\ f_n & \text{otherwise} \end{cases} \tag{3.25}$$

A utilização do fator de convergência para estimativa de lambda, da forma como está implementada no algoritmo, é demonstrada na Equação 3.26:

$$\lambda_{n+1} = \lambda^{\text{est}} * f + \lambda * (1 - f) \tag{3.26}$$

Estes cálculos são apresentados no Algoritmo 3, que possui as mesmas entradas e saída que os dois algoritmos apresentados anteriormente. O λ inicial utilizado neste algoritmo é calculado através de valores α e β definidos empiricamente, na expectativa de que todas as codificações aconteçam próximas da área de interesse. Este Algoritmo não utiliza os trechos iniciais de código compartilhados entre os Algoritmos 1 e 2, justamente por não depender de intervalos e nem necessitar das codificações iniciais.

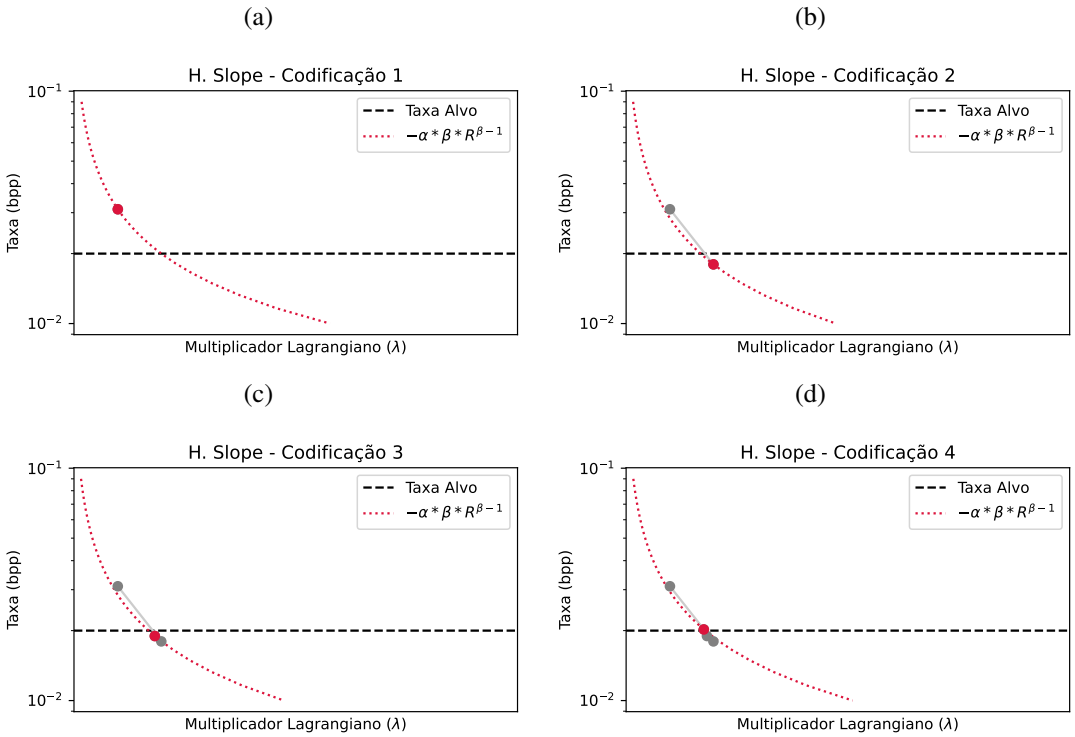
As primeiras iterações do algoritmo H. Slope são ilustradas na Figura 16. Assim como no H. Split, é possível perceber que os valores escolhidos de λ em cada ponto também localizam-se exatamente na intersecção entre a curva de predição anterior e da taxa-alvo. Vale notar, na figura, que as predições neste caso são geradas através de uma única codificação, diferente do que acontece na bissecção e no H. Split.

Algoritmo 3: Hyperbolic Slope

Entrada: R^{target}, ε, max_iter
Saída: λ

$f \leftarrow 1$ // fator de convergência
$\alpha \leftarrow 25$
$\beta \leftarrow -1.5$
$\lambda \leftarrow a \times (R^{\text{target}})^{\beta}$
para $i = 0$ **até** $max\_iter - 1$ **faça**
    $R, D \leftarrow C(\lambda)$ // codifica o LF com λ e retorna RD
    se $|R - R^{target}| < \varepsilon$ **então**
        retorna λ
    fim
    $\beta \leftarrow \lambda \times R/D$
    $\alpha \leftarrow D \times R^{\beta}$
    $\lambda^{\text{est}} \leftarrow \alpha \times \beta \times (R^{\text{target}})^{-(\beta+1)}$
    se $i \neq 0$ ***e*** $R^{target} \in]min(R, R^{last}), max(R, R^{last})[$ **então**
        $f \leftarrow f/2$
    fim
    $\lambda \leftarrow \lambda^{\text{est}} \times f + \lambda \times (1 - f)$ // Equação 3.26
    $R^{\text{last}} \leftarrow R$
fim
/* O algoritmo não convergiu em até max_iter iterações. */
retorna λ

Figura 16 – Exemplo ilustrativo do algoritmo H. Slope aplicado ao problema de controle de taxa. Os pontos mostram a relação entre λ e taxa em uma codificação. O ponto vermelho representa a codificação atual. A reta tracejada preta marca o valor da taxa-alvo. A linha pontilhada vermelha representa a predição utilizada para encontrar o λ utilizado na codificação atual.



Fonte: o autor.

4 RESULTADOS

4.1 CONFIGURAÇÃO EXPERIMENTAL

Os três algoritmos propostos (bisseção, H. Split e H. Slope) foram implementados em duas versões. Uma delas feita utilizando linguagem Python 3.11.2, a fim de facilitar testes rápidos e a visualização de resultados. A outra foi feita na linguagem C++ 11 e adaptada no código de referência do JPLM para efetivamente avaliar a eficiência dos métodos no local onde serão primariamente utilizados. Em ambos os casos, os valores de taxa e distorção foram obtidos através de estimativas que já estavam disponíveis no JPLM. Anteriormente, estas estimativas eram usadas apenas para o processo de RDO.

Para configurar o controle de taxa, algumas opções foram adicionadas à Interface de Linha de Comando – *Command Line Interface* (CLI) do JPLM. Estas novas opções estão listadas na Tabela 3.

Tabela 3 – Argumentos criados na CLI do JPLM e suas respectivas funções.

Flag Compacta	**Flag Completa**	**Função**	**Valor Padrão**
-tr	--target-rate	Encontra o lambda correspondente a uma taxa-alvo em bpp e utiliza no processo de RDO para o modo 4DTM. Se o valor for igual a zero, o parâmetro --lambda é utilizado diretamente.	0
-tra	--target-rate-algorithm	Escolhe o algoritmo de controle de taxa para o modo 4DTM. As opções disponíveis são: • bisection • hsplit • hslope	hslope
-tri	--target-rate-iterations	Define o número máximo de iterações permitidas durante o processo de controle de taxa.	20

Fonte: o autor.

Os experimentos relacionados ao tempo de codificação foram executados em lote através de *scripts* escritos em Python, que realizam chamadas de sistema para executar o codificador. Cada experimento relacionado ao tempo de execução foi repetido 10 vezes, a fim de evitar vieses. O número de repetições poderia ser aumentado se fosse observada uma variabilidade muito grande nas medições, porém o coeficiente de variação calculado para cada taxa foi de,

em média, $1,13\%$, um valor bem baixo que indica uma boa representatividade dos dados. No caso de experimentos relacionados ao número de iterações dos algoritmos isto não é necessário, pois tanto os algoritmos quanto o software de referência são totalmente determinísticos. Todos os algoritmos foram limitados a um número máximo de 40 iterações (*i.e.*, max_iter=40). Foi escolhido um número máximo de iterações maior do que o padrão de 20 iterações para garantir que o algoritmo da bisseção convergisse para todos os casos testados, a fim de deixar as análises mais precisas. Além disso, o teste de convergência admite um erro de no máximo 1% da taxa-alvo (*i.e.*, $\varepsilon = 0,01 * R^{\text{target}}$).

A máquina utilizada para executar todos os experimentos está equipada com um processador AMD Ryzen™ 9 7950X@4.5GHz, que possui 16 núcleos físicos e 32 núcleos virtuais, e tamanhos de cache nos níveis 1, 2 e 3 de 1MB, 16MB e 64MB respectivamente. Em relação a memória RAM, o computador possui 4 pentes de memória DDR5 dual-channel DRAM@4000MHz, com 32GB cada, totalizando 128GB. Os LFs, tanto no formato original quanto codificados, foram armazenados em um SSD de 2TB do modelo Kingston KC3000 PCIe 4.0 NVMe M.2. O codificador foi configurado para utilizar todos os 16 núcleos físicos disponíveis na máquina. Estes experimentos, da forma como foram conduzidos, foram viabilizados pelo trabalho de Seidel, Fernandes e Güntzel (2024), que criaram uma versão paralela do codificador, reduzindo consideravelmente o tempo de execução necessário.

