DISSERTAÇÃO DE MESTRADO

Codificador de Geometria de Nuvens de Pontos Sem Perdas Com Seleção de Contextos *Silhouette 4D*

Evaristo Ramalho Lucchezi da Silva

Brasília, julho de 2021

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

DISSERTAÇÃO DE MESTRADO

Codificador de Geometria de Nuvens de Pontos Sem Perdas Com Seleção de Contextos *Silhouette 4D*

Evaristo Ramalho Lucchezi da Silva

Dissertação de Mestrado submetida ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Mestre em Engenharia Elétrica

Publicação: PPGEE.DM-775/21

Banca Examinadora

Prof. Eduardo Peixoto Fernandes da Silva, Ph.D. ENE/UnB
Orientador

Prof. Ricardo Lopes de Queiroz, Ph.D. CiC/UnB
Examinador interno

Profa. Carla Liberal Pagliari, Ph.D. IME
Examinador externo

Prof. Daniel Guerreiro e Silva, Dr. ENE/UnB
Suplente

FICHA CATALOGRÁFICA

SILVA, EVARISTO RAMALHO LUCCHEZI DA
Codificador de Geometria de Nuvens de Pontos Sem Perdas Com Seleção de Contextos *Silhouette 4D* [Distrito Federal] 2021.
xvi, 62 p., 210 x 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2021).
Dissertação de Mestrado - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. Nuvens de Pontos	2. Compressão
3. Codificação Aritmética	4. Seleção de Contextos
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

SILVA, E.R.L (2021). *Codificador de Geometria de Nuvens de Pontos Sem Perdas Com Seleção de Contextos Silhouette 4D*. Dissertação de Mestrado, Departamento de Engenharia Elétrica, Publicação PPGEE.DM-775/21, Universidade de Brasília, Brasília, DF, 62 p.

CESSÃO DE DIREITOS

AUTOR: Evaristo Ramalho Lucchezi da Silva
TÍTULO: Codificador de Geometria de Nuvens de Pontos Sem Perdas Com Seleção de Contextos *Silhouette 4D*.
GRAU: Mestre em Engenharia Elétrica ANO: 2021

Evaristo Ramalho Lucchezi da Silva
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

Agradeço a Deus por ter me dado a oportunidade de concluir o mestrado. Agradeço também ao orientador Eduardo Peixoto, que sempre ajudou prontamente em todo o processo de criação do trabalhos aqui presente. Agradeço também a família e amigos, por sempre me ajudarem quando foi necessário, facilitando bastante a realização do trabalho.

RESUMO

Devido à grande quantidade de informação em uma nuvem de pontos, algoritmos de compressão são necessários para viabilizar seu uso nas mais diversas aplicações. Embora a maioria dos algoritmos de compressão para geometria de nuvens de pontos sejam baseados em *octrees*, recentemente bons resultados foram obtidos utilizando um algoritmo baseado em imagens binárias, o *Silhouette 3D* (S3D). O foco principal desse trabalho é a extensão do S3D para nuvens de pontos dinâmicas, onde uma nuvem de pontos de referência está disponível. Assim este trabalho apresenta três algoritmos distintos. O primeiro é um algoritmo de pré-processamento de imagens binárias que busca a melhor combinação possível de localização de contextos que será usada posteriormente em um codificador aritmético. Esse algoritmo de codificação sem perdas possui resultados superiores ao padrão JBIG para os dois conjuntos de dados avaliados. O segundo, *Silhouette 4D* (S4D), é a extensão do algoritmo S3D, incluindo codificação do tipo *inter-frame* dentro do algoritmo, e um algoritmo de seleção de modos. Esse trabalho conseguiu resultados superiores ao padrão MPEG G-PCC para todas as nuvens de pontos analisadas. Finalmente, o terceiro inclui a ideia de seleção de contextos desenvolvida para imagens binárias dentro do algoritmo do S4D, possuindo um ganho significativo se comparado com a aplicação anterior e possuindo, até o presente momento, os melhores valores de taxa para compressão sem perdas de geometria de nuvens de pontos para os dois conjuntos de dados analisados.

ABSTRACT

Due to the large amount of information in a point cloud, compression algorithms are needed to enable their use in the most diverse applications. Although most compression algorithms for point cloud geometry are based on *octrees*, recently good results have been achieved using an algorithm based on binary images, the *Silhouette 3D* (S3D). The main focus of this work is to extend the S3D to dynamic point clouds, creating an algorithm that uses a reference point cloud to improve compression effiency. Thus, this work presents three distinct algorithms. The first is a pre-processing algorithm for encoding binary images that searches for the best possible combination of contexts that will be used later in an arithmetic encoder. This lossless encoding algorithm has superior results to the JBIG standard for the 2 datasets evaluated. The second, *Silhouette 4D* (S4D), is the extension of the S3D algorithm, including inter-frame coding and a mode selection algorithm that chooses between two sets of contexts to encode. This work achieved better performance than the MPEG G-PCC standard for all point clouds analysed. Finally, the third algorithm includes the idea of context select developed for binary images within the S4D algorithm, having

a significant gain over the S4D rate values, having so far the best rate values for lossless geometry compression of point clouds for the two datasets analysed.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

Nuvens de pontos podem ser definidas como um conjunto de pontos em um espaço de três dimensões. Cada ponto possui necessariamente informação sobre localização, geometria, e podem possuir outras informações adicionais, chamadas de atributos, como cores ou vetores normais. Nuvens de pontos podem ser estáticas, em que não ocorre a mudança de posicionamento dos pontos dentro da região, com toda a informação representada em apenas um quadro, ou dinâmicas, em que a nuvem de pontos se modifica em cada instante no tempo, ou seja, uma nuvem de pontos dinâmica é uma sequência de quadros de nuvens de pontos. Este conceito é semelhante ao vídeo, que é uma sequência de imagens bidimensionais (2D).

O principal objetivo das aplicações que usam nuvens de pontos é a representação e visualização do espaço tridimensional descrito de maneira mais precisa possível, de forma que todas as características de ocupação e atributos sejam facilmente reconhecidas e tratadas. O processo de criação e compressão de nuvens de pontos atualmente é bastante usado em aplicações para carros autônomos [1] [2]. Nuvens de pontos também são bastante usadas na geração de DEM (Modelos de Elevação Digital), que tem por objetivo representar de maneira precisa informações de geografia de uma dada superfície [3]. Finalmente, existem aplicações de nuvens de pontos para representação de informações volumétricas em imageamento médico [4] [5].

Dada a imensa quantidade de pontos presentes em cada nuvem de pontos, são necessárias técnicas de compressão para facilitar a transmissão e o armazenamento desses dados. Essas técnicas de compressão podem ser sem perdas, ou seja, quando não há perda de informação no processo de compressão dos dados, ou com perdas, quando a informação resultante da compressão é diferente do conteúdo original. Os algoritmos de compressão de nuvens de pontos podem se dividir em duas categorias: codificadores *intra-frame*, que realizam a compressão usando somente informação da nuvem de pontos sendo codificada; e codificadores *inter-frame*, que realizam a compressão usando informação tanto da nuvem de pontos atual quanto de nuvens de pontos previamente codificadas. Este método é usado para compressão de nuvens de pontos dinâmicas.

Vários codificadores de geometria e atributos foram propostos nos últimos anos, de maneira a tentar viabilizar o uso nas mais diversas aplicações. As duas principais aplicações de compressão de nuvens de pontos estão sendo desenvolvidas atualmente pelo grupo MPEG (*Moving Picture Experts Group*), o G-PCC (*Geometry-based Point Cloud Compression*) [6] e o V-PCC (*Video-based Point Cloud Compression*) [7].

Um dos principais desafios para codificação das nuvens de pontos atualmente é a dificuldade em conceber um algoritmo de codificação *inter-frame* com bons resultados de compressão, pois a maioria dos codificadores desse tipo acaba tendo problemas de desempenho por conta da dificuldade em se analisar o comportamento dos pontos ao longo de diversas nuvens de pontos em sequência, prejudicando o desempenho de compressão desses. Por conta desse comportamento a

maioria dos trabalhos com bons resultados de compressão de nuvens de pontos, tanto para atributos quanto para geometria, usam codificadores do tipo *intra-frame*.

A proposta principal desse trabalho é trabalhar em um algoritmo de compressão *inter-frame* de geometria que consiga um desempenho superior ao G-PCC [6], que é um codificador *intra-frame*. Alguns trabalhos feitos nessa área [8] [9], fazem uso de técnicas de compressão de imagens binárias para codificar, sem perdas, a geometria das nuvens de pontos possuindo resultados promissores, se comparados com o G-PCC. Pensando nesse tipo de aplicação, esse trabalho propõe três aplicações, explorando conceitos de compressão de imagens binárias para compressão de nuvens de pontos. O primeiro algoritmo desenvolvido é um seletor de locais de contextos para imagens binárias, que busca determinar para a imagem atual, quais os melhores locais de contextos para uso em um codificador aritmético binário adaptativo ao contexto. O segundo algoritmo, *Silhouette 4D* (S4D), é a extensão do *Silhouette 3D* (S3D) [9], que divide a nuvem de pontos em uma série de silhuetas e as codifica usando locais de contextos tanto da própria silhueta quanto de silhuetas previamente codificadas. O S4D expande esse conceito e usa também locais de contextos de nuvens de pontos previamente codificadas, transformando-o em um codificador *inter-frame*. O terceiro algoritmo, *Silhouette 4D with Context Selection* (S4D-CS), usa a implementação de seleção de locais de contextos dentro do S4D, selecionando previamente um conjunto de locais de contextos padrão que será usado para codificar as silhuetas.

1.1 TRABALHOS PUBLICADOS

Para cada um dos algoritmos descritos neste trabalho, um artigo acadêmico foi publicado, sendo eles:

- *Context Selection for Lossless Compression of Bi-Level Images* publicado no XXXVIII Simpósio Brasileiro de Telecomunicações e Processamento de Sinais - SBrT 2020 [10].
- *Silhouette 4D: An Inter-Frame Lossless Geometry Coder Of Dynamic Voxelized Point Clouds*, publicado no *2020 IEEE International Conference on Image Processing* (ICIP 2020), que recebeu o prêmio *2020 Conference Best Paper Award for Industry* do *IEEE Signal Processing Society* (SPS) [11].
- *Silhouette 4D with Context Selection: Lossless Geometry Compression of Dynamic Point Clouds* publicado na revista *Signal Processing Letters* (SPL) [12].

1.2 ESTRUTURA DA DISSERTAÇÃO

O Capítulo 2 contém uma revisão de literatura em conceitos usados em codificação de imagens binárias como entropia, codificação aritmética e padrão JBIG [13]. Já o Capítulo 3 faz uma

revisão em métodos de compressão de nuvens de pontos, explicando conceitos de compressão de geometria, que é o foco desse trabalho, além dos padrões do MPEG e outros trabalhos desenvolvidos nessas áreas. O Capítulo 4 explica o algoritmo proposto para compressão de imagens binárias. O Capítulo 5 mostra primeiro o algoritmo do S4D [11], focando nas diferenças em relação ao S3D [9]. Na segunda metade do capítulo é mostrado o algoritmo do S4D-CS e seus resultados comparados com o padrão MPEG G-PCC [14], o S4D [11] e o S3D [9]. O capítulo 6 apresenta as conclusões tiradas acerca dos algoritmos desenvolvidos.

2 COMPRESSÃO DE FONTES BINÁRIAS

Para um melhor entendimento dos algoritmos desenvolvidos neste trabalho é preciso primeiro revisitar alguns conceitos da teoria da compressão de sinais, em especial a compressão de fontes binárias. Neste capítulo serão abordados os conceitos básicos de compressão, entropia, além de uma explicação do padrão JBIG [13], e no final alguns trabalhos desenvolvidos nessa área.

2.1 CONCEITOS EM COMPRESSÃO

Um algoritmo que recebe na entrada uma dada informação X e cria uma representação alternativa Y diferente do dado original é chamado de codificador. Além deste, é necessário um segundo algoritmo que recebe essa representação alternativa Y e cria uma reconstrução Z, chamada de decodificador. Um algoritmo de compressão de dados é a junção de um codificador e um decodificador onde o objetivo é que a representação alternativa Y seja mais compacta que a informação original X (isto é, necessite de um menor número de bits para o seu armazenamento). Um esquemático representado esse procedimento é mostrado na Figura 2.1.

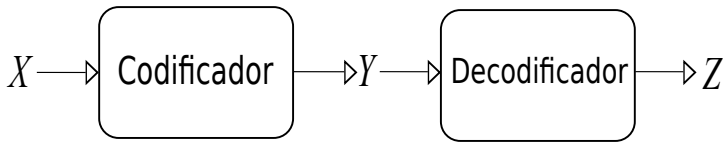


Figura 2.1: Esquemático de algoritmo de compressão.

Técnicas de compressão podem ser sem perdas, quando a informação em X é a mesma encontrada em Z, ou com perdas, quando a informação em Z é diferente de X. No segundo caso o ganho de compressão pode ser muito maior, ao custo de uma distorção na reconstrução. A compressão com perdas é usada em aplicações em que a perda de dados pode ser tolerada.

Técnicas de compressão podem ser avaliadas de várias maneiras como: medição de complexidade do algoritmo, a velocidade de execução do algoritmo, o tamanho do arquivo final ou o nível de distorção na reconstrução. Como neste trabalho fazemos uso de técnicas de compressão sem perdas, a medida usada para avaliação é o tamanho do arquivo comprimido pelo codificador.

2.1.1 Entropia

Para se entender melhor a ideia de compressão sem perdas, deve-se compreender alguns conceitos de probabilidade e estatística além do conceito de entropia introduzido por Shannon [15].

Todo o conjunto de possíveis ocorrências dentro de um experimento aleatório é chamado de espaço amostral. Cada possível ocorrência é chamada de evento, e o resultado de um experimento aleatório é chamado de variável aleatória. Assim, uma variável aleatória pode ser definida como o resultado de um experimento aleatório. Uma variável aleatória pode gerar amostras independentes, onde cada resultado não possui nenhuma dependência com valores previamente obtidos, ou dependentes, onde o valor de cada amostra muda as probabilidades de ocorrência das próximas. Um exemplo simples de experimento aleatório é o lançamento de um dado, que possui espaço amostral $S = \{1, 2, 3, 4, 5, 6\}$, e caso o dado não esteja viciado, cada valor possui probabilidade fixa de $\frac{1}{6}$. Neste exemplo, a realização de um lançamento do dado não interfere na probabilidade do próximo lançamento, isto é, as amostras são independentes.

A entropia de uma fonte de informação, podendo essa ser um espaço amostral, corresponde à quantidade de informação que pode ser obtida através dos valores de probabilidade de ocorrência de cada evento, usando a seguinte fórmula:

$$H(S) = -\sum_{x_n} P(x_n) \log_m P(x_n) \ , \tag{2.1}$$

considerando $H(S)$ como o valor de entropia para um dado espaço amostral S, e m é a base do logaritmo, que define a unidade. O mais comum é usar $m = 2$, que define a unidade como sendo bits.

Considerando o exemplo do experimento aleatório do dado, o valor de entropia, considerando unidade de medida binária, pode ser definido como,

$$H(S) = -(P(x_n = 0) \log_2 P(x_n = 0) + ... + P(x_n = 6) \log_2 P(x_n = 6)),$$

e, como todas as probabilidades são iguais,

$$H(S) = -6 \left[\frac{1}{6} \log_2 \left(\frac{1}{6} \right) \right],$$
$$H(S) = 2.59 \ bits \ por \ simbolo \ .$$

Logo, para a transmissão de toda a informação desse experimento aleatório, são necessários em média 2.59 bits para a codificação de cada um dos símbolos.

Para um espaço amostral com 2 possíveis eventos, $x_n = 0$ ou $x_n = 1$, podemos resumir essa equação para a forma:

$$H(S) = -(P(x_n = 0) \log_2 P(x_n = 0) + P(x_n = 1) \log_2 P(x_n = 1)) \ . \tag{2.2}$$

Shannon em seu trabalho [15] demonstrou que esse valor de entropia é a quantidade mínima de símbolos binários necessários para compressão sem perdas de um valor dentro de um conjunto, considerando sempre independência entre os elementos dentro do conjunto.

2.1.2 Entropia de Fontes Dependentes

A Eq 2.1 define a entropia de fontes que geram amostras independentes. Na prática, a maioria das fontes reais apresenta alguma dependência entre as amostras. Essa dependência pode ser modeladas por modelos de Markov. Seja x_n uma sequência de observações. Em um modelo de Markov de ordem k temos:

$$P(x_n|x_{n-1}, ..., x_{n-k}) = P(x_n|x_{n-1}, ..., x_{n-k}, ...), \tag{2.3}$$

ou seja, limitamos a modelagem do passado da fonte para as k amostras passadas. No jargão de compressão, as amostras passadas $x_{n-1}, x_{n-2}, ..., x_{n-k}$, conhecidas, formam o *contexto* para o símbolo atual x_n.

Cada conjunto $S_i = x_{n-1}, x_{n-2}, ..., x_{n-k}$ forma um contexto, que é modelado com uma distribuição de probabilidades própria. A entropia de um processo modelado desta maneira é dada por:

$$H = \sum_i P(S_i)H(S_i). \tag{2.4}$$

Como exemplo, considere a imagem binária na Figura 2.2. Considerando as amostras independentes, a imagem possui probabilidades:

$$P(x_n = 0) = 0.56,$$
$$P(x_n = 1) = 0.44,$$

e, com isso, podemos então calcular a entropia $H(S)$ de um modelo de Markov de ordem 0 para esta imagem (isto é, quando a Eq 2.4 se reduz a 2.1), como o símbolo a ser codificado de uma imagem é o *pixel*, a unidade da entropia é dada em bits por *pixel* (bpp),

$$H(S) = -P(x_n = 0)\ \log_2 P(x_n = 0) - P(x_n = 1)\ \log_2 P(x_n = 1),$$
$$H(S) = -0.56\ \log_2 0.56 - 0.44\ \log_2 0.44 = 0.99\, bpp.$$

Considerando agora o contexto de ordem 1 (isto é a amostra previamente codificada), temos:

$$P(x_n = 0|x_{n-1} = 0) = 0.96,$$
$$P(x_n = 1|x_{n-1} = 0) = 0.04,$$
$$P(x_n = 0|x_{n-1} = 1) = 0.05,$$
$$P(x_n = 1|x_{n-1} = 1) = 0.95\ ,$$

Figura 2.2: Imagem *Lena* Binarizada.

que são as probabilidades condicionais da imagem. Podemos definir o estado S_0 como $x_{n-1} = 0$ e S_1 como $x_{n-1} = 1$. Calculando as entropias para cada contexto temos,

$$H(x_n|x_{n-1} = 0) = -0.96\ \log_2 0.96 - 0.04\ \log_2 0.04 = 0.26\, bpp,$$
$$H(x_n|x_{n-1} = 1) = -0.05\ \log_2 0.05 - 0.95\ \log_2 0.95 = 0.30\, bpp.$$

Como calculamos as probabilidades de ocorrência de cada símbolo (bit), e as entropias por contexto, podemos agora calcular a entropia da imagem:

$$H = P(x_{n-1} = 0)\ H(x_n|x_{n-1} = 0) + P(x_{n-1} = 1)\ H(x_n|x_{n-1} = 1),$$
$$H = 0.56 \cdot 0.26 + 0.44 \cdot 0.30 = 0.2742\ bpp.$$

É importante observar que a entropia da imagem não muda, uma vez que a imagem em si é a mesma. Porém, usando modelos mais sofisticados podemos estimar melhor essa entropia e atingir taxas de compressão maiores.

2.2 CODIFICAÇÃO ARITMÉTICA

A codificação aritmética é um tipo de codificação de entropia para compressão sem perdas de dados. A primeira ideia para esse tipo de codificação foi descrita por Shannon [15]. Vários

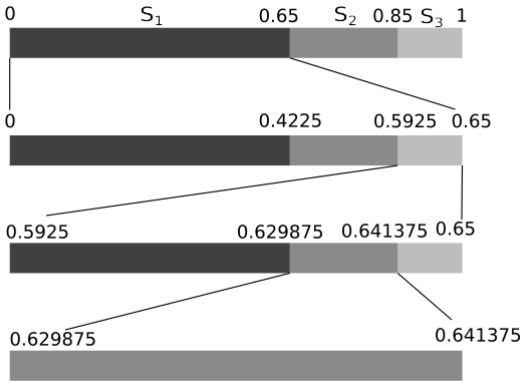


Figura 2.3: Procedimento de Codificação Aritmética.

trabalhos buscaram otimizar o algoritmo [16] [17] [18] [19], mas o trabalho mais conhecido é o trabalho desenvolvido por Rissanen e Langdon em 1979 [20].

A ideia da codificação aritmética é representar uma sequência de valores com apenas um único número no intervalo $[0, 1)$. O algoritmo divide este subintervalo em uma quantidade de subintervalos igual ao número de símbolos, sem sobreposição. O tamanho de cada subintervalo é proporcional à probabilidade de ocorrência daquele símbolo. Símbolos mais prováveis são associados a subintervalos maiores, e símbolos menos prováveis são associados a subintervalos menores. O algoritmo então codifica um símbolo escolhendo o subintervalo associado a ele, e divide este subintervalo da mesma forma que o primeiro intervalo foi dividido, repetindo o procedimento para um novo símbolo. Como os subintervalos são escolhidos sem sobreposição, cada subintervalo define unicamente uma sequência, de forma que, sabendo o subintervalo final, o número de símbolos codificados, o alfabeto de símbolos e suas probabilidades, a sequência é unicamente definida. A informação passada para o decodificador, no entanto, não é o subintervalo final $[l_n, u_n)$, mas apenas um número qualquer dentro desse intervalo, chamado de *tag*. Geralmente, é escolhido o número que pode ser representado com a menor quantidade de dígitos possível.

Um exemplo de codificação aritmética é mostrado na Figura 2.2, onde tem-se um alfabeto com três símbolos s_1, s_2 e s_3, com probabilidades de 65%, 20% e 15% respectivamente. Matematicamente, as fórmulas para atualização de intervalos são as seguintes:

$$l^{(n)} = l^{(n-1)} + \left(u^{(n-1)} - l^{(n-1)}\right) \cdot F_X(x_n - 1),$$
$$u^{(n)} = l^{(n-1)} + \left(u^{(n-1)} - l^{(n-1)}\right) \cdot F_X(x_n),$$

sendo $l^{(n)}$ e $u^{(n)}$ os novos intervalos inferior e superior, respectivamente, $l^{(n-1)}$ e $u^{(n-1)}$ os intervalos anteriores, e $F_X(.)$ a Função de Distribuição Acumulada. Note que os valores iniciais são $[l^0, u^0) = [0, 1)$.

No procedimento mostrado na Figura 2.3, é codificada a sequência s_1, s_3, s_2. Para cada novo

símbolo codificado, o intervalo é atualizado usando as fórmulas de atualização de intervalos mostradas previamente. O primeiro símbolo codificado, s_1, reduz o intervalo de valores para $[0, 0.65]$, permanecendo as probabilidades de ocorrência de cada símbolo dentro do intervalo reduzido. Logo após é codificado s_3, reduzindo o intervalo para $[0.5925, 0.65]$ e finalmente é codificado s_2, gerando o intervalo final $[0.629875, 0.641375]$. Como mencionado anteriormente, qualquer valor dentro desse intervalo é capaz de representar a sequência. O valor com menor quantidade de símbolos binários dentro dessa sequência é $0.640625(0.101001_2)$, logo podemos usá-lo para representar esta sequência de símbolos.

2.2.1 Codificação arimética adaptativa ao contexto

Seguindo a ideia discutida na seção 2.1.2, podemos melhorar o modelo de probabilidade usado olhando símbolos previamente codificados. Por exemplo, podemos modelar a probabilidade como $p(x_n|x_{n-1}, x_{n-2}, ..., x_{n-k})$, sendo que cada valor específico de $[x_{n-1}, x_{n-2}, ..., x_{n-k}]$ é chamado de contexto.

Após a codificação de um símbolo usando o modelo probabilístico acima, nada impede que diferentes probabilidades sejam usadas para codificar o próximo símbolo, desde que essa mudança seja transmitida para o decodificador, ou que ele possa inferir esta mudança de alguma forma. O mais comum é usar símbolos previamente codificados, que podem ser inferidos no decodificador sem a necessidade de transmissão de informação lateral. Para a codificação de imagens como a Figura 2.2 os modelos probabilísticos são obtidos através da verificação dos valores das posições previamente codificados da imagem (os *pixels*), tornando desnecessária a transmissão de informação lateral.

Esse tipo de codificação pode ser adaptativo, ou seja, a cada símbolo dentro de uma sequência codificado, o modelo probabilístico usado no processo é atualizado, gerando um modelo que no final irá representar de maneira mais fiel o comportamento de ocorrência dos contextos no geral. Um codificador aritmético que se utiliza desse tipo de abordagem é chamado de codificador aritmético adaptativo ao contexto.