Todas as execuções utilizaram um subconjunto de LFs presentes nas CTC (Pereira et al., 2019), e têm como alvo suas respectivas taxas requeridas. O subconjunto escolhido exclui apenas os LFs "Set2 2K sub" e "Poznan Laboratory 1" devido ao tempo elevado que é necessário para a codificação destes LFs. Outro motivo importante que motivou esta escolha é o fato de que o modo 4DTM é mais eficiente para LFs capturados por câmeras plenópticas, enquanto o modo 4DPM funciona melhor para HDCA (Alves et al., 2020). Ambos os LFs removidos foram capturados por dispositivos do tipo HDCA.

O subconjunto de LFs escolhidos e as taxas correspondentes estão representados na Tabela 4.

Tabela 4 – LFs e taxas alvo escolhidas para os testes.

Light Field	Taxa Alvo (bpp)				
Todos os Lenslets	0.001	0.005	0.02	0.1	0.75
Greek e Sideboard	–	0.005	0.02	0.1	0.75
Tarot Cards	0.001	0.005	0.02	0.1	0.75

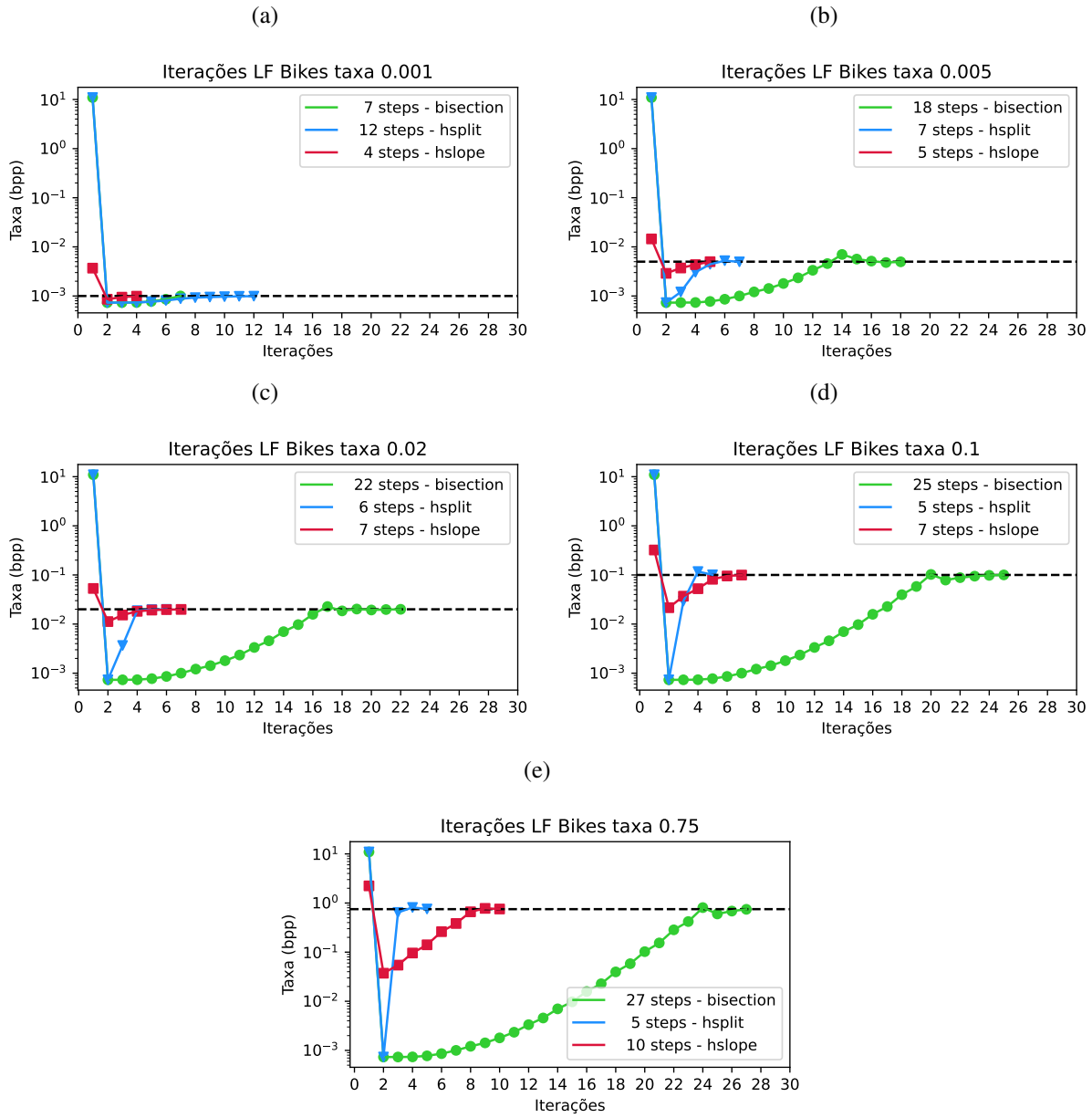
Fonte: o autor.

Uma consideração importante a se fazer é que a taxa 0.001 bpp definida nas CTC (Pereira et al., 2019), conforme listado na Tabela 2, não é compatível com os LFs Greek e Sideboard, e por isso, foi removida da busca. Isso acontece porque somente a sinalização necessária para cada bloco (*start of block marker*) já ocupa um tamanho maior do que o necessário para atingir a taxa, tornando a busca impossível.

4.2 NÚMERO DE ITERAÇÕES

A Figura 17 mostra uma comparação detalhada entre os passos de convergência de cada algoritmo para o controle de taxa do LF “Bikes”, considerando cada uma das taxas alvo. Já as tendências gerais dos algoritmos podem ser melhor observadas através da Figura 18, que ilustra o número máximo, mínimo e médio de iterações entre os LFs para alcançar cada taxa alvo.

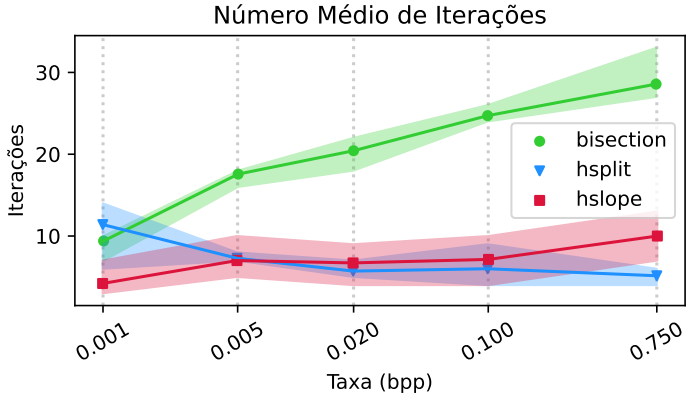
Figura 17 – Número de iterações necessárias para o controle de taxa do LF “Bikes”. Cada ponto representa o bpp alcançado em um passo de iteração. A reta tracejada preta representa a taxa alvo. Note que em todos os casos, *i.e.* todos os bpps, os dois primeiros passos dos algoritmos bisseção e H.Split são idênticos, tal como apresentado nos Algoritmos 1 e 2.



Fonte: o autor.

Os experimentos demonstraram que para a maior parte dos LFs, os algoritmos H. Split

Figura 18 – Número de iterações necessárias para o controle de taxa dos LFs testados. Cada ponto representa o bpp alcançado em um passo de iteração. A faixa semi transparente representa o intervalo que contém os números mínimos e máximos de iteração necessárias para cada taxa-alvo. As linhas pontilhadas cinza representam as taxas-alvo.



Fonte: o autor.

e H. Slope convergiram para a taxa-alvo em um número menor de iterações do que a bisseção. Houveram apenas algumas poucas exceções em que o algoritmo H. Split levou um número maior de iterações para convergir. Este fenômeno aconteceu somente para a taxa de 0.001 bpp, como pode ser observado na Figura 18.

Por outro lado, o algoritmo H. Slope convergiu mais rapidamente para as taxas mais baixas do que para as demais. Já para as taxas mais altas, especialmente 0.75 bpp, este algoritmo precisou de mais iterações para convergir. Porém, deve-se lembrar que mesmo levando mais iterações, o algoritmo H. Slope tem a vantagem de não necessitar a execução com um valor de λ muito baixo (mais lento). Isso pode ser visto claramente na Figura 17. O efeito prático disso será visto na análise do tempo de execução de cada algoritmo.

Nos demais casos, os algoritmos H. Split e H. Slope convergiram em uma quantidade semelhante de iterações, com uma pequena vantagem para o H. Split. Essa vantagem pode ser explicada pelo fato de que a função hiperbólica se adapta melhor à curva R-λ do que à curva de RD, como pode ser observado ao comparar as Figuras 13 e 15.