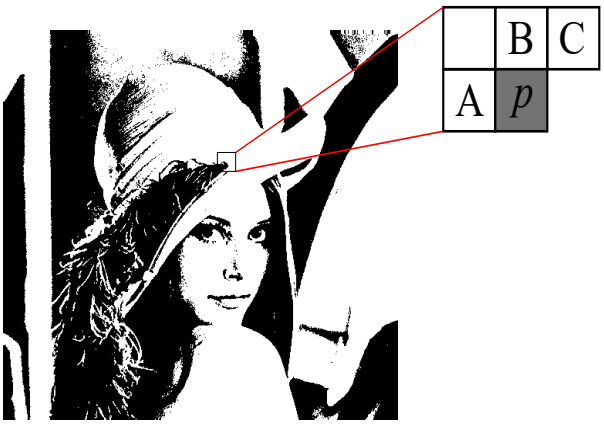
2.3 CODIFICAÇÃO DE IMAGENS BINÁRIAS

Uma imagem binária consiste em uma matriz 2D de *pixels* que só podem possuir dois valores, definidos como branco e preto. A compressão desse tipo de imagem foi amplamente utilizada em várias aplicações ao longo do tempo, sendo a aplicação mais popular a transmissão de documentos através de máquinas de fax [21].

O estado da arte em compressão de imagens binárias é obtido através de codificação aritmética adaptativa ao contexto, que é utilizada tanto nos padrões JBIG [13] e JBIG2 [22] [23]. Nessa técnica os *pixels* da imagem são codificados começando pelo primeiro *pixel* superior à esquerda,

continuando a codificação na ordem esquerda-direita, e, ao chegar no último *pixel* da linha, continua a codificação na linha abaixo. O procedimento de codificação é feito em toda a imagem, logo, a sequência a ser codificada são todos os *pixels* da imagem. Uma imagem de dimensão 512×512, por exemplo, possui uma sequência de 262144 bits.

Se o método de probabilidade não fizer uso de contextos, o algoritmo irá usar somente um conjunto de probabilidades para codificar todos os bits da imagem. No entanto, se o codificador usar um modelo com contextos, o procedimento é um pouco diferente, já que agora existe mais de um modelo probabilístico possível para codificação. O algoritmo começa então verificando um certo número de *pixels* codificados previamente nas vizinhanças do *pixel* atual sendo codificado. Cada um desses *pixels* usado no procedimento de codificação para criação dos contextos é chamado neste trabalho de local de contexto. Cada possível valor binário gerado a partir desses *pixels* é um contexto. A Figura 2.4 mostra as diferenças entre os conceitos de local de contexto e contexto, e a Figura 2.5 mostra os dezesseis locais de escolha que o codificador aritmético adaptativo ao contexto do JBIG pode usar.



A	B	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Figura 2.4: Locais de contexto A,B e C (no centro) para a imagem *Lena* (à esquerda). Cada possível valor desse conjunto é definido como um contexto, sendo todos os possíveis contextos para os 3 locais de contexto mostrados na tabela à direita.

2.3.1 JBIG

O algoritmo padrão para compressão sem perdas de imagens binárias que faz uso de codificação aritmética é o JBIG (*Joint Bi-Level Image Experts Group*) [13], criado em 1992. O algoritmo é composto de cinco blocos que fazem algumas operações na imagem, buscando sempre mais eficiência na compressão, sem nenhuma perda de informação, com um codificador aritmético com contextos adaptativos (*QM coder*) [21]. São eles:

- Redução de Resolução: Cria uma imagem com metade do tamanho da original, retirando metade das linhas e das colunas, procurando preservar o máximo de informação possível.
- Predição Típica: Procura por regiões de cor sólida e sinaliza para o codificador aritmético

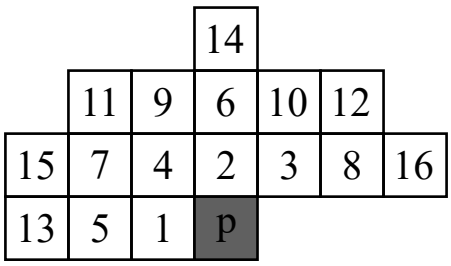


Figura 2.5: Locais de contextos usados no padrão JBIG.

ignorá-las na codificação. Também procura por *pixels* que não possuem valores típicos, isso é, possuem um valor único diferente de todos os *pixels* na vizinhança, também sinalizando-os ao codificador.

- Predição Determinística: Analisa a imagem original e a imagem de resolução reduzida de maneira a predizer o valor de um *pixel*, esse *pixel* predito não será codificado.
- Modelos padrão e adaptativo: Criam os modelos e selecionam os contextos que serão usados no codificador aritméticos

Depois de todo o procedimento feito pelos cinco blocos descritos, o algoritmo codifica usando o codificador aritmético binário adaptativo ao contexto.

2.3.2 Outros Trabalhos em codificação de imagens

Trabalhos que fazem uso de codificação de imagens binárias foram desenvolvidos de maneira a aproveitar as técnicas para alguma aplicação específica. Pinho em seu trabalho [24] usa algoritmos de codificação binárias para codificar os mapas de contorno de uma imagem binária, enquanto Abdat *et al* [25] representa uma imagem binária com um código *Gray*, codificando cada plano como uma imagem binária. Mais recentemente, Khursheed *et al* [26] utilizou imagens binárias para representar objetos, aplicando compressão e utilizando técnicas de Região de Interesse. Outro trabalho recente [8] usa imagens binárias para representação de ocupação dentro de uma nuvem de pontos.

Alguns trabalhos foram propostos de maneira a otimizar o funcionamento do algoritmo do JBIG [13]. O trabalho [27] usa um algoritmo de *quadtree* para acelerar o processo de compressão, gerando uma pequena perda no processo. Martins e Forchhammer [28] propõem um algoritmo que possui um pré processamento que gera uma árvore binária que indica quais *pixels* serão utilizados como contextos, essa abordagem é baseada na ideia descrita em [29] que nem todos os contextos do JBIG [13] são usados igualmente, e que mais da metade dos bits são gerados dos 10 contextos mais importantes. Esse trabalho também compara o uso de árvores estáticas, geradas com o uso de imagens de treinamento, com árvores dinâmicas, geradas durante o processo de codificação, que posteriormente serão passadas para o decodificador.

Trabalhos propondo algoritmos alternativos ao do JBIG [13] foram desenvolvidos também.

Como o trabalho [30], que propõe um codificador de duas regras para comprimir a imagem. Primeiro, o algoritmo calcula a probabilidade dependente de contexto da imagem, e a usa num módulo de predição de *pixel*, ajustando um parâmetro *Run-Length*, criando o codificador BLC. Zahir *et al* [31] propõe o *Near Minimum Sparse Pattern Coding*, que possui codificação em blocos semelhantes ao JBIG2, no entanto, com algoritmo de baixa complexidade. Zhou *et al* [32] propõe uma aplicação de codificação sem perdas com o algoritmo proposto de Eliminação de Redundâncias Direta e um Modo de contextos dinâmicos para codificação aritmética com contextos.

2.4 CONCLUSÃO DO CAPÍTULO

Neste capítulo foram abordados alguns conceitos para codificação de imagens que serão fundamentais para o entendimento dos algoritmos desenvolvidos no trabalho. Em especial, os algoritmos desenvolvidos se baseiam em codificação aritmética adaptativa ao contexto, discutida na seção 2.2.1. No próximo capítulo serão abordados conceitos de codificação de nuvens de pontos, também fundamentais para o entendimento dos algoritmos desenvolvidos neste trabalho.

3 COMPRESSÃO DE NUVENS DE PONTOS

Nuvens de pontos são um conjunto de pontos em um espaço tridimensional que buscam representar uma determinada forma ou objeto da maneira mais realista possível. Normalmente, cada ponto dessa nuvem de pontos pode ser localizado em qualquer posição do espaço. Para que estas nuvens de pontos sejam processadas, é comum que seja feito o processo de voxelização. Voxelização é o processo que converte a nuvem de pontos de um espaço contínuo para um espaço discreto. Esse processo converte os pontos da nuvem de pontos em *voxels* (*volume elements*). Cada *voxel* é representado geometricamente através de suas coordenadas espaciais X,Y e Z. O arquivo também pode conter informações extras para a geração da nuvem de pontos que são chamadas de atributos. Essas informações podem representar cores, componentes de vetores normais ou até informações a respeito da geração do arquivo. Uma renderização de uma projeção de uma nuvem de pontos pode ser vista na Figura 3.1. Essa nuvem de pontos contém tanto informações de geometria quanto cores e pertence ao Banco de Dados *Microsoft Voxelized Upper Bodies* [33].

Como nuvens de pontos são um conjunto de informações diferentes, as técnicas de compressão para geometria e atributos são diferentes. Neste capítulo serão abordadas algumas técnicas, começando pelo procedimento padrão para compressão de geometria, *octree*, depois uma explicação do padrão G-PCC [6], além da descrição do codificador *Silhouette 3D* e revisão de literatura de outros métodos para compressão de nuvens de pontos.

3.1 CODIFICAÇÃO DE GEOMETRIA DE NUVENS DE PONTOS

A estratégia mais comum entre codificadores de geometria desenvolvidos ao longo dos anos é tentar identificar o máximo de regiões vazias possíveis dentro da nuvem de pontos, pois geralmente 98% do espaço dentro de uma nuvem de pontos é vazia, codificando assim somente as regiões ocupadas [34]. O principal algoritmo desenvolvido nessa área é o escaneamento *octree* descrito a seguir.

Figura 3.1: Nuvem de pontos Ricardo.

3.1.1 Octrees

O escaneamento *octree*, desenvolvido inicialmente em [35], é uma técnica de modelamento geométrico que divide a nuvem de pontos em regiões, identificando quais estão vazias e quais possuem pontos. No primeiro nível do escaneamento, o algoritmo assume que a nuvem de pontos está dentro de um cubo de lado 2^N, com N sendo a quantidade de bits usada para descrição da posição de um ponto em um determinado eixo, e divide a nuvem de pontos em 8 sub-cubos de lado 2^{N-1}. O codificador usa o código binário 0 para representar uma região vazia e 1 para regiões não vazias. Logo, esse primeiro nível usa 8 bits para sinalizar a ocupação. Como o algoritmo já sabe que as regiões marcadas com zero não possuem *voxels*, o algoritmo para a codificação nessas regiões, continuando o procedimento de subdivisão nos sub-cubos ocupados, que ainda não estão claramente representados pelo codificador. O procedimento é recursivo, e os sub-cubos marcados como ocupados são novamente subdivididos no segundo nível em oito cubos de tamanho 2^{N-2}. Esse procedimento é repetido até que todos os *voxels* sejam identificados pelo codificador.

Um exemplo de escaneamento *octree* de um dado espaço 3D é mostrado na Figura 3.2, com as regiões ocupadas representadas com a cor cinza. Na imagem a direita, é realizado o escaneamento *octree* no primeiro nível, verificando a ocupação dos oito primeiro sub-cubos, dentre eles, dois estão ocupados, gerando o valor 01100000. Já na imagem a esquerda, é mostrado o escaneamento *octree* no segundo nível, como somente 2 sub-cubos estavam ocupados, o processo de decomposição continuou somente nos 2 sub-cubos ocupados, gerando os valores 10000000 para a primeira região e 01001000 para a segunda. O procedimento continua até que todos os voxels sejam escaneados. Até o segundo nível, a sequência (*bitstream*) 011000001000000001001000 descreve a nuvem de pontos.

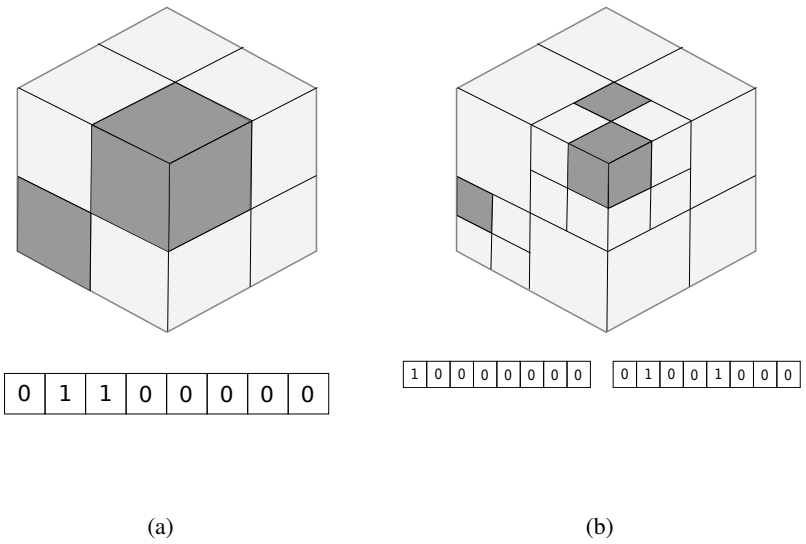


Figura 3.2: Escaneamento octree: (a) Primeiro nível (b) Segundo nível.