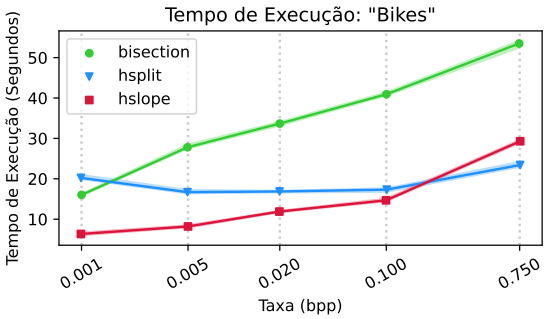
4.3 TEMPO DE EXECUÇÃO

A Figura 19 apresenta o tempo de execução mensurado para cada um dos algoritmos testados. Devido ao grande número de iterações, como visto na Figura 18, o algoritmo da bisseção também apresentou um desempenho pior do que os demais em tempo de execução. Por outro lado, apesar do H. Slope exigir um número maior de iterações para convergir, ele executou mais rapidamente do que o H. Split na maior parte dos testes. Isso se deve ao fato de que o próprio codificador exige um tempo maior de execução para λs menores, e o H. Slope realiza as iterações em uma região mais próxima à de interesse, enquanto o H. Split tem pelo menos uma iteração com λ pequeno (longo tempo de execução) para atender às exigências do

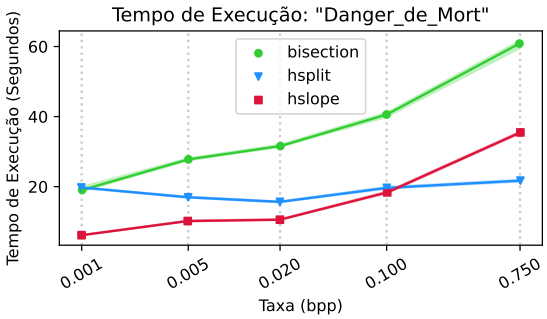
intervalo inicial.

Figura 19 – Tempo de execução para cada um dos algoritmos testados. Cada ponto representa o tempo médio de uma codificação para uma taxa específica. As linhas semi transparentes representam os valores mínimos e máximos de tempo registrados. Neste caso é difícil de observar esses valores mínimos e máximos pois houve pouca variação entre as repetições de um mesmo algoritmo.

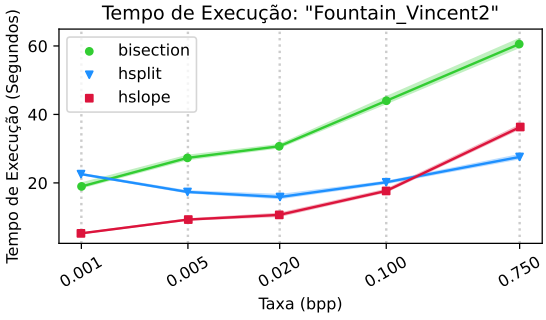
(a)



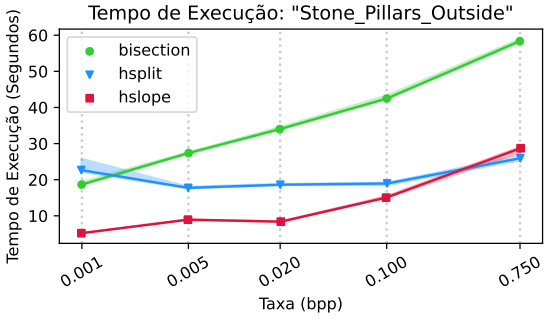
(b)



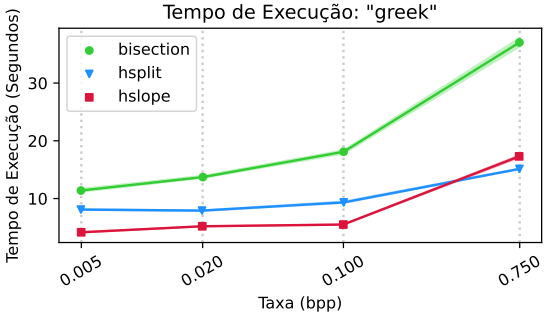
(c)



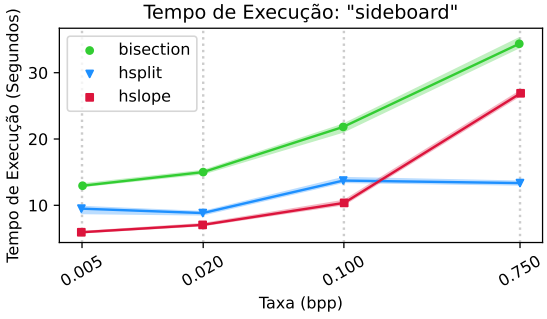
(d)



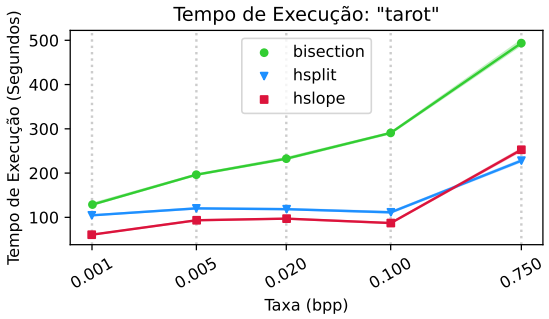
(e)



(f)



(g)



Fonte: o autor.

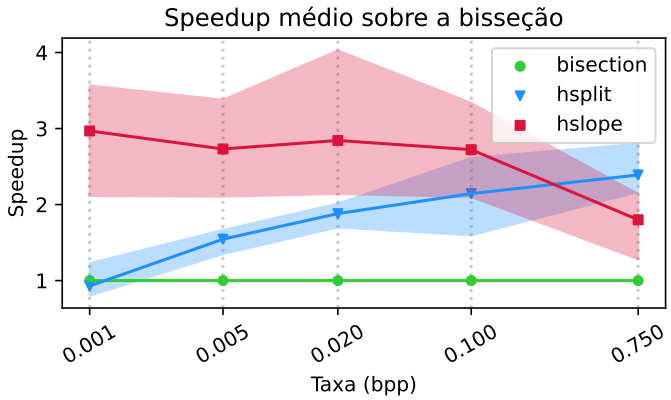
Esta diferença entre H. Split e H. Slope passa a ser menos significativa quando a taxa alvo é maior, pois isto implica na necessidade de usar λs menores e, consequentemente, um tempo maior de execução em todas as iterações. Esta situação faz com que o número reduzido de iterações do H. Split supere a otimização em que o H. Slope se baseia, e no caso de taxas maiores, o H. Split tende a ser um pouco mais eficiente.

A relação entre os algoritmos pode ser mais facilmente observada ao comparar os tempos em relação à bisseção. Assim, medimos a aceleração (*speedup*) dos demais algoritmos comparados à bisseção. O cálculo do *speedup* é feito através da Equação 4.1, usando a mediana dos tempos da bisseção e a mediana dos tempos do algoritmo que queremos comparar (Touati; Worms; Briais, 2013).

$$\text{speedup}_{\text{algoritmo}} = \frac{\text{mediana}(\text{tempos}_{\text{bisseção}})}{\text{mediana}(\text{tempos}_{\text{algoritmo}})} \tag{4.1}$$

A Figura 20 mostra o *speedup* para todos os LFs testados. Evidentemente, todos os valores da bisseção aparecem no gráfico iguais a 1, afinal este é o próprio algoritmo de referência. O valor dos demais algoritmos representa um fator de quantas vezes mais rápido (ou mais lento, no caso de valores negativos) ele performou em relação à bisseção.

Figura 20 – Speedup médio de cada algoritmo em relação à Bisseção. Cada ponto representa o *speedup* de uma codificação em relação ao algoritmo de Bisseção. A faixa semi transparente representa o valor máximo e mínimo de *speedup* entre os LFs testados.



Fonte: o autor.

Outra informação importante para analisarmos é o sobrecusto (*overhead*) que os algoritmos propostos impõem em relação ao tempo de codificação sem a funcionalidade de controle de taxa. Esta comparação é feita codificando os LFs para as taxas alvo utilizando λs previamente conhecidos. Note que este conhecimento prévio não é comum, tal que batizamos a execução com os valores de λ previamente conhecidos de "oráculo". Os λs utilizados, as taxas atingidas e os erros absolutos e relativos estão listados na Tabela 5. Tais valores foram utilizados no Core Experiment (CE) 12 do JPEG Pleno (Seidel; Fernandes, 2024a). Estes valores de λ foram encontrados através do algoritmo H. Slope com o limite máximo de 1% de erro.