Como as nuvens de pontos no geral são esparsas, esse procedimento por si só leva a uma alta taxa de compressão. Alguns autores sugerem utilizar codificadores de entropia no *bitstream* resultante como em [36] e [37].

3.2 PADRONIZAÇÃO DO MPEG

O MPEG [38] possui um processo de padronização de compressão de nuvem de pontos em andamento. Nesse processo, as nuvens de pontos são divididas em 3 categorias: categoria 1 são as nuvens de pontos estáticas; categoria 2 são nuvens de pontos dinâmicas; e categoria 3 são nuvens de pontos adquiridas dinamicamente através de LiDAR (*Light Detection And Ranging*). Duas tecnologias de compressão foram definidas, o G-PCC [6] que é responsável pela compressão de nuvens de pontos da categoria 1 e 3 e o V-PCC [7], que usa técnicas de compressão de vídeos para as nuvens de pontos de categoria 2. Também foram desenvolvidos dois modelos de teste, o TMC13 (*Test Model 1 and 3*) para o G-PCC [6] e o TMC2 (*Test Model 2*) para o V-PCC [7], como o V-PCC só possui modo de compressão com perdas, ele não será aprofundado neste trabalho. Já o G-PCC possui compressão de geometria com e sem perdas, logo é interessante um pequeno aprofundamento, pois os algoritmos desenvolvidos são avaliados usando como base esse codificador.

3.2.1 G-PCC

O algoritmo do G-PCC [6], padrão do grupo MPEG para compressão de nuvens de pontos do tipo 1 e tipo 3, foi lançado em 2019 e possui codificação completa de geometria e cor, sendo os codificadores de geometria usados em *trisoup* e em *octree*. Já para codificação de atributos é usado o método *Prediction*, que possui um esquema de predição baseado em distância, e os métodos RAHT [34] (*Region-Adaptive Hierarchical Transform*) e *Lifting* [6].

3.2.1.1 Codificador geométrico do G-PCC

O codificador de geometria do G-PCC [6] primeiro representa a nuvem de pontos utilizando *octrees*, conforme descrito na seção 3.1.1. Em seguida é usado um algoritmo de algoritmo de codificação aritmética adaptado ao contexto adicional. Nesse procedimento o algoritmo usa como contexto os bits do nível superior previamente codificados, além dos bits previamente codificados do mesmo nível. Para se codificar o segundo bit no entanto, são usados esses 8 bits, mais o bit previamente codificado, o procedimento de codificação continua aumentando a quantidade de contextos até chegar no ultimo *pixel* do sub-cubo, logo o procedimento varia o número de locais de contexto usados entre 8 e 15.

O escaneamento em *octree* do G-PCC se alia a outro algoritmo chamado de DCM (*Direct Code Mode*), que codifica as coordenadas (X,Y e Z) diretamente dos *voxels* mais isolados, caso

certas condições de vizinhanças de *pixels* sejam satisfeitas. Esse modo é útil para codificar *pixels* que possuem poucos vizinhos, e que necessitam de muitos níveis de *octree* para que sejam bem representados.

A outra opção de codificação de geometria presente no padrão é o algoritmo baseado em *triangle soup* ou *trisoup*, que representa a superfície de um objeto dentro de uma nuvem de pontos como uma série de triângulos, sendo os vértices desse triângulos devidamente codificados por um codificador aritmético. Esse método é bastante usado em nuvem de pontos com superfícies bastante densas. É importante se observar que o método *trisoup* envolve perdas na reconstrução.

3.3 *SILHOUETTE 3D*

O codificador *Silhouette 3D* [9], usa uma representação fundamentalmente diferente da *octree*. A ideia aqui é representar a nuvem de pontos como uma sequência de imagens, e utilizar técnicas de compressão de imagens binárias para comprimir a nuvem de pontos.

Considere uma nuvem de pontos sendo codificada, a matriz de ocupação $G(i,j,k)$, com $i,j,k = 1,..,N$, com $N = 2^r$ e r a resolução do processo de voxelização, é a representação da informação de ocupação da geometria da nuvem de pontos, sendo definida como,

$$G(x,y,z) = \begin{cases} 1, \text{se o ponto x,y,z pertence à nuvem de pontos,} \\ 0, \text{caso contrário,} \end{cases} \tag{3.1}$$

essa matriz pode então ser considerada como um *array* tridimensional binario. A técnica de decomposição usada neste trabalho é baseada em uma transformação $silhouette$ definida como:

$$\begin{aligned} I_{n,m}^{axis}(i,j) &= silhouette(G, axis, n, m) \\ &= \begin{cases} \sum_{s=n}^{m} G(s,i,j) \text{ if } axis = x \\ \sum_{s=n}^{m} G(i,s,j) \text{ if } axis = y \\ \sum_{s=n}^{m} G(i,j,s) \text{ if } axis = z \ , \end{cases} \end{aligned} \tag{3.2}$$

sendo G a matriz de ocupação, $axis$ o eixo de referência escolhido (x,y, ou z), n e m os intervalos inicial e final, respectivamente. Para um dado eixo de referência, a transformação $silhouette$ seleciona os pontos dentro da matriz de ocupação dentro do intervalo $[n,m]$ e agrupa-os em uma única imagem realizando operações de soma nos *pixels* ponto a ponto. Como G é um $array$ booleano, as operações de soma são na realidade uma série de operações booleanas OR. As imagens binárias resultantes $I_{n,m}^{axis}(i,j)$ se assemelham à silhuetas formadas pela projeção de luz na nuvem de pontos na direção do eixo de referência, o que dá nome ao codec.

Um exemplo de silhueta obtida da primeira nuvem de pontos da sequência *Ricardo9* [33] é mostrada na Figura 3.3 onde as regiões com pontos estão marcadas em preto e as sem pontos em

branco. Essa silhueta foi criada usando todo o intervalo da nuvem de pontos no eixo y, ou seja $I^y_{1,N}(i,j)$

Figura 3.3: Silhueta da nuvem de pontos *Ricardo9*.

Cada silhueta gerada carrega individualmente uma grande quantidade de informação. Na Figura 3.4, dado que $I^z_{2,4}(0,3) = 0$ corresponde à um *pixel* não ocupado, podemos então concluir que os *voxels* $P(0,3,z)$, para $z = 2,3,4$, são todos não ocupados. Em geral, $I^{axis}_{n,m}(x,y) = 0$, indica que, para $n \leq k \leq m$:

$$\begin{cases} P(k,x,y) = 0 \text{ se } axis = x, \\ P(x,k,y) = 0 \text{ se } axis = y, \\ P(x,y,k) = 0 \text{ se } axis = z. \end{cases}$$

Conceitualmente, o algoritmo constrói uma árvore binária aonde cada nó da árvore contém informação sobre uma silhueta para uma região específica da nuvem de pontos. O nó raiz armazena $I^{axis}_{1,N}$. O algoritmo continua então dividindo o intervalo $[1,N]$ em outros dois de igual tamanho, construindo então as subárvores da direita e da esquerda recursivamente. Deste modo, se um nó arbitrário armazena $I^{axis}_{a,b}$, seu sub-nó da esquerda armazena $I^{axis}_{a,b/2}$ e o sub-nó da direita $I^{axis}_{b/2+1,b}$. Em qualquer ponto da árvore, os *pixels* não ocupados da silhueta informam que o *pixel* correspondente em todas as subárvores da esquerda e direita também não são ocupados. A estrutura em árvore permite que o algoritmo consiga localizar os *pixels* ocupados de maneira semelhante à uma busca binária.

A Figura 3.5 mostra o primeiro nível do processo de decomposição, chamado no trabalho de Decomposição diádica para a nuvem de pontos *Ricardo9*. Para se codificar a silhueta atual $I^y_{1,N/2}$ por exemplo a silhueta $I^y_{1,N}$ do nó pai é usada como uma máscara de ocupação para delimitar a área que deve ser verificada pelo codificador. Enquanto para a codificação de $I^y_{N/2+1,N}$ são usadas como máscara tanto $I^y_{1,N}$ quanto $I^y_{1,N/2}$.

A Decomposição diádica seguida do modelo de codificação aritmética descrita anteriormente funciona de maneira eficiente para os primeiros níveis, entretanto sua efetividade diminui a medida que a quantidade de pontos para se codificar em cada silhueta se torna muito reduzida. Para tentar obter algum ganho de compressão nessas regiões com menos pontos, foi desenvolvido o algoritmo de Divisão em silhuetas unitárias (*Single mode* no trabalho original). Esse algoritmo é acionado quando as silhuetas tem um intervalo de valores menor que um dado valor P, definido como 16 no trabalho.

Seja $I_{a,b}^{axis}$ uma silhueta com $b-a \leq P$, a Divisão em silhuetas unitárias irá codificar cada corte dentro desse intervalo separadamente, ou seja, serão codificadas as silhuetas $\{I_{a,a}^{axis}, I_{a+1,a+1}^{axis}, ..., I_{b,b}^{axis}\}$, usando a silhueta anterior para geração dos locais de contexto. Depois de codificadas, soma-se o valor de todas as codificações e compara-se com o valor obtido codificando $I_{a,b}^{axis}$, o método que obteve menor valor de taxa será sinalizado e passado para o *bitstream* final.

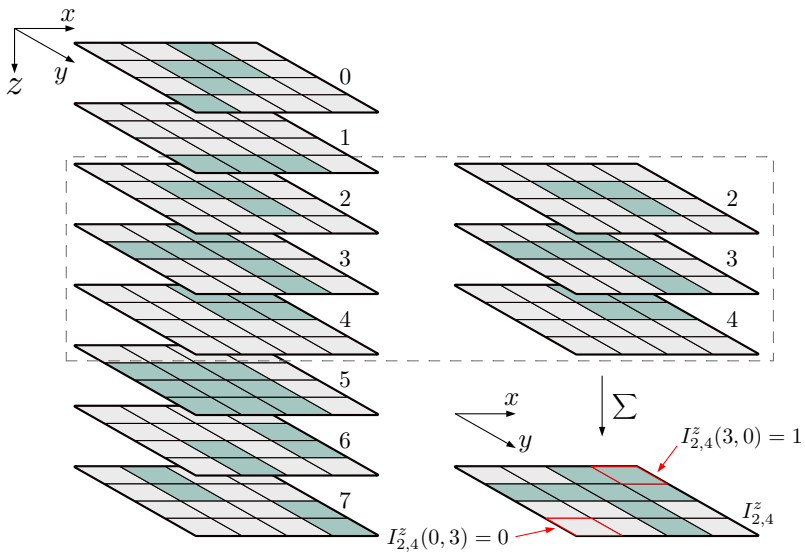
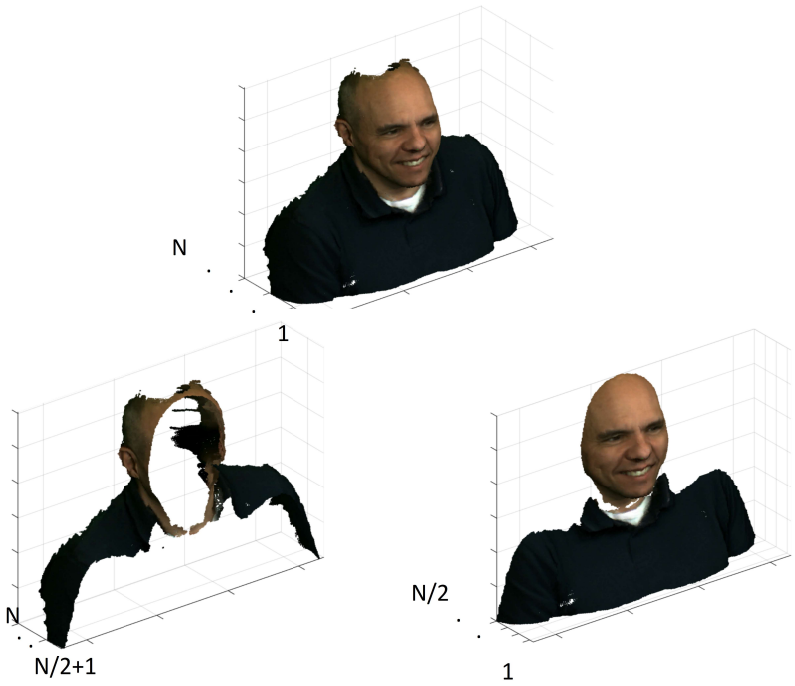


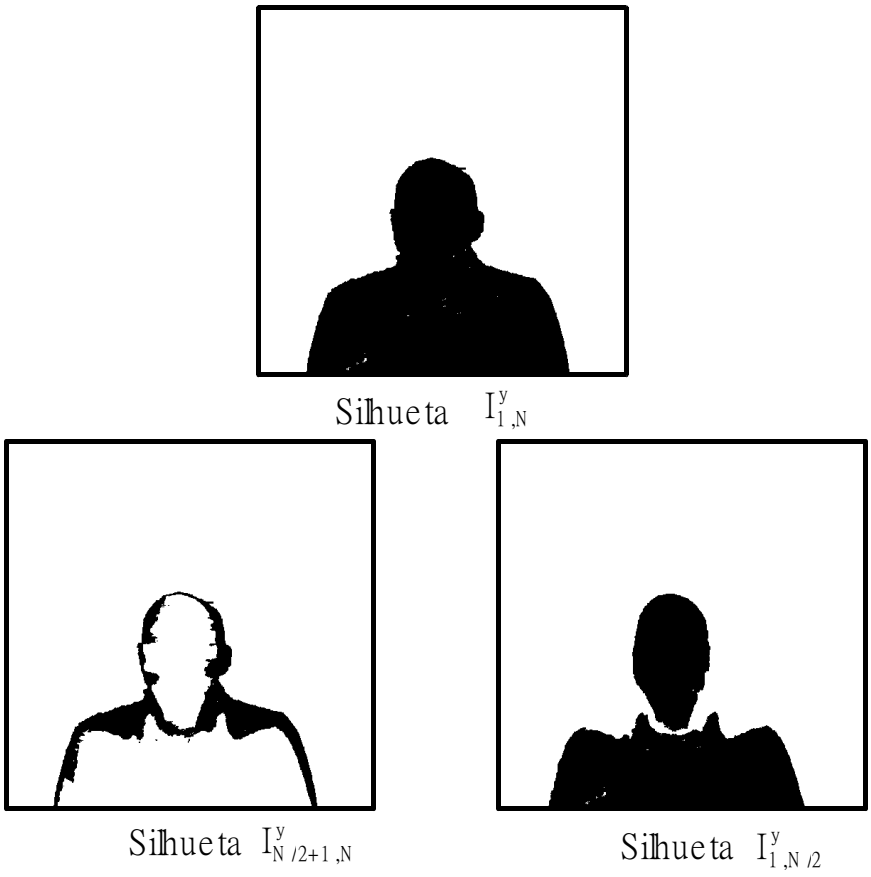
Figura 3.4: Decomposição em silhuetas.

3.3.1 Descrição do Algoritmo

O algoritmo 1 descreve o funcionamento do S3D. Primeiro, o algoritmo começa recebendo a matriz de ocupação que é previamente calculada, criando a primeira silhueta com todos os pontos no eixo selecionado (linha 3), codificando assim a primeira silhueta (linha 4). Em seguida é realizada a Decomposição diádica, gerando as duas silhuetas, cada uma com metade do intervalo do eixo (linhas 6 e 7), que serão então codificadas (linhas 10 e 11), usando máscaras previamente geradas, que indicam quais bits devem ser codificados (linhas 8 e 9). Depois, o codificador usa recursividade para percorrer a árvore de silhuetas e codificar as demais silhuetas (linhas 12,13,14), testando o algoritmo de Divisão em silhuetas unitárias para as silhuetas que satisfaçam a condição de intervalo P (linha 16), no final é comparado qual dos dois métodos gerou uma saída mais compacta (linha 17).



(a)



(b)

Figura 3.5: Decomposição diádica: (a) Nuvens de pontos (b) Silhuetas geradas.

Algorithm 1: S3DCode()

Input: $\mathbf{G}$: Matriz de ocupação.
$axis$: O eixo em que a divisão ocorre
n,m : Os intervalos inferior e superior
I : A silhueta do nível anterior
P : Tamanho do intervalo para uso da Divisão em silhuetas unitárias

Output: $bitstream$: O arquivo com a representação compacta de G

```
1  begin
2      if I Não for passado como argumento then
3          I = silhouette(G,axis,m,n)
4          encodeFirstImage(Y)
       // Divide o intervalo em 2
5      mid = ((m - n + 1) / 2) + n - 1
       // Realiza Decomposição diádica
6      I_L = silhouette(G, axis, n, mid)
7      I_R = silhouette(G, axis, mid+1, m)
       // Criação das máscaras
8      mIleft = I
9      mIRight = and(I,I_L)
       // Codifica silhueta da esquerda
10     encodeSilhouette(I_L,mIleft)
       // Codifica silhueta da direita
11     encodeSilhouette(I_R,mIRight)
       // Aplica o código recursivamente
12     if mid > n then
13         S3DCode(G, axis, n, mid, I_L)
14         S3DCode(G, axis, mid + 1, m, I_R)
       // Testa a codificação de Divisão em silhuetas unitárias
15     if (m - n + 1) <= P then
16         singleModeEnc(G, axis, n, m, Y)
       // Seleciona o melhor modo, caso a Divisão em silhuetas
          unitárias não seja usado, salva o bitstream gerado no modo
          diádico
17     bitstream = selectBestMode(rateDyadic,rateSingleMode)
```

3.3.2 Codificador de entropia

O codificador de entropia usado no trabalho foi o codificador aritmético binário adaptativo ao contexto descrito no Capítulo 2, usando 16 bits de precisão, sem nenhum dos códigos usado em outras aplicações para melhoria de desempenho computacional. A ideia proposta para o uso do codificador aritmético no S3D é usar locais de contexto não somente da silhueta atualmente codificada, mas sim estender esse conceito e usar locais de contexto da silhueta codificada previamente no mesmo nível não só no mesmo ponto, mas também em bits adjacentes. Por exemplo se está sendo codificada a silhueta $I^{axis}_{N/2+1,N}$, os locais de contexto utilizados serão os da silhueta $I^{axis}_{1,N/2}$, que foi codificada antes dessa, esses locais de contexto são chamados neste trabalho de contextos 3D. A Figura 3.6 mostra os locais de contexto do codificador S3D, sendo que os locais de contexto da esquerda são aqueles utilizados para codificação da silhueta com todos os pontos, enquanto os locais de contextos das imagens central e direta são os contextos 3D que são usados para codificar o resto das silhuetas e o *pixel* p é o *pixel* sendo codificado.

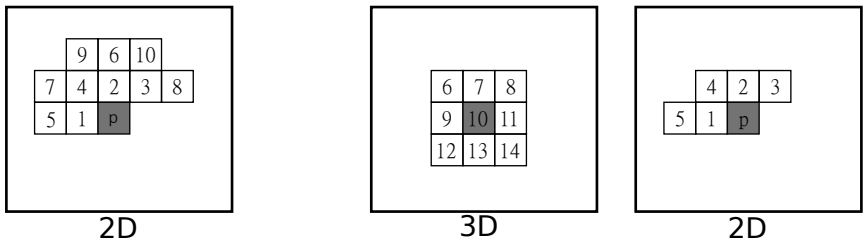


Figura 3.6: Locais de contexto para o S3D: (a) Locais de contexto da silhueta $I^{axis}_{1,N}$, (b) Locais de contexto para as silhuetas restantes.

3.4 OUTROS TRABALHOS EM COMPRESSÃO DE NUVENS DE PONTOS

Muitos outros trabalhos foram propostos na área de compressão de nuvens de pontos, especialmente para a codificação de geometria. O trabalho proposto por Garcia e Queiroz [37] usa o método de escaneamento *octree* descrito anteriormente junto com 2 métodos de compressão entrópica baseada em contextos, uma delas sendo o codificador aritmético e outra um codificador LZW. Os resultados dos 2 métodos propostos para 6 nuvens de pontos foram comparados com o padrão MPEG G-PCC, Codificador aritmético e o LZW sem contextos, e ambos os métodos obtiveram os melhores resultados de taxa, com o codificador LZW com contexto apresentando um desempenho superior ao codificador aritmético com contexto.

Existem também algumas aplicações de codificação com perdas de nuvens de pontos, um exemplo é a desenvolvida por Tang [39], que usa condições de parada para a segmentação *octree*, além de realizar procedimentos de análise estatísticas nas vizinhanças dos pontos divididos, criando máscaras de bits no último nível da *octree* codificado e um algoritmo para detecção e

remoção de *outliers*.

No trabalho de Dricot *et al* [40] é usado um algoritmo de compressão com perdas que usa RDO *Rate Distortion Optimization* capaz de eficientemente criar um escaneamento *octree* adaptado baseado em métricas selecionadas de taxa e distorção, criando um procedimento mais flexível para geração de *octree*.

Já Rente *et al* [41] propõe um codificador de geometria com duas camadas, sendo a primeira (*Base Layer*) um escaneamento *octree* padrão e a segunda (*Enhancement Layer*) é uma transformada *Graph-Based* que possui as funções de base derivadas da estrutura do sinal. Usando essa transformada a energia do sinal é devidamente compactada, deixando-o apto para compressão.

A aplicação descrita por Quach *et al* [42] usa um método de compressão que usa uma rede neural convolucional, esse modelo é treinado usando o *dataset* ModelNet40 e avaliado usando *Microsoft Voxelized Upper Bodies*, e possui bons resultados comparados ao padrão MPEG.

Outro método que também foi usado junto com escaneamento octree foi o TSPLVQ (*Tree-Structured Point-Lattice Vector Quantization*) introduzido em [43], que é a extensão de uma ideia de quantizador vetorial [44] [45] usada previamente em codificação de imagens e vídeos. Esse método procura reduzir ao máximo a quantidade de dados usada para representação de uma nuvem de pontos, considerando a distorção que é gerada pelo nuvem de pontos decodificada.

Daribo [46] propõe um método que explicitamente incorpora uma predição espacial, sendo essas predições curvas no espaço 3D, que são codificadas calculando primeiramente as direções dentro da curva, e, depois codificando-as usando um codificador aritmético. Esse método não faz uso de escanamento *octree* como as aplicações citadas anteriormente.

Outro algoritmo que não faz uso de escaneamento *octree* é o proposto em [47] que faz a representação da nuvem de pontos 3D em um modelo de árvore binária, lidando com os valores de posição dos pontos de maneira mais diretamente, conseguindo lidar com nuvens de pontos não *voxelizadas*.

Uma estratégia de codificação que usa uma Transformada Reversível de Blocos de Autômatos Celulares Hierárquicos foi proposta em [48] que decompõe hierarquicamente os volumes de *voxels*, obtendo bons resultados de compressão além de uma eficiência de representação. Outro trabalho do mesmo autor expande essa ideia [49], lidando também com redundância temporal.

Outro método para codificação de geometria usando correlação temporal, ou seja, codificação *inter-frame*, é o procedimento descrito em [50], que usa uma *octree* de predição O_P, gerada de uma *octree* de referência O_R, para codificação da *octree* a ser codificada O_C, para a codificação da geometria esse trabalho realiza uma operação XOR entre O_P e O_C, o resultado dessa operação é então codificado usado um codificador aritmético. Essa aplicação foi aprimorada por Garcia *et al* em [51], realizando um processo de ordenação ordem crescente em O_P, gerando O_{SP}, essa ordem é passada para O_C, reorganizando o *bitstream* e gerando O_{SC}, que então é codificada entropicamente.

O trabalho proposto em [36] codifica nuvens de pontos usando a um codificador aritmético

baseado em contextos, nesse trabalho 2 métodos para geração de contextos são usados, o primeiro usa a *octree* de nível anterior para gerar os contextos (P(PNI)), enquanto o segundo usa ideia de super resolução, que realiza operações de *downsampling* e verificações de ocupação nas vizinhanças de *voxels* da *octree* de referência O_R (P(SR-NI)), e usa essa informação para codificação dos *voxels* da *octree* O_C a serem codificadas. O trabalho avalia a codificação usando somente um dos dois métodos, além de avaliar também a super resolução com uso de regiões reduzidas(P(SR-Ex)), e finalmente codifica com a versão do proposta do algoritmo que usa todos esses métodos P(Full), usando a distância mínima de Hamming para seleção de método de geração de contextos a ser utilizado, logo após isso o algoritmo é codificado usando o codificador aritmético. Os resultados obtidos são superiores aos outros trabalhos previamente publicados pelos mesmo autores [51] [37].