Note que dentre os resultados obtidos, alguns deles possuem erros relativos maiores do que 1%. Isso acontece porque o erro neste contexto é calculado levando em consideração dados do cabeçalho que em muitos casos não estão disponíveis durante o processo de controle de taxa. Portanto, o controle de taxa é feito apenas na porção da *bitstream* dedicada à codificação do LF.

Tabela 5 – Valores de λ utilizados para a codificação do oráculo, taxas atingidas e erros obtidos.

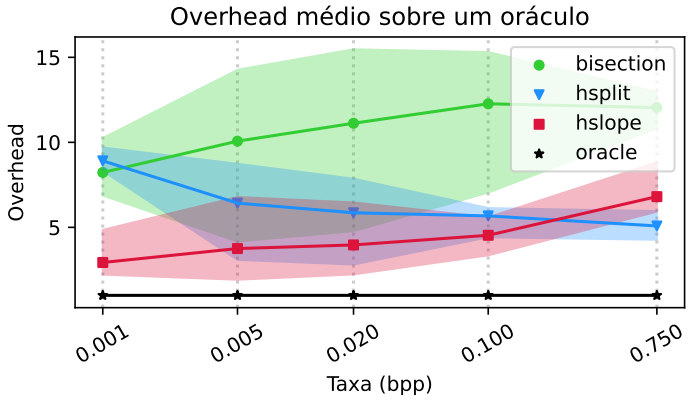
Light Field	**Lambda**	**Taxa alvo (bpp)**	**Taxa obtida (bpp)**	**Erro absoluto**	**Erro Relativo (%)**
Bikes	3.26894e+07	0.001	0.00102022	2.02165e-05	1.98159
Bikes	439381	0.005	0.00502622	2.62155e-05	0.521575
Bikes	39521.9	0.02	0.0198984	0.00010159	0.510542
Bikes	3971.19	0.1	0.0995997	0.000400273	0.401881
Bikes	263.593	0.75	0.756118	0.00611795	0.809126
Danger_de_Mort	4.44621e+07	0.001	0.0010272	2.71971e-05	2.6477
Danger_de_Mort	605676	0.005	0.00498939	1.06073e-05	0.212596
Danger_de_Mort	65547.3	0.02	0.0199846	1.53792e-05	0.076955
Danger_de_Mort	5864.83	0.1	0.100893	0.000893235	0.885327
Danger_de_Mort	364.985	0.75	0.7538	0.00380021	0.504141
Fountain_Vincent2	4.56958e+07	0.001	0.00102266	2.26597e-05	2.21576
Fountain_Vincent2	450320	0.005	0.00505728	5.72792e-05	1.13261
Fountain_Vincent2	53230.8	0.02	0.0198988	0.000101241	0.508779
Fountain_Vincent2	5161.16	0.1	0.0998514	0.000148622	0.148843
Fountain_Vincent2	321.708	0.75	0.757003	0.00700309	0.925107
Stone_Pillars_Outside	3.65863e+07	0.001	0.00102144	2.14381e-05	2.09882
Stone_Pillars_Outside	233486	0.005	0.00499288	7.11695e-06	0.142542
Stone_Pillars_Outside	27401.8	0.02	0.0202239	0.000223881	1.10701
Stone_Pillars_Outside	3092.27	0.1	0.0991114	0.000888567	0.896533
Stone_Pillars_Outside	242.102	0.75	0.756648	0.00664848	0.878674
greek	348438	0.005	0.00505311	5.31081e-05	1.051
greek	21271.7	0.02	0.0199822	1.7768e-05	0.0889191
greek	1321.16	0.1	0.0999556	4.43824e-05	0.0444021
greek	42.5601	0.75	0.745314	0.00468577	0.628697
sideboard	1.20262e+06	0.005	0.00502334	2.33441e-05	0.464712
sideboard	138657	0.02	0.0199299	7.01377e-05	0.351923
sideboard	11574.8	0.1	0.100391	0.000391152	0.389628
sideboard	546.693	0.75	0.754993	0.00499283	0.661308
tarot	1.19677e+07	0.001	0.000999794	2.06126e-07	0.0206168
tarot	774731	0.005	0.00502312	2.31247e-05	0.460365
tarot	96624	0.02	0.0200972	9.7198e-05	0.483639
tarot	6411.06	0.1	0.100221	0.000221363	0.220874
tarot	157.216	0.75	0.749955	4.509e-05	0.00601236

Fonte: (Seidel; Fernandes, 2024a)

O *overhead* no tempo de execução é computado como mostra a Equação 4.2, levando em consideração a mediana dos tempos do algoritmo testado em relação à mediana dos tempos de execução do oráculo.

$$\text{overhead}_{\text{algoritmo}} = \frac{\text{mediana}(\text{tempos}_{\text{algoritmo}})}{\text{mediana}(\text{tempos}_{\text{oráculo}})} \tag{4.2}$$

Figura 21 – *Overhead* médio do tempo de execução de cada algoritmo em relação a um oráculo. Cada ponto representa o *overhead* de uma codificação em relação a um oráculo. A faixa semi transparente representa o valor máximo e mínimo de *overhead* entre os LFs testados.



Fonte: o autor.

A Figura 21 apresenta o *overhead* médio em relação ao oráculo, considerando todos os LFs testados. Assim, pode-se observar que a codificação pode levar até 15× mais tempo para concluir ao fazer a busca pela taxa alvo (usando a Bisseção). Ainda, estes dados de *overhead* demonstram que apesar da eficiência dos algoritmos H. Split e H. Slope, eles ainda exigem uma quantidade de tempo considerável em relação a uma execução sem a busca por taxa.

Portanto, a passagem direta do multiplicado Lagrangiano pelo usuário continua sendo um recurso útil para os casos em que o valor de λ é conhecido com antecedência, pois a codificação é mais rápida quando uma busca não se faz necessária. Por este motivo, algoritmos de controle de taxa implementados no codificador apresentam ao usuário qual foi o valor de λ encontrado em cada busca, para que ele possa ser reutilizado de maneira rápida em outras codificações. Note que sem os algoritmos de controle de taxa, o usuário dificilmente conheceria o valor de λ com antecedência.

5 CONCLUSÕES

Neste trabalho foram criados três algoritmos para controle de taxa de LFs no padrão JPEG Pleno utilizando o 4DTM. O primeiro deles foi uma simples adaptação do método de bisseção. Os outros algoritmos, H. Split e H. Slope, são inéditos e se baseiam, respectivamente, na relação entre as curvas de R-λ e RD com uma função hiperbólica. Foi demonstrado que os algoritmos são capazes de atingir a taxa-alvo com erro de no máximo 1% (considerando apenas a *bitstream* do LF, sem cabeçalhos ou informações extra), sem perda de qualidade.

Os algoritmos novos obtiveram um tempo de execução significativamente menor do que o método da bisseção, alcançando resultados até 4 vezes mais rápidos. Apesar dos resultados promissores, o tempo de execução é relativamente longo em relação a um oráculo (conhecimento prévio do valor de λ para atingir a taxa-alvo). Isso implica que a passagem direta de um multiplicador Lagrangiano pelo usuário continua sendo um recurso útil, desde que este seja conhecido com antecedência. Note que normalmente, em situações de captura ou avaliação de novas ferramentas e configurações de codificação, este não é o caso.

Ainda, durante o desenvolvimento deste TCC, os métodos implementados foram usados para a definição de valores de λ para as taxas-alvo das CTC nos CEs 12 e 13 do JPEG Pleno Light Field (Seidel; Fernandes, 2024a; Seidel; Fernandes, 2024b). Além disso, tendo em vista o longo tempo de execução do JPLM, foi criada uma versão paralela do mesmo (Seidel; Fernandes; Güntzel, 2024), o que permitiu a experimentação necessária.

Em geral, os objetivos citados na Seção 1.1 foram alcançados. Como foi citado, os algoritmos atingem uma taxa-alvo dentro de um limite de precisão. Não houve perda de eficiência de codificação em relação à versão original do JPLM, uma vez que se manteve o mesmo valor de λ para todos os blocos 4D. Por outro lado, o tempo de codificação ainda é alto, pois apesar da eficiência dos algoritmos, a busca representa uma parte considerável do tempo de execução do software.

5.1 TRABALHOS FUTUROS

A análise dos algoritmos criados sugere uma possibilidade de exploração de novas técnicas que combinem a eficiência do H. Slope para taxas menores, mas que, assim como o H. Split, funcione bem para taxas maiores. Existem ainda opções para melhorar o H. Slope que não foram avaliadas, como a utilização de valores iniciais mais adequados com base em características do LF, ou o uso de estratégias diferentes para a atualização do fator de convergência.

Finalmente, como os algoritmos estão implementados no JPLM, pretendemos enviar uma contribuição ao JPEG descrevendo os algoritmos para uma possível integração ao código oficial público e para que faça parte da próxima edição do padrão JPEG Pleno Part 4 (ISO/IEC 21794-4) (ISO Central Secretary, 2022).