Outra aplicação mais recente usa um codificador de geometria que aplica uma função volumétrica dentro do conjunto de dados, sendo ele geometria [52] ou atributos [53] sendo que essa função é definida a medida que o espaço é percorrido. Esse método é chamado de representação implícita. A codificação é feita usando octree com algo que ele chama aqui de volumes bézier. Sendo que a *octree* utilizada dentro do algoritmo sofre *pruning* de maneira a retirar as informações de geometria que pouco contribuem, as regiões restantes depois desse procedimento são os volumes bézier, que são codificados usando um codificador aritmético.

Milani *et al* [54] realiza compressão sem perdas primeiro particionando o volume de *voxels* em octantes de tamanho 2x2x2, que são caracterizados usando números inteiros, depois são convertidos em octetos usando um CA *3D Cellular Automata* que é então comprimido usando um codificador aritmético com contextos, os contextos que eles usam são obtidos dos do octetos de cima e da esquerda.

Outro trabalho recente [55] faz uso de algoritmos de aprendizagem profunda para codificação de geometria, separando a nuvem de pontos em diversos blocos 3D, selecionando o modelo de aprendizagem que gera os melhores resultados de compressão para cada um dos blocos individualmente.

3.4.1 Codificação de Atributos

Métodos para compressão de atributos de nuvens de pontos foram desenvolvidos separadamente e aplicados em conjunto com métodos de codificação de geometria, um dos primeiros trabalhos foi proposto em [56], que constrói grafos em pequenas vizinhas da nuvem de pontos, conectando os pontos próximos e tratando os atributos como sinais sobre esses grafos, a transformada de grafos (em inglês *Gaussian Transform* ou GT) é então adotada para descorrelacionar o sinal.

Outro trabalho desenvolvido por Queiroz *et al* [57] introduz a GPT (*Gaussian Process Transform*), que é uma transformada ortogonal linear de sinais definidos por um conjunto finito de pontos, assim como a GT, entretanto, esse trabalho busca abordar um dos casos em que a GT

não obtêm um bom desempenho, que é quando os sinais são definidos em grafos embutidos em domínios euclidianos, tendo um ganho em compactação de energia na ordem de 6 dB.

Um dos trabalhos mais conhecidos em compressão de atributos, usado no padrão G-PCC, é o RAHT [34], que é uma transformada ortogonal adaptativa que pode comprimir sinais de cor de nuvens de pontos, com resultados bastante satisfatórios se comparados à GT. No entanto, é bastante superior em termos de eficiência computacional, podendo ser utilizado inclusive para compressão de cores em tempo real. Essa transformação é hierárquica, semelhante à uma variação adaptativa de uma *wavelet* HAAR [58]. Os coeficientes gerados são então quantizados utilizando um quantizador escalar uniforme e codificados usando um codificador aritmético, que assume distribuições laplacianas.

3.5 CONCLUSÃO DO CAPÍTULO

Neste capítulo foram abordados alguns dos conceitos básicos sobre nuvens de pontos, além de uma breve explicação sobre o padrão MPEG G-PCC [6] e o codificador *Silhouette 3D*, que será a base para o codificador proposto no capítulo 5. No próximo capítulo será detalhado o primeiro algoritmo proposto no trabalho, o seletor de contextos para imagens binárias.

4 ALGORITMO DE SELEÇÃO DE CONTEXTO

O primeiro algoritmo proposto neste trabalho é um algoritmo de pré-processamento de imagens que será usado para otimizar um codificador aritmético padrão com adaptação de contextos, selecionando de maneira independente qual combinação de contextos gera o melhor resultado. Note que, como o codificador é binário, o número de contextos se relaciona com o número de locais de contexto através da equação $N_c = 2^{L_c}$ onde N_c é o número de contextos e L_c é o número de locais de contexto usados. Nesta dissertação, definimos que um local de contexto como a posição de um símbolo previamente codificada, (em uma imagem seria um *pixel* previamente codificado), enquanto um contexto é o valor combinado dos símbolos, assim como descrito na seção 2.3.

O primeiro passo para a criação de um algoritmo de codificação aritmética adaptativo ao contexto é a criação do modelo probabilístico, ou seja, usar os símbolos previamente codificados para estimar as probabilidades do próximo símbolo. A aplicação mais conhecida que usa essa abordagem para imagens binárias é o JBIG [13] discutido na seção 2.3.1. O JBIG usa locais de contexto fixos mostrados na Figura 4.1, sendo que a quantidade desses locais de contexto a ser usada no procedimento pode ser sinalizada na entrada. Caso não seja, 10 locais de contexto são usados por padrão. Entretanto, os locais de contexto usados pelo JBIG são sinalizados apenas pela sua quantidade, por exemplo, se forem selecionados 5 locais de contexto, o algoritmo necessariamente usa os locais 1,2,3,4 e 5 da Figura 4.1 para compressão da imagem.

Para uma primeira verificação de qual seria um número ótimo de locais de contexto para codificação de imagens binárias, comparamos os valores de taxas obtidos ao codificar uma imagem usando de 1 até 16 locais de contexto, na ordem da Figura 4.1, com o valor de entropia, usando um modelo de Markov de ordem k, discutido na seção 2.1.2. Os resultados são mostrados nas Figuras 4.2 e 4.3. Observando os dois gráficos, percebe-se que o valor de entropia sempre decai à medida que o número de contextos aumenta, conforme esperado. Os valores de taxa, no entanto, começam decrescendo a medida em que mais contextos são usados até que um valor mínimo é atingido, e a partir dele os valores de taxa começam a divergir dos valores de entropia. Isso ocorre porque a partir de um certo número de contextos a imagem começa a contabilizar uma grande quantidade de contextos que ocorrem raramente, prejudicando a adaptabilidade das probabilidades a serem usadas no codificador, levando a um aumento na taxa. Outra observação a ser feita é que cada imagem possui uma diferente quantidade ideal de contextos, o que sugere a necessidade um método para determinação da quantidade ideal de contextos para uma melhor compressão.

Nos experimentos mostrados pelas Figuras 4.2 e 4.3, apenas 16 combinações de locais de contexto são usadas: $\{1\}, \{1,2\}, \{1,2,3\}, ..., \{1,2,3,..,16\}$. No entanto existem 65536 composições possíveis para esses 16 locais de contexto, e provavelmente o menor valor mostrado nos gráficos não é o menor valor de taxa possível.

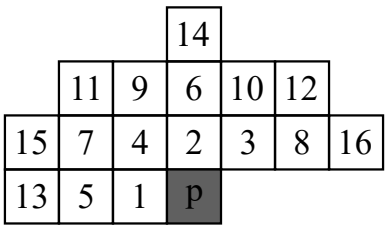


Figura 4.1: Locais de contexto usados no JBIG.

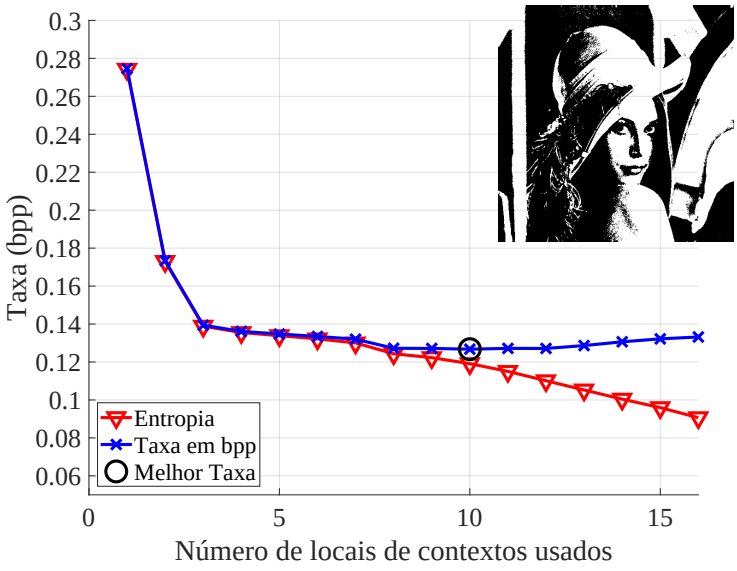


Figura 4.2: Gráficos taxa/entropia x número de contextos para a imagem binarizada *Lena*.

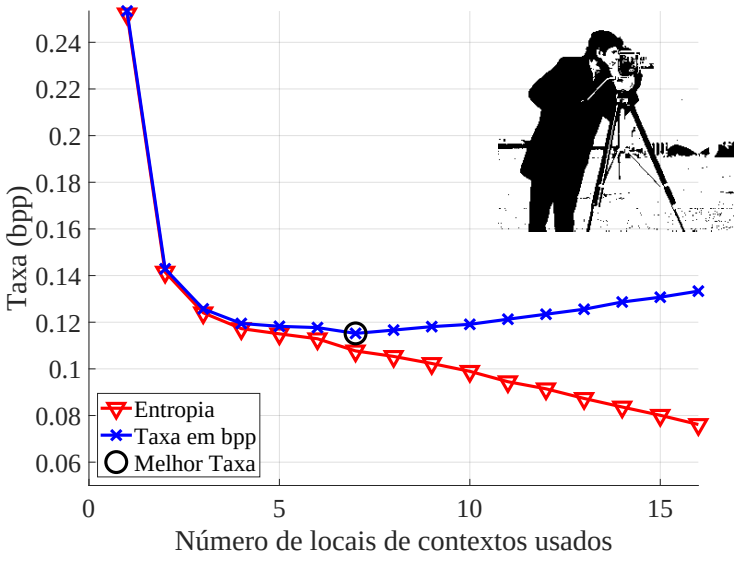


Figura 4.3: Gráficos taxa/entropia x número de contextos para a imagem binarizada *Cameraman*.

O método proposto visa fazer uma seleção de locais de contexto que é completamente independente, diferentemente de alguns trabalhos que foram feitos nessa área para otimização de compressão de textos [28] [29] que usam dependências de contextos. Cada *pixel* pode ser incluído sem depender de nenhum fator calculado dos outros *pixels*. Para isso usamos um conjunto de locais de contexto selecionados, denominado aqui como vetor de contextos, que indica quais locais de contexto estão sendo utilizados naquela codificação.

O primeiro desafio é como implementar esse algoritmo para seleção dos contextos de maneira eficiente. Uma primeira ideia seria simplesmente codificar a imagem usando cada uma das 65536 combinações, salvando o vetor de contexto que gerar a menor probabilidade. No entanto, esse tipo de abordagem é bastante inviável computacionalmente, pois embora codificar uma imagem não seja tão custoso, codificar a mesma imagem dezenas de milhares de vezes acaba gerando bastante custo computacional. Para se resolver esse problema, duas aplicações foram desenvolvidas neste trabalho: o método guloso e o método rápido [10].

4.1 MÉTODO GULOSO

A primeira aplicação, chamada aqui de Método Guloso, em inglês *greedy*, começa codificando a imagem 16 vezes, cada uma usando um dos locais de contexto na Figura 4.1. Depois de codificado, o local de contexto que apresentou a menor taxa é incluído no vetor de contextos. A partir de agora, esse local de contexto sempre será utilizado, e o algoritmo repete o procedimento de codificação, agora 15 vezes, usando o local já marcado em conjunto com os outros. Novamente, aquele que, junto com o local de contexto previamente selecionado obtiver o melhor resultado será marcado. O algoritmo continua dentro desse ciclo, até que a taxa pare de decrescer, que é o que ocorre com as duas imagens estudadas anteriormente, ou então o algoritmo chegue na 16ª iteração, aonde todos os contextos são usados.

Esse algoritmo é bem mais viável do que o citado anteriormente, pois codifica a imagem no mínimo 31 vezes e no máximo $16 + 15 + 14 + ... + 2 + 1 = 136$ vezes, comparado com as 65536 possibilidades da busca completa. No entanto, ainda é necessário codificar a imagem várias vezes, o que pode inviabilizar essa aplicação.

4.2 MÉTODO RÁPIDO (PROPOSTO)

O segundo algoritmo é um algoritmo de duas passagens. Na primeira passagem o algoritmo faz uma leitura prévia da imagem, verificando a ocorrência de todos os contextos possíveis. Como são usados os 16 locais de contexto do JBIG, tem-se 2^{16} (65536) contextos possíveis. O algoritmo então guarda as frequências para estimar $p(x_n|x_{n-1}, x_{n-2}, ..., x_{n-16})$ onde x_n é o símbolo atual e $x_{n-1}, x_{n-2}, ..., x_{n-16}$ são os locais de contexto.