REFERÊNCIAS

ADELSON, E. H.; BERGEN, J. R. The plenoptic function and the elements of early vision. In: **Computational Models of Visual Processing**. [S.l.]: MIT Press, 1991. p. 3–20.

AHMAD, I. et al. An overview of rate control techniques in HEVC and SHVC video encoding. **Multimedia Tools and Applications**, Springer, v. 81, n. 24, p. 34919–34950, 2022.

AHMED, N.; NATARAJAN, T.; RAO, K. Discrete cosine transform. **IEEE Transactions on Computers**, C-23, n. 1, p. 90–93, 1974.

ALVES, G. D. O. et al. The JPEG Pleno light field coding standard 4D-transform mode: How to design an efficient 4D-native codec. **IEEE Access**, v. 8, p. 170807–170829, 2020.

AMOR, M. Á. M. del; BRUNS, V.; SPARENBERG, H. Parallel efficient rate control methods for JPEG 2000. In: TESCHER, A. G. (Ed.). **Applications of Digital Image Processing XL**. SPIE, 2017. v. 10396, p. 103960R. Disponível em: https://doi.org/10.1117/12.2273005.

ASTOLA, P. et al. JPEG Pleno: Standardizing a coding framework and tools for plenoptic imaging modalities. **ITU Journal: ICT Discoveries**, v. 3, 06 2020. Disponível em: http://handle.itu.int/11.1002/pub/8153d79a-en.

ASTOLA, P.; TABUS, I. WaSP: Hierarchical warping, merging, and sparse prediction for light field image compression. In: **2018 7th European Workshop on Visual Information Processing (EUVIP)**. Tampere, Finland: [s.n.], 2018. p. 1–6.

BALSTER, E. J.; FORTENER, B. T.; TURRI, W. F. Rate-distortion development for optimal truncation in JPEG2000 imagery. In: **2010 4th International Symposium on Communications, Control and Signal Processing (ISCCSP)**. [S.l.: s.n.], 2010. p. 1–6.

BURDEN, R. L.; FAIRES, J. D.; BURDEN, A. M. Numerical analysis. In: ____. Tenth edition. United States of America: Cengage Learning, 2015. cap. 2.1 The Bisection Method, p. 48–54.

CHAN, D.; LIANG, J.; TU, C. ρ-domain rate control for JPEG XR. In: **2010 Conference Record of the Forty Fourth Asilomar Conference on Signals, Systems and Computers**. [S.l.: s.n.], 2010. p. 226–230.

CHOI, H. et al. Pixel-wise unified rate-quantization model for multi-level rate control. **IEEE Journal of Selected Topics in Signal Processing**, IEEE, v. 7, n. 6, p. 1112–1123, 2013.

CHOU, P.; MIAO, Z. Rate-distortion optimized streaming of packetized media. **IEEE Transactions on Multimedia**, v. 8, 04 2001.

CONCEIÇÃO, R. A. da. **A Survey of Light-field Coding: Concepts and State-of-the-art Literature Review**. 2017. Exame de qualificação de doutorado.

CONTE, S.; BOOR, C. D. **Elementary Numerical Analysis: An Algorithmic Approach**. [S.l.]: McGraw-Hill, 1980. (International series in pure and applied mathematics). ISBN 9780070124479.

EBRAHIMI, T. et al. Jpeg pleno: Toward an efficient representation of visual reality. **Ieee Multimedia**, IEEE, v. 23, n. 4, p. 14–20, 2016.

HE, Z.; KIM, Y. K.; MITRA, S. K. Low-delay rate control for dct video coding via/spl rho/-domain source modeling. **IEEE transactions on Circuits and Systems for Video Technology**, IEEE, v. 11, n. 8, p. 928–940, 2001.

INTEL. **Intel 64 and IA-32 Architectures Software Developer's Manual**. [S.l.], 2024.

ISO Central Secretary. **Information technology – JPEG 2000 image coding system – Part 1: C ore coding system**. Geneva, CH, 2019.

ISO Central Secretary. **Information technology — Plenoptic image coding system (JPEG Pleno) — Part 1: Framework**. Geneva, CH, 2020.

ISO Central Secretary. **Information technology — Plenoptic image coding system (JPEG Pleno) — Part 2: Light field coding**. Geneva, CH, 2021.

ISO Central Secretary. **Information technology — Plenoptic image coding system (JPEG Pleno) — Part 3: Conformance testing**. Geneva, CH, 2021.

ISO Central Secretary. **Information technology — Plenoptic image coding system (JPEG Pleno) — Part 4: Reference software**. Geneva, CH, 2022.

ISO Central Secretary. **Information technology — Plenoptic image coding system (JPEG Pleno) — Part 5: Holography**. Geneva, CH, 2023.

ISO Central Secretary. **Information technology — Plenoptic image coding system (JPEG Pleno) — Part 6: Learning-based point cloud coding**. Geneva, CH, 2023.

ITU-T. **Parameter values for the HDTV standards for production and international programme exchange**. Genebra, 2008.

LEVOY, M. et al. Light field microscopy. In: **ACM SIGGRAPH 2006 Papers**. New York, NY, USA: Association for Computing Machinery, 2006. (SIGGRAPH '06), p. 924–934. ISBN 1595933646. Disponível em: https://doi.org/10.1145/1179352.1141976.

LI, B. et al. λ domain rate control algorithm for high efficiency video coding. **IEEE Transactions on Image Processing**, v. 23, n. 9, p. 3841–3854, 2014.

MA, S.; GAO, W.; LU, Y. Rate-distortion analysis for H.264/AVC video coding and its application to rate control. **IEEE transactions on circuits and systems for video technology**, IEEE, v. 15, n. 12, p. 1533–1544, 2005.

MILANI, S.; CELETTO, L.; MIAN, G. A. An accurate low-complexity rate control algorithm based on (ρ, E_q)-domain. **IEEE transactions on circuits and systems for video technology**, IEEE, v. 18, n. 2, p. 257–262, 2008.

NG, R. et al. **Light Field Photography with a Hand-held Plenoptic Camera**. [S.l.], 2005. Stanford University Computer Science Tech Report p. Disponível em: https://hal.science/hal-02551481.

OH, P. S. K.-J. **JPEG Pleno Holography Uses Cases and Requirements**. 2019. Doc. ISO/IEC JTC 1/SC 29/WG1 (ITU-T SG16), 88th JPEG Meeting, Online. Disponível em: https://jpeg.org/jpegpleno/documentation.html.

ORTEGA, A.; RAMCHANDRAN, K. Rate-distortion methods for image and video compression. **IEEE Signal Processing Magazine**, v. 15, n. 6, p. 23–50, 1998.

PEREIRA, F. et al. **JPEG Pleno Light Field Coding Common Test Conditions V3.3**. 2019. Doc. ISO/IEC JTC 1/SC 29/WG1 N84049, 84th JPEG Meeting, Brussels, Belgium. Disponível em: https://jpeg.org/jpegpleno/documentation.html.

PERRA, C. et al. An overview of the emerging JPEG Pleno standard, conformance testing and reference software. In: SCHELKENS, P.; KOZACKI, T. (Ed.). **Optics, Photonics and Digital Technologies for Imaging Applications VI**. SPIE, 2020. v. 11353, p. 207 – 219. Disponível em: https://doi.org/10.1117/12.2555841.

PERRY, S. **JPEG Pleno Point Cloud Use Cases and Requirements, v1.6**. 2019. Doc. ISO/IEC JTC 1/SC 29/WG1 (ITU-T SG16), 88th JPEG Meeting, Online. Disponível em: https://jpeg.org/jpegpleno/documentation.html.

PERWASS, C.; WIETZKE, L. Single lens 3D-camera with extended depth-of-field. In: ROGOWITZ, B. E.; PAPPAS, T. N.; RIDDER, H. de (Ed.). **Human Vision and Electronic Imaging XVII**. SPIE, 2012. v. 8291, p. 829108. Disponível em: https://doi.org/10.1117/12.909882.

RAYTRIX. **3D light field technology**. 2015. [Online; acessado em 27 de Junho de 2023]. Disponível em: https://raytrix.de/technology-2/.

RAYTRIX. **3D light field technology**. 2015. [Online; acessado em 27 de Junho de 2023]. Disponível em: https://raytrix.de/applications/.

RAYTRIX. **2D-Camera-Shop**. 2019. [Online; acessado em 27 de Junho de 2023]. Disponível em: https://raytrix.de/c42/.

SEIDEL, I.; FERNANDES, A. F. da S. **Core Experiment 12 on JPEG Pleno Light Field report by UFSC**. Online, 2024.

SEIDEL, I.; FERNANDES, A. F. da S. **Core Experiment 13 on JPEG Pleno Light Field report by UFSC**. Sapporo, Japan, 2024.

SEIDEL, I.; FERNANDES, A. F. da S.; GüNTZEL, J. L. A parallel JPEG pleno baseline block-based profile light field encoder using OpenMP. In: **26ª Escola de Microeletrônica e 39º Simpósio Sul de Microeletrônica**. [S.l.: s.n.], 2024.

SULLIVAN, G.; WIEGAND, T. Rate-distortion optimization for video compression. **IEEE Signal Processing Magazine**, v. 15, n. 6, p. 74–90, 1998.

TOUATI, S.; WORMS, J.; BRIAIS, S. The Speedup-Test: A Statistical Methodology for Program Speedup Analysis and Computation. **Concurrency and Computation: Practice and Experience**, Wiley, v. 25, n. 10, p. 1410–1426, 2013. Article first published online: 15 OCT 2012. Disponível em: https://inria.hal.science/hal-00764454.

TURNER, C. S. A fast binary logarithm algorithm [dsp tips & tricks]. **IEEE Signal Processing Magazine**, IEEE, v. 27, n. 5, p. 124–140, 2010.

WILBURN, B. et al. High performance imaging using large camera arrays. **ACM Trans. Graph.**, Association for Computing Machinery, New York, NY, USA, v. 24, n. 3, p. 765–776, jul 2005. ISSN 0730-0301. Disponível em: https://doi.org/10.1145/1073204.1073259.

WILBURN, B. et al. High performance imaging using large camera arrays. **ACM Trans. Graph.**, Association for Computing Machinery, New York, NY, USA, v. 24, n. 3, p. 765–776, jul 2005. ISSN 0730-0301. Disponível em: https://doi.org/10.1145/1073204.1073259.

YANG, H. et al. A novel rate control scheme for video coding in HEVC-SCC. **IEEE Transactions on Broadcasting**, IEEE, v. 66, n. 2, p. 333–345, 2019.

ANEXO A – CÓDIGO FONTE

O código fonte desenvolvido para este trabalho, juntamente com os dados obtidos na pesquisa, encontram-se publicamente disponíveis no repositório do autor no GitHub, acessível através do seguinte link https://github.com/andrefpf/codigo-tcc.

Os experimentos foram disponibilizados através de Jupyter Notebooks, enquanto as modificações no software de referência foram disponibilizadas através de um arquivo diff, contendo apenas o que foi implementado neste trabalho.

ANEXO B – ARTIGO

Rate-control methods for the JPEG Pleno 4D Transform Mode

André Filipe da Silva Fernandes, Ismael Seidel*
Embedded Computing Lab. (ECL),
*Computer Science Graduate Program (PPGCC),
Dept. of Computer Science and Statistics (INE), Federal University of Santa Catarina (UFSC), Florianópolis, Brazil
andre.filipe.fernandes@grad.ufsc.br, ismael.seidel@ufsc.br

***Abstract*—A major concern in lossy data compression algorithms is balancing the amount of information that can be discarded and its impact on data quality. The codification of light fields in the *4D Transform Mode* (4DTM), according to the JPEG Pleno Standard, solves this problem using a Lagrangian multiplier. This technique works by selecting the encoder options in such a way that minimized a cost, defined by the distortion, the rate, and a user-defined lambda (λ) parameter. Although this is a simple and effective solution, it is hard for a user to encode a light field aiming for specific rates, like it is requested by the standard Common Test Conditions (CTC). This work describes two new algorithms to efficiently find a lambda parameter such that the rate of the resulting light field matches a user defined target. These algorithms were named Hyperbolic Split and Hyperbolic Slope, and they can find the target rates up to 4 times faster than the bisection.**

***Index Terms*—Light fields. JPEG Pleno. Rate control.**

I. Introduction

Light Field (LF) is an image technology capable of representing static scenes in a very immersive way. It can represent accurately stereo parallax, motion parallax, reflection, refraction, and volumetric effects for real-world scenes [1]. To achieve such a high quality depiction, LFs represent the space as a 4 dimensional matrix of pixels, whose dimensions are usually named (t, s, v, u) [2]. The dimensions t and s represents respectively the vertical and the horizontal coordinates of the image plane. The dimensions v and u represents the angle of the incident light in the image plane both in the vertical and horizontal direction.

Because a LF is a 4-Dimensions (4D) matrix, it can be interpreted as a 2-Dimensions (2D) matrix of views, that are 2D images captured from slightly different angles.

The encoding of LFs in the Joint Photographic Experts Group (JPEG) Pleno Standards can be done in two modes [3]:

1) *4D Prediction Mode* (4DPM), a model that predicts some views based on the surrounding information and encodes the differences.
2) *4D Transform Mode* (4DTM), a model based on a multi level *4 Dimensional Discrete Cosine Transform* (4D-DCT).

The 4DPM works best when the distance between the views is higher, like when it is captured using a High Density Camera Array (HDCA) [1], [2], [4]. On the other hand, the 4DTM works best when the views are closer, a common characteristic of LFs captured with lenslet cameras. This paper focuses on providing a rate control method specifically made for the 4DTM.

The 4DTM can be understood better if divided in a few steps, that are represented in 1.

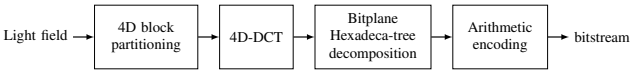


Fig. 1. Simplified flow of the 4DTM encoding process.

First, the LF is divided in equal sized blocks. These blocks can be further partitioned recursively in a configurable range of block sizes, both in the spatial (t, s) and the views (v, u) dimensions. This partitioning is useful to separate different textures for the following step, the Discrete Cosine Transform (DCT).

The DCT is executed over the subdivided 4d blocks, and represents the content as a set of frequency coefficients. This procedure accumulates most of the relevant information in the first coefficients, therefore the last coefficients can be quantized to improve compression with lower impact in the quality.

The quantization process is done using bit planes. A set of flags are used to store these bit planes exploring the fact that the DCT transformed blocks are usually sparse, and the absolute value of a few coefficients are much larger than others.

In order to take every decision in these steps, a Rate-Distortion Optimization (RDO) is conducted. A cost J is defined as shown in (1), where D is the distortion, R is the rate of compression and λ is a parameter to control this balance.

$$J = D + \lambda R \tag{1}$$

After the cost is calculated, the optimization process consists in selecting the minimal J between all the possible options.

No works were found targeting the rate control problem for LF codecs. Although, there are various methods that were proposed for other medias, such as image and video [5]–[11]. The work of Li *et al.* [11] proposes a method suited to the High Efficiency Video Coding (HEVC) standard and is based on a analysis in the λ domain. Although the algorithms they proposed can not be adapted to be used in the JPEG Pleno

standard, the idea to use a hyperbolic model that fits the Rate-Distortion (RD) or the R-λ curves is still very useful. The hyperbolic model is defined on (2).

$$f(x) = \alpha * x^{\beta} \tag{2}$$

Figures 2 and 3 shows that the hyperbolic model fits the RD and the R-λ curves very tightly in LFs encoded using JPEG Pleno Model (JPLM). The figures are showing data from the Bikes LF, but it also works well for other LFs available in the Common Test Conditions (CTC).

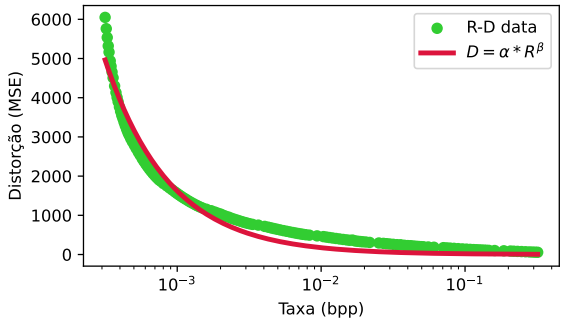


Fig. 2. RD curve fitting using the hyperbolic model for the Bikes LF

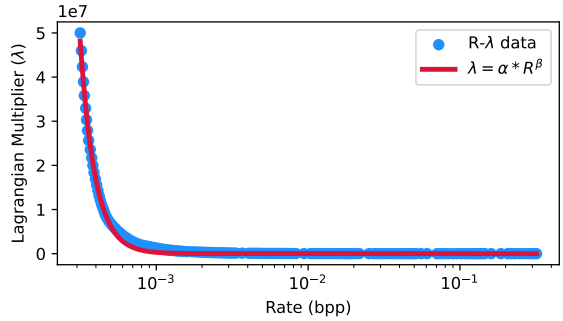


Fig. 3. R-λ curve fitting using the hyperbolic model for the Bikes LF

The main contributions of this paper are the proposal and evaluation of two new methods to control efficiently the rate of LF codecs that follows JPEG Pleno standard. These methods can find targets, with maximum error of 1% of the target rate, in up to 4 times faster than the algorithm of bisection.