Considere k possíveis locais de contexto de interesse, previamente definidos. Considere também um vetor de contextos $\mathbf{V}$ cujos elementos possam ser os locais de contexto $x_{n-1}, x_{n-2}, ...$ $, x_{n-k}$. No caso de imagens, x_{n-k} corresponde a uma localização espacial calculada a partir de um x_n e mostrado na Figura 4.1. Naturalmente os elementos de $\mathbf{V}$ são únicos. Na i-ésima iteração podemos calcular a entropia de um processo de markov de ordem i como:

$$H_i = H(x_n|\mathbf{V^i}), \tag{4.1}$$

onde $\mathbf{V^i}$ é o vetor de contextos na i-ésima iteração. Por construção, $\mathbf{V^i}$ tem i elementos.

O algoritmo começa com $\mathbf{V^0} = \emptyset$, (isto é, o conjunto vazio), e escolhe o primeiro local de contexto como:

$$\arg\min_m H\left(x_n|x_{n-m}\right) \ . \tag{4.2}$$

O local de contexto escolhido é adicionado ao vetor de contextos:

$$\mathbf{V^1} = \mathbf{V^0} \cup x_{n-m} = x_{n-m} \ . \tag{4.3}$$

No caso geral o algoritmo escolhe um novo local de contexto como:

$$\arg\min_m H\left(x_n|\mathbf{V^{i-1}} \cup x_{n-m}\right), \tag{4.4}$$

e então adiciona este local de contexto ao vetor:

$$\mathbf{V^i} = \mathbf{V^{i-1}} \cup x_{n-m} \ . \tag{4.5}$$

O critério de parada utiliza a entropia da i-ésima iteração H_i definida na Equação (4.1), e é definido pelas equações:

$$\left|\frac{H_i - H_{i-1}}{H_{i-1}}\right| < \left|\frac{H_{i-1} - H_{i-2}}{H_{i-2}}\right|, \tag{4.6}$$

$$|H_i - H_{i-1}| < T_1, \tag{4.7}$$

caso uma delas seja satisfeita, o algoritmo para de obter novos locais de contexto, passando o vetor com os locais de contexto marcados para o codificador aritmético.

A Equação (4.6) verifica em qual ponto a entropia começa a divergir mais lentamente, avaliando a queda percentual do valor de entropia considerando o valor anterior, geralmente no ponto em que a queda percentual começa a subir novamente, significa que os valores de entropia estão decrescendo mais lentamente, que é uma boa estimativa para quando os valores de taxa de compressão começam a subir novamente, como pode ser visto nos gráficos das Figuras 4.2 e 4.3. Já

a Equação (4.7) verifica o valor atual e anterior de entropia, verificando se são menores que um dado limiar T_1 (definido como *0.001 bpp*). Essa segunda condição visa eliminar o caso aonde a primeira condição nunca é atendida, sendo geralmente satisfeita em imagens com baixos valores de taxa.

Os locais de contexto selecionados são transmitidos no cabeçalho do arquivo. O decodificador primeiro interpreta o cabeçalho e depois realiza a decodificação usando aqueles locais de contexto.

4.3 RESULTADOS SELEÇÃO DE CONTEXTOS

Para se verificar a eficácia do algoritmo, utilizamos um banco de imagens de domínio público [59]. Foram comparados: (i) A aplicação do codificador aritmético com uso de contextos em ordem fixa explicada anteriormente de 3 até 11; (ii) JBIG com as configurações padrão; (iii) O algoritmo Guloso descrito anteriormente. Os valores para os locais de contexto 1,2 e 12-16 não são mostrados na tabela devido aos alto valores de taxa encontrados para todas as imagens. Todos os codificadores usados fazem compressão sem perdas e os resultados são exibidos usando a métrica de bits por *pixel* (bpp). As imagens utilizadas são mostradas na Figura 4.4. Estas imagens foram binarizadas antes do processo de compressão usando a função *imbinarize* do MATLAB, que usa o método de Otsu [60].

A Tabela 4.1 mostra o resultado para essas experimentações com 11 imagens, com os melhores valores de taxa sendo mostrados em negrito e os melhores valores de taxa para locais de contexto fixos estando sublinhados. Pelos valores obtidos pode-se perceber a aplicação proposta consegue atingir valores bastante semelhantes à aplicação JBIG, apresentando menor taxa na maioria das figuras. Outra observação é a variabilidade do número ideal de locais de contexto fixos para codificação aritmética padrão, sendo que a quantidade varia entre 7 e 11 para as imagens testadas. O algoritmo que obteve os melhores valores em todas as imagens é o algoritmo Guloso, o que confirma que a proposta de selecionar os locais de contexto individualmente de maneira independente é uma boa estratégia para otimização do codificador aritmético adaptativo ao contexto. O codificador proposto conseguiu atingir a mesma performance do Guloso em 5 das imagens (*Baboon*,*Goldhill*,*Mountain*,*Peppers* e *Zelda*), obteve resultados bem próximos para outras 4 imagens (*Boat*, *Cameraman*, *Sails* e *Watch*). Somente para as imagens de *Barbara* e *Lena* o algoritmo proposto obteve desempenho bastante inferior ao Guloso. Como as taxas absolutas de desempenho na Tabela 4.1 são pequenas, a Tabela 4.3 mostra o ganho relativo percentual do algoritmo proposto, comparando com os valores do JBIG, e do codificador aritmético com 8 e 10 contextos fixos. Esse ganho percentual é calculado como:

$$G_i = \frac{T_{proposto} - T_i}{T_i}, \tag{4.8}$$

com $T_{proposto}$ sendo o valor de taxa do algoritmo proposto para cada imagem, T_i o valor para o método sendo usado como âncora e G_i o ganho percentual.

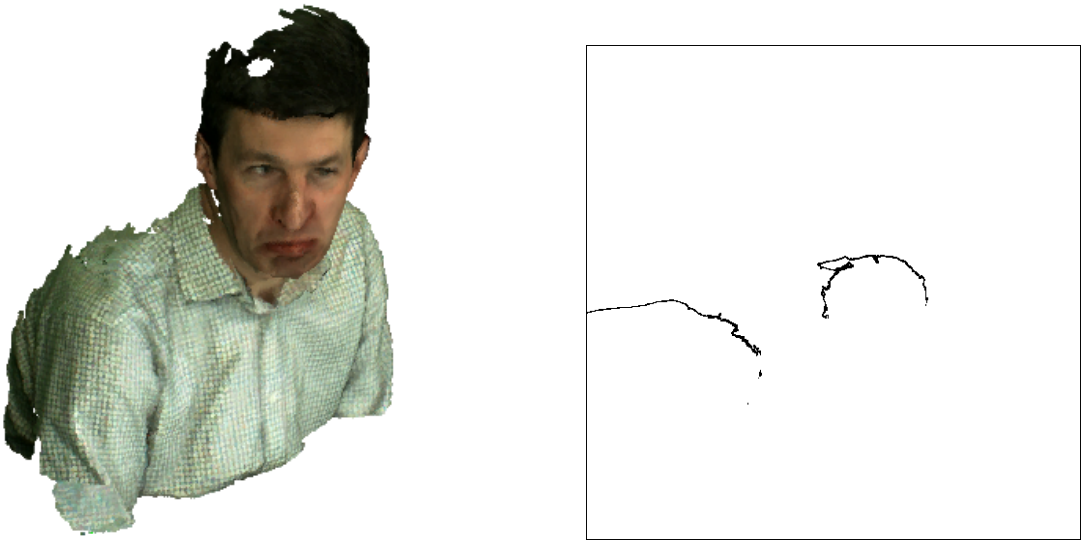
O codificador proposto aqui também foi usado para codificação de cortes de nuvens de pontos, sendo um corte uma visão 2D de uma das posições do terceiro eixo, pois foi reportado por [8] que o padrão JBIG possui um desempenho bastante ruim para esse tipo de imagens, por causa da grande esparsidade inerente dessas imagens. O *dataset* usado aqui é o *Microsoft Upper Bodies* [33]. Todas as nuvens de pontos deste *dataset* são voxelizadas. Essas silhuetas são codificadas usando todos os métodos citados anteriormente, com a exceção do método Guloso, devido a alta complexidade computacional para codificar cada silhueta. Todas as silhuetas são codificadas pelo algoritmo, exceto aquelas em que todos os valores são 0. As nuvens de pontos utilizadas, em conjunto com uma silhueta de um corte na metade do eixo x, são mostradas nas Figuras 4.5 até 4.9.

Os resultados obtidos são apresentados na Tabela 4.2, sendo o ganho relativo percentual do método proposto em comparação com o codificador aritmético com 4, 8 e 10 contextos e o JBIG, mostrado na Tabela 4.4, usando a Equação 4.8 para cálculo dos ganhos. Os valores absolutos são reportados segundo o padrão de codificação de nuvens de pontos em bit por *voxel* ocupado (*bit per occupied voxel*). Novamente os resultados em negrito indicam o melhor valor de taxa, enquanto os sublinhados são os melhores valores para o codificador aritmético padrão.

A principal ideia por trás dessa aplicação em silhuetas de nuvens de pontos é avaliar a eficácia de usar o codificador aritmético padrão em comparação com o JBIG, e como pode-se perceber pelas Tabelas 4.2 e 4.4, pode-se concluir que o JBIG possui resultados de taxa bastante inferiores ao codificador aritmético padrão. Outro detalhe é a maior variabilidade do número ótimo de contextos para o codificador padrão indo de 3 contextos para o *Ricardo9* até 8 para o *David9* [33]. O algoritmo proposto obteve os melhores resultados para todas as nuvens de pontos avaliadas, com a exceção do Ricardo9 com 3 contextos, sendo o valor obtido de taxa bastante próximo.

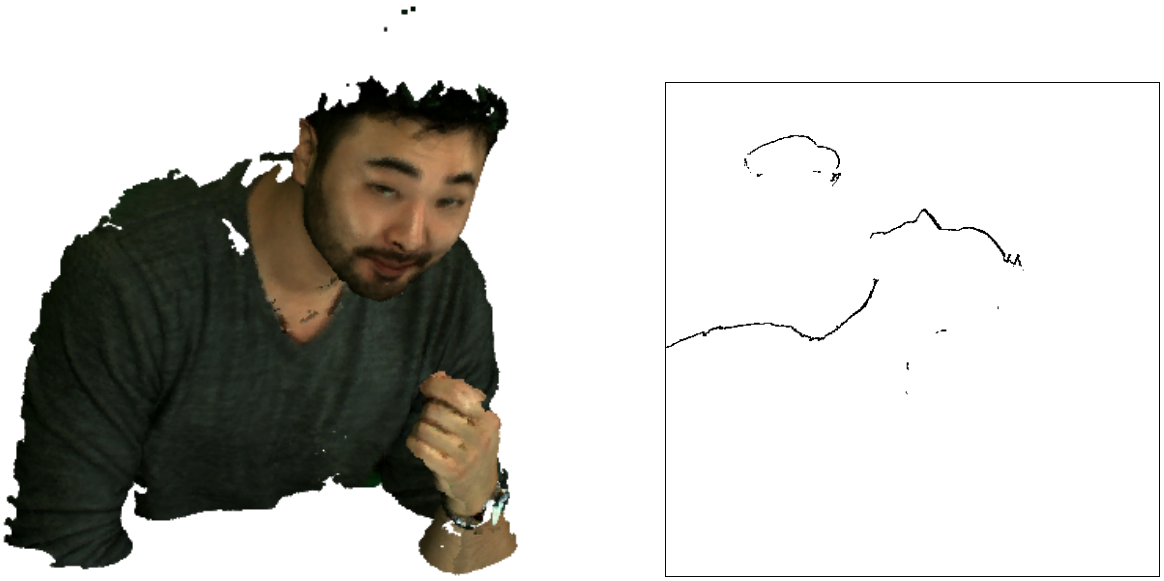
O algoritmo foi codificado usando o MATLAB, e segundo as análises feitas durante o processo de codificação, o codificador aritmético com a seleção de contextos é somente 4% mais lento do que o codificador com contextos fixos, sendo a razão o processo de pré-seleção descrito anteriormente estar pouco otimizado. Mesmo assim, o procedimento dura em média 2 segundos, considerando um computador Windows com processador Intel Core i7-7500U com 8GB of RAM, indicando a baixa complexidade do algoritmo.