This paper is organized as follows. Section II presents new algorithms and describes how their equations were derived. Section III explains the experimental setup. Section IV shows the comparative results between the two new algorithms and the bisection. Finally, in section V, we present the conclusions and the next steps of this research.

II. Proposal

To find an effective rate control algorithm, we can treat the encoder as a function $E : \mathbb{R} \rightarrow \mathbb{R}^2$, that maps a Lagrangian multiplier (λ) to a pair of rate (R) and distortion (D).

In this context, the rate control problem can be defined as the effort to find a value λ^{target} such that $E(\lambda^{\text{target}}) = (R_{\text{i}}, D_{\text{i}})$ and $|R_{\text{i}} - R^{\text{target}}| \leq$ precision, where R^{target} is the rate of the output file in terms of bits per pixel.

A. Bisection

A common starting point for this kind of problem is the bisection algorithm. The main idea of the bisection is to start with two values that define an interval such that the expected result lies between them. The algorithm then proceeds by repeatedly halving the interval and updating it, so that the expected target remains inside it. The algorithm stops when the exact result, or a good approximation, is found.

Applied to this problem, the algorithm starts with λ_0^{start}, the start of the interval in the first iteration, and λ_0^{end}, the end of the interval in the first iteration. It is expected that λ^{target} relies in this interval, otherwise the algorithm will not work.

In every iteration, the interval parameters are updated according to (4). The algorithm stops when R_n^{start} or R_n^{end} is close enough to R^{target} or if it passed the maximum number of iterations.

$$\lambda_n^{\text{half}} = \frac{1}{2} * (\lambda_n^{\text{start}} + \lambda_n^{\text{end}}) \tag{3}$$

$$(\lambda_{n+1}^{\text{start}}, \lambda_{n+1}^{\text{end}}) = \begin{cases} (\lambda_n^{\text{half}}, \lambda_n^{\text{end}}) & \text{if } R_n^{\text{half}} > R^{\text{target}} \\ (\lambda_n^{\text{start}}, \lambda_n^{\text{half}}) & \text{otherwise} \end{cases} \tag{4}$$

This approach works best for linearly distributed data, which is not how the λ's are distributed in this application.

B. Hyperbolic Split Algorithm

To take advantage of the hyperbolic model inside JPLM, it is possible to make the bisection algorithm much better by improving the interval splitting. Instead of splitting the interval in the middle, this new algorithm uses two previously computed values of R and λ to find a better approximation for λ^{target}, and then splits the interval in this point. This subtle change is enough to make the values converge much faster.

To find a good approximation for λ^{target}, based on the hyperbolic model, we define a function λ^{hsplit} as shown in (5).

$$\lambda(R) \approx \lambda^{\text{hsplit}}(R) := \alpha * R^{\beta} \tag{5}$$

Then we find α_n and β_n, in such a way that $\lambda^{\text{hsplit}}(R_n^{\text{start}}) = \lambda_n^{\text{start}}$ and $\lambda^{\text{hsplit}}(R_n^{\text{end}}) = \lambda_n^{\text{end}}$. These requirements lead to the following equations that need to be satisfied for every iteration of the algorithm:

$$\lambda_n^{\text{start}} = \alpha_n * (R_n^{\text{start}})^{\beta_n} \tag{6}$$

$$\lambda_n^{\text{end}} = \alpha_n * (R_n^{\text{end}})^{\beta_n} \tag{7}$$

Every term, except for α and β are known (the λ's were used to obtain the rates in previous codifications). First we will find β. To do it we can divide one equation by the other to eliminate α_n.

$$\frac{\lambda_n^{\text{start}}}{\lambda_n^{\text{end}}} = \frac{\cancel{\alpha_n}}{\cancel{\alpha_n}} * \left(\frac{R_n^{\text{start}}}{R_n^{\text{end}}}\right)^{\beta_n} \tag{8}$$

To isolate the β_n term, that is in the exponent of the equation, we take the logarithm on both sides of (8) to use the property of logarithms that $log(a^b) = b * log(a)$. This way β_n can be isolated, as shown in (9), (10) and (11). The base of the logarithm is not important in this case, as long as it is the same in both sides of the equation. During the algorithm implementations in JPLM, the base 2 was chosen.

$$log\left(\frac{\lambda_n^{\text{start}}}{\lambda_n^{\text{end}}}\right) = log\left(\left(\frac{R_n^{\text{start}}}{R_n^{\text{end}}}\right)^{\beta_n}\right) \tag{9}$$

$$log\left(\frac{\lambda_n^{\text{start}}}{\lambda_n^{\text{end}}}\right) = \beta_n * log\left(\frac{R_n^{\text{start}}}{R_n^{\text{end}}}\right) \tag{10}$$

$$\beta_n = \frac{log\left(\lambda_n^{\text{start}}\right) - log\left(\lambda_n^{\text{end}}\right)}{log(R_n^{\text{start}}) - log(R_n^{\text{end}})} \tag{11}$$

Using the value found for β_n in (11) and the definition made in (6), the parameter α_n can be obtained as shown in (12).

$$\alpha_n = \lambda_n^{\text{start}} * (R_n^{\text{start}})^{-\beta_n} \tag{12}$$

This way, the interval of lambdas will be updated as described in (13) and (14).

$$\lambda_n^{\text{half}} = \alpha_n * (R^{\text{target}})^{\beta_n} \tag{13}$$

$$(\lambda_{n+1}^{\text{start}}, \lambda_{n+1}^{\text{end}}) = \begin{cases} (\lambda_n^{\text{half}}, \lambda_n^{\text{end}}) & \text{if } R_n^{\text{half}} > R^{\text{target}} \\ (\lambda_n^{\text{start}}, \lambda_n^{\text{half}}) & \text{otherwise} \end{cases} \tag{14}$$

We expect this algorithm to converge faster than the bisection and thus significantly reduce the computational cost of the search.

C. Hyperbolic Slope Algorithm

Although the Hyperbolic Split Algorithm (H. Split) performed well for its purpose, it still has a major issue that can be addressed: The two initial codifications. Because the answer always needs to lie within the interval, at least one of the initial values needs to be very small in the first iteration of the algorithm. The problem is that a codification with a small lambda usually takes much more time to run than one with a large lambda.

To solve this problem, a new algorithm named Hyperbolic Slope Algorithm (H. Slope) was developed. This algorithm explores the tight fit of the hyperbolic model with the R-D curve, as shown in Fig. 2. Another characteristic explored is the fact that the lambda parameter is the negative slope of the distortion as a function of the rate [11], [12] Which can be expressed as in (15).

$$\lambda = -\frac{\mathrm{d}}{\mathrm{d}R} D(R) \tag{15}$$

Using the hyperbolic approximation, the distortion as a function of the rate can be approximated by the function $D^{\text{hslope}}(R)$, as presented in (16).

$$D(R) \approx D^{\text{hslope}}(R) \coloneqq \alpha * R^{\beta} \tag{16}$$

Because of the relation expressed in (15), the lambda approximation can be defined as the negative of the slope of (16), as shown in (17) and (18).

$$\lambda(R) \approx \lambda^{\text{hslope}}(R) \coloneqq -\frac{\mathrm{d}}{\mathrm{d}R} D^{\text{hslope}}(R) \tag{17}$$

$$\lambda^{\text{hslope}}(R) \coloneqq -\frac{\mathrm{d}}{\mathrm{d}R} \alpha * R^{\beta} \tag{18}$$

Therefore, calculating the derivative, the function is defined only in terms of α and β as shown in the Equation 19.

$$\lambda^{\text{hslope}}(R) \coloneqq -\alpha * \beta * R^{\beta - 1} \tag{19}$$

Using these relations, the whole behavior of both the R-D curve and the R-λ curves can be modeled together, by finding values α_n and β_n, in such a way that $D^{\text{hslope}}(R_n) = D_n$ and $\lambda^{\text{hslope}}(R_n) = \lambda_n$. Where D_n, R_n, and λ_n are the distortion and rate of the LF encoded by a certain lambda in the n$^{\text{th}}$ iteration, respectively.

Another particularity of this approach, is that we do not need an interval to be narrowed, a single value is constantly updated until the target is found. To find the needed parameters, it is useful to rewrite the refereed equations as follows:

$$D_n = \alpha_n * (R_n)^{\beta_n} \tag{20}$$

$$\lambda_n = -\alpha_n * \beta_n * (R_n)^{\beta_n - 1} \tag{21}$$

$$\tag{22}$$

The exponent (-1) can be removed by dividing the right side of the equation by R_n. This is useful because the term $\alpha_n (R_n)^{\beta_n}$ is now detached, as shown in (23), and it is precisely the definition of $D^{\text{hslope}}(R)$. Thus, we can replace it by D_n, getting rid of the unknown term α, as show in Equation 24.

$$\lambda_n = -\frac{\beta_n}{R_n} * \alpha_n (R_n)^{\beta_n} \tag{23}$$

$$\lambda_n = -\frac{\beta_n}{R_n} * D_n \tag{24}$$

This way, using (24), the β_n value can be easily isolated. Given a triple (λ_n, R_n, D_n) of known values, β_n is obtained as shown in (25).

$$\beta_n = -\frac{\lambda_n * R_n}{D_n} \tag{25}$$

On the other hand, the α_n value is obtained similarly as in the H. Split, using the obtained β_n value, as demonstrated in (26).

$$\alpha_n = D_n * (R_n)^{-\beta_n} \tag{26}$$

Using these calculations, a good estimative of lambda to be used is obtained in the following way:

$$\lambda_{n+1}^{\text{est}} = -\alpha_n * \beta_n * (R_n)^{\beta_n - 1} \tag{27}$$

To improve the convergence for rates very close to the target, it is useful to use a convergence factor (f) that starts as 1, and decreases by half every time the algorithm overshoots the target, as expressed in (28). The use of the convergence factor is demonstrated in (29).

$$f_{n+1} = \begin{cases} f_n/2 & \text{if } (R_n < R^{\text{target}} < R_{n-1}) \\ & \text{or } (R_n > R^{\text{target}} > R_{n-1}) \\ f_n & \text{otherwise} \end{cases} \tag{28}$$

$$\lambda_{n+1} = \lambda_{n+1}^{\text{est}} * f_n + \lambda_n * (1 - f_n) \tag{29}$$

III. Method

All referred algorithms (Bisection, H. Split and H. Slope) were implemented inside the JPLM Reference software. Both the rate and the distortion used by these algorithms were available approximations already used for the RDO procedure inside the encoder.

The experiments were made using python scripts to call the JPLM encoder with the required parameters, and every time measure was repeated 10 times and the median value was used to minimize biases. All experiments used a computer equipped with a 16-core, 32-threads AMD Ryzen™ 9 7950X@4.5GHz CPU, with 64MB, 16MB, and 1MB total cache sizes for level 3, level 2, and level 1, respectively. The machine uses four 32GB (total 128GB) of dual-channel DDR5 DRAM@4000MHz. The input LFs and generated output JPEG Pleno file format (JPL) were stored in a Kingston KC3000 PCIe 4.0 NVMe M.2 SSD 2TB.

Both algorithms were executed for the LFs present in the CTC [13], and aimed the target bit rates requested in the same document and represented in Table I.

TABLE I
TARGET RATES USED IN THIS WORK, BASED ON THE CTC [13].

Dataset	Target Bitrate (bpp)				
All Lenslets	0.001	0.005	0.02	0.1	0.75
Greek and Sideboard	–	0.005	0.02	0.1	0.75
Tarot Cards	0.001	0.005	0.02	0.1	0.75

IV. Results

Figure 4 shows the average number of codifications needed by each algorithm to find the target rate for every LF in the selected subset of the CTC. The figure show that the bisection algorithm needs much more codifications to converge than the other two algorithms, specially for higher target rates. The algorithms H. Split and H. Slope took about the same number of iterations to converge.

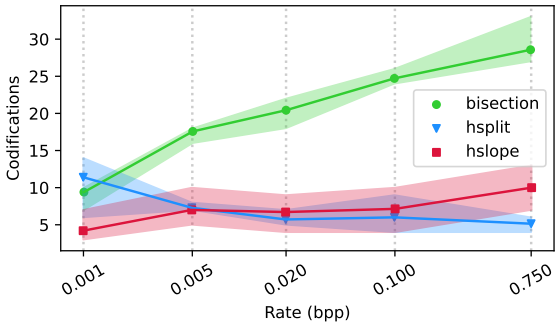


Fig. 4. Average number of codifications needed for every algorithm to find the target rates for the LFs selected.

The Figure 5 compares how the algorithms H. Split and H. Slope performed against the bisection in terms of execution time. According to the figure, H. Slope the fastest for most target rates, achieving target rates up to 4 times faster than bisection. Even though the number of iterations needed by H. Split and H. Slope were about the same, the execution time shows that H. Slope has a clear advantage over the other algorithms for most target rates.

This behaviour can be explained because H. Slope does not require codifications with small λ's (that would require more time to encode), as happens with the initial codification of the other two algorithms. This also explains why the performance of this algorithm decreases as the target rate increases, because higher targets needs lower λ's, witch take longer to encode.

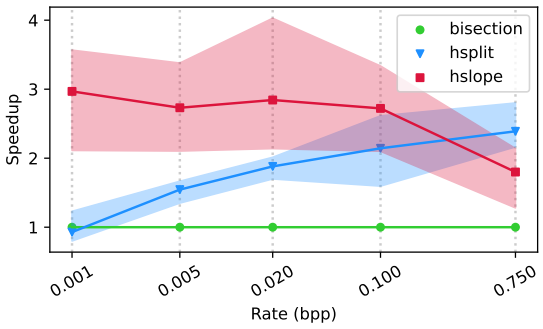


Fig. 5. Average speedup for the implemented algorithms in relation to the bisection, to find the target rates for the LFs selected.

V. Conclusions

This paper proposed and analyzed three algorithms to perform rate control for the JPEG Pleno standard. The algorithms can find the target rates with maximum error of 1% (considering only the bitstream of the LF), and without any losses in coding efficiency. The first of them is a simple implementation of the bisection method, while the other two, named H. Split and H. Slope, are new and were based on the tight fit of the hyperbolic model with the RD and R-λ curves, respectively.

The most promising algorithm is the H. Split, which performed netter for most target rates in every LF. H. Split performed up to 4 times faster than the bisection. The only

concern was for the target rate of 0.75 bpp, were the H. Split performed better.

These algorithms were already implemented inside the JPLM, and will be proposed as a contribution to the JPEG Pleno committee. The intention is to make the proposed implementation part of a new version of JPLM and include it in a new edition of ISO/IEC 21794-4 [14].

REFERENCES

[1] R. S. Overbeck *et al.*, "A system for acquiring, processing, and rendering panoramic light field stills for virtual reality," vol. 37, no. 6, Dec. 2018. [Online]. Available: https://doi.org/10.1145/3272127.3275031.

[2] G. De Oliveira Alves *et al.*, "The JPEG Pleno light field coding standard 4D-transform mode: How to design an efficient 4D-native codec," *IEEE Access*, vol. 8, pp. 170 807–170 829, 2020.

[3] ISO Central Secretary, "Information technology — Plenoptic image coding system (JPEG Pleno) — Part 2: Light field coding," en, International Organization for Standardization, Geneva, CH, Standard ISO/IEC 21794-2:2021, 2021.

[4] B. Wilburn *et al.*, "High performance imaging using large camera arrays," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 765–776, Jul. 2005. [Online]. Available: https://doi.org/10.1145/1073204.1073259.

[5] I. Ahmad *et al.*, "An overview of rate control techniques in HEVC and SHVC video encoding," *Multimedia Tools and Applications*, vol. 81, no. 24, pp. 34 919–34 950, 2022.

[6] S. Ma *et al.*, "Rate-distortion analysis for H.264/AVC video coding and its application to rate control," *IEEE transactions on circuits and systems for video technology*, vol. 15, no. 12, pp. 1533–1544, 2005.

[7] H. Choi *et al.*, "Pixel-wise unified rate-quantization model for multi-level rate control," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 6, pp. 1112–1123, 2013.

[8] Z. He *et al.*, "Low-delay rate control for dct video coding via/spl rho/-domain source modeling," *IEEE transactions on Circuits and Systems for Video Technology*, vol. 11, no. 8, pp. 928–940, 2001.

[9] S. Milani *et al.*, "An accurate low-complexity rate control algorithm based on (ρ, E_q)-domain," *IEEE transactions on circuits and systems for video technology*, vol. 18, no. 2, pp. 257–262, 2008.

[10] H. Yang *et al.*, "A novel rate control scheme for video coding in HEVC-SCC," *IEEE Transactions on Broadcasting*, vol. 66, no. 2, pp. 333–345, 2019.

[11] B. Li *et al.*, "λ Domain rate control algorithm for high efficiency video coding," *IEEE Transactions on Image Processing*, vol. 23, no. 9, pp. 3841–3854, 2014.

[12] A. Ortega and K. Ramchandran, "Rate-distortion methods for image and video compression," *IEEE Signal Processing Magazine*, vol. 15, no. 6, pp. 23–50, 1998.

[13] F. Pereira *et al.*, *JPEG Pleno light field coding common test conditions v3.3*, Doc. ISO/IEC JTC 1/SC 29/WG1 N84049, 84th JPEG Meeting, Brussels, Belgium, 2019. [Online]. Available: https://jpeg.org/jpegpleno/documentation.html.

[14] ISO Central Secretary, "Information technology — Plenoptic image coding system (JPEG Pleno) — Part 4: Reference software," en, International Organization for Standardization, Geneva, CH, Standard ISO/IEC 21794-4:2022, 2022.