Figura 4.4: Imagens binarizadas usadas para avaliar o codificador: (a) *Baboon* 512×512 (b) *Barbara* 512×512 (c) *Boat* 512×512 (d) *Cameraman* 256×256 (e) *Goldhill* 512×512 (f) *Lena* 512×512 (g) *Mountain* 480×640 (h) *Peppers* 512×512 (i) *Sails* 512×768 (j) *Watch* 768×1024 (k) *Zelda* 512×512.



(a) (b)

Figura 4.5: (a) Nuvem de pontos *Andrew9*; (b) Silhueta de *Andrew9* no eixo x.



(a) (b)

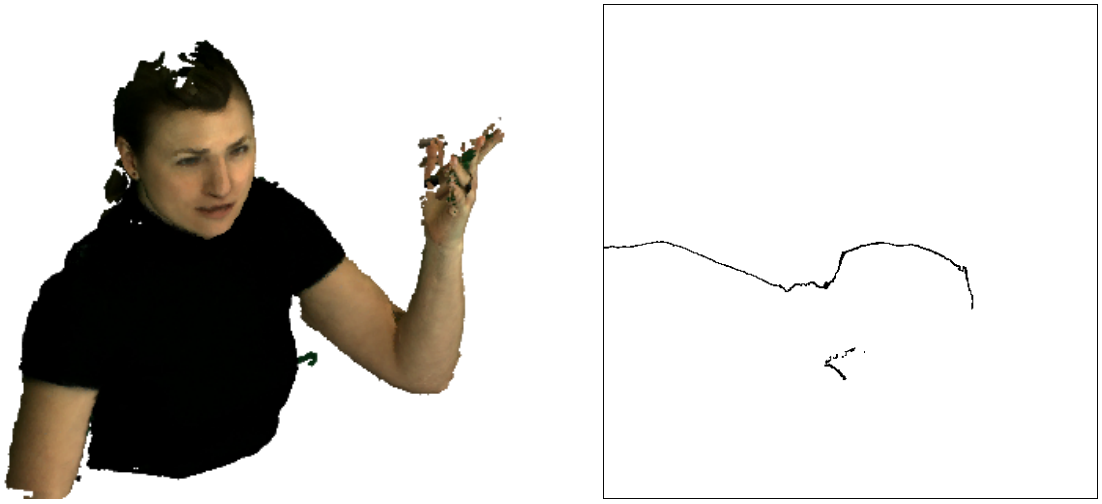
Figura 4.6: (a) Nuvem de pontos *David9*; (b) Silhueta de *David9* no eixo x.

(a) (b)

Figura 4.7: (a) Nuvem de pontos *Phil9*; (b) Silhueta de *Phil9* no eixo x.

(a) (b)

Figura 4.8: (a) Nuvem de pontos *Ricardo9*; (b) Silhueta de *Ricardo9* no eixo x.



(a) (b)

Figura 4.9: (a) Nuvem de pontos *Sarah9*; (b) Silhueta de *Sarah9* no eixo x.

Tabela 4.1: Resultados de taxa para as imagens naturais binárias. Todas as taxas estão em bits por *pixel*(bpp).

	Codificador usando número fixo de contextos											
Imagem	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**JBIG**	**Guloso**	**Proposto**
Baboon	0.5029	0.4964	0.4941	0.4894	0.4870	0.4829	0.4812	0.4808	0.4820	0.4898	**0.4776**	**0.4776**
Barbara	0.2718	0.2562	0.2503	0.2441	0.2350	0.2313	0.2285	0.2216	0.2209	0.2165	**0.2153**	0.2281
Boat	0.1339	0.1294	0.1279	0.1278	0.1247	0.1233	0.1241	0.1252	0.1271	0.1242	**0.1225**	0.1238
Cameraman	0.1257	0.1195	0.1182	0.1180	0.1152	0.1166	0.1181	0.1191	0.1213	0.1211	**0.1151**	0.1160
Goldhill	0.2024	0.1962	0.1912	0.1868	0.1847	0.1807	0.1792	0.1801	0.1812	0.1807	**0.1780**	**0.1780**
Lena	0.1394	0.1361	0.1348	0.1335	0.1320	0.1272	0.1271	0.1268	0.1272	0.1355	**0.1255**	0.1320
Mountain	0.4135	0.3985	0.3883	0.3830	0.3798	0.3749	0.3733	0.3718	0.3717	0.3825	**0.3698**	**0.3698**
Peppers	0.1256	0.1185	0.1151	0.1141	0.1128	0.1097	0.1087	0.1085	0.1088	0.1130	**0.1075**	**0.1075**
Sails	0.1958	0.1921	0.1913	0.1893	0.1868	0.1817	0.1805	0.1793	0.1794	0.1838	**0.1761**	0.1763
Watch	0.1281	0.1258	0.1254	0.1224	0.1214	0.1182	0.1152	0.1147	0.1147	0.1133	**0.1121**	0.1171
Zelda	0.1274	0.1248	0.1214	0.1211	0.1203	0.1176	0.1178	0.1186	0.1200	0.1218	**0.1161**	**0.1161**
Média	0.2151	0.2085	0.2053	0.2027	0.2000	0.1967	0.1958	0.1951	0.1958	0.1984	**0.1923**	0.1947

Tabela 4.2: Resultados de taxa para os cortes de nuvens de pontos. Todos os resultados estão em bits por *voxel* ocupado (bpov).

	Codificador usando número fixo de contextos										
Imagem	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**JBIG**	**Proposto**
Andrew9	1.955	1.940	1.933	1.930	1.981	1.973	2.050	2.130	2.220	2.752	**1.904**
David9	2.037	2.024	2.015	2.013	2.066	2.008	2.087	2.171	2.264	2.666	**1.935**
Phil9	1.984	1.981	1.970	1.956	1.999	1.970	2.037	2.104	2.182	2.705	**1.911**
Ricardo9	**1.915**	1.921	1.930	1.932	1.988	1.998	2.080	2.166	2.253	3.119	1.916
Sarah9	2.025	2.018	2.015	2.011	2.070	2.033	2.111	2.193	2.287	2.886	**1.939**
Média	1.983	1.977	2.064	2.060	2.122	2.089	2.164	2.244	2.333	2.826	**1.921**

Tabela 4.3: Ganhos em porcentagem do método proposto em comparação com outras abordagens em imagens(8, 10 e JBIG).

	Ganhos do proposto sobre		
Imagem	**8**	**10**	**JBIG**
Baboon	-1.1%	-0.7%	-2.5%
Barbara	-1.4%	2.9%	5.4%
Boat	0.4%	-1.1%	-0.3%
Cameraman	-0.5%	-2.6%	-4.2%
Goldhill	-1.5%	-1.2%	-1.5%
Lena	3.8%	4.1%	-2.6%
Mountain	-1.4%	-0.5%	-3.3%
Peppers	-2.0%	-0.9%	-4.9%
Sails	-3.0%	-1.7%	-4.1%
Watch	-0.9%	-2.1%	3.4%
Zelda	-1.3%	-2.1%	-4.7%
Média	-0.8%	-0.5%	-1.8%

Tabela 4.4: Ganhos em porcentagem do método proposto em comparação com outras abordagens em nuvens de pontos(4, 8, 10 e JBIG).

	Ganhos do proposto sobre			
Imagem	**4**	**8**	**10**	**JBIG**
Andrew9	-1.9%	-3.5%	-10.6%	-30.8%
David9	-4.4%	-3.6%	-10.9%	-27.4%
Phil9	-3.5%	-3.0%	$-\ 9.2\%$	-29.4%
Ricardo9	-0.2%	-4.1%	-11.5%	-38.6%
Sarah9	-3.9%	-4.6%	-11.6%	-32.8%
Média	-2.8%	-3.8%	-10.8%	-31.8%

4.4 CONCLUSÃO DO CAPÍTULO

Neste capítulo apresentamos um algoritmo para seleção de contextos para codificação de imagens binárias. Os testes mostraram que o algoritmo proposto tem um bom desempenho na codificação de imagens naturais, e um excelente desempenho na codificação de silhuetas geradas a partir de nuvens de pontos. No próximo capítulo apresentaremos um codificador de nuvens de pontos dinâmicas, e depois discutiremos como o algoritmo de seleção de contextos pôde ser adaptado para a codificação de nuvens de pontos.

5 *SILHOUETTE 4D* - COMPRESSÃO DE NUVENS DE PONTOS DINÂMICAS

Quando consideramos a compressão de nuvens de pontos dinâmicas, podemos utilizar um quadro de referência, previamente codificado, para reduzir a taxa do quadro atual. Aqui desenvolvemos dois algoritmos, baseados no algoritmo *Silhouette 3D* [9]. *Silhouette 4D* (S4D) [11] e o *Silhouette 4D with Context Selection* (S4D-CS) [12].

5.1 *SILHOUETTE 4D*

O segundo algoritmo desenvolvido neste trabalho é o *Silhouette 4D* (S4D) [11]. Esse algoritmo é uma extensão da ideia já usada de representar nuvens de pontos como uma série de silhuetas e locais de contexto de outra imagem binária para codificação, criando uma aplicação com codificação *inter-frame* para compressão de nuvens de pontos.

A estrutura básica do código, descrita no Algoritmo 2, é semelhante à do S3D descrito anteriormente. O algoritmo começa gerando a primeira silhueta com todos os pontos (linha 3), depois codificando-a (linha 5). Depois de codificada, a Decomposição diádica é feita, criando 2 silhuetas uma para cada metade do eixo (linhas 7 e 9). Máscaras são criadas de maneira a otimizar a codificação (linhas 11-12). O processo de compressão é então realizado (linhas 16 e 20). Por fim o processo é aplicado recursivamente para codificação dos demais níveis (linhas 21-23). No final o algoritmo codifica usando a Divisão em silhuetas unitárias (linhas 24 e 25) caso a condição seja satisfeita e compara quantos bits foram gastos usando ambos os métodos, Decomposição diádica ou Divisão em silhuetas unitárias, salvando no *bitstream* final aquele com o menor valor de taxa (linha 26).

As linhas descritas acima fazem menção apenas as funções originais do S3D, no entanto, 2 modificações foram aplicadas em cima desse algoritmo, de maneira a otimizar o processo de codificação: a adição do contexto 4D e do Modo multi que serão comentadas posteriormente.

5.1.1 Estendendo a aplicação de 3D para 4D

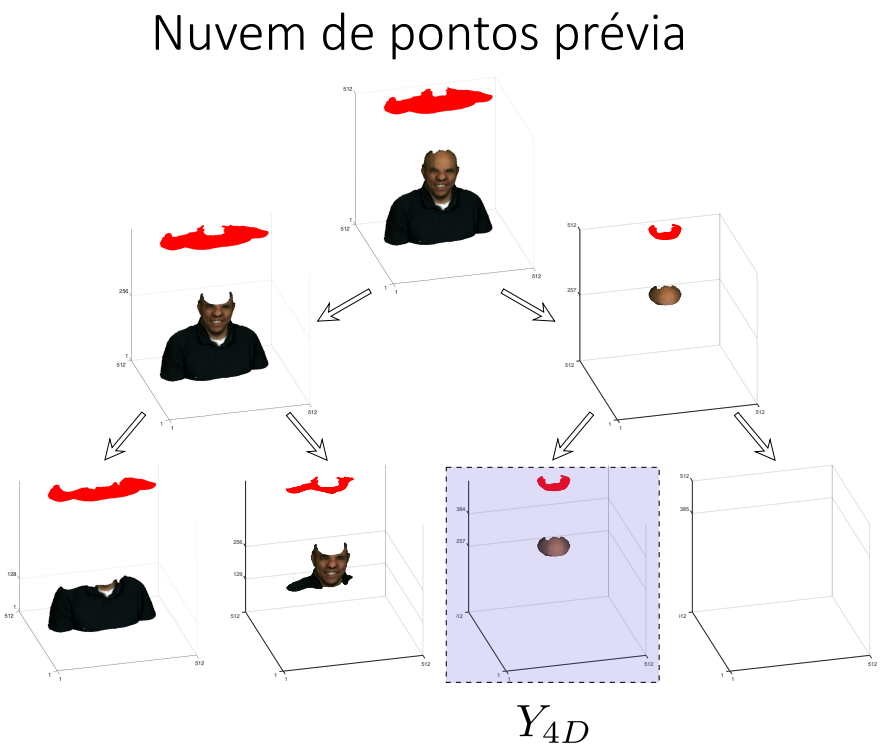
Para que se torne viável fazer usos de locais de contexto da nuvem de pontos previamente codificada, ela precisa primeiro ser adicionada no codificador, realizando o processo de geração de silhuetas de maneira semelhante à nuvem de pontos sendo codificada (linhas 4,8 e 10), como mostrado na Figura 5.1.

Algorithm 2: S4DCode()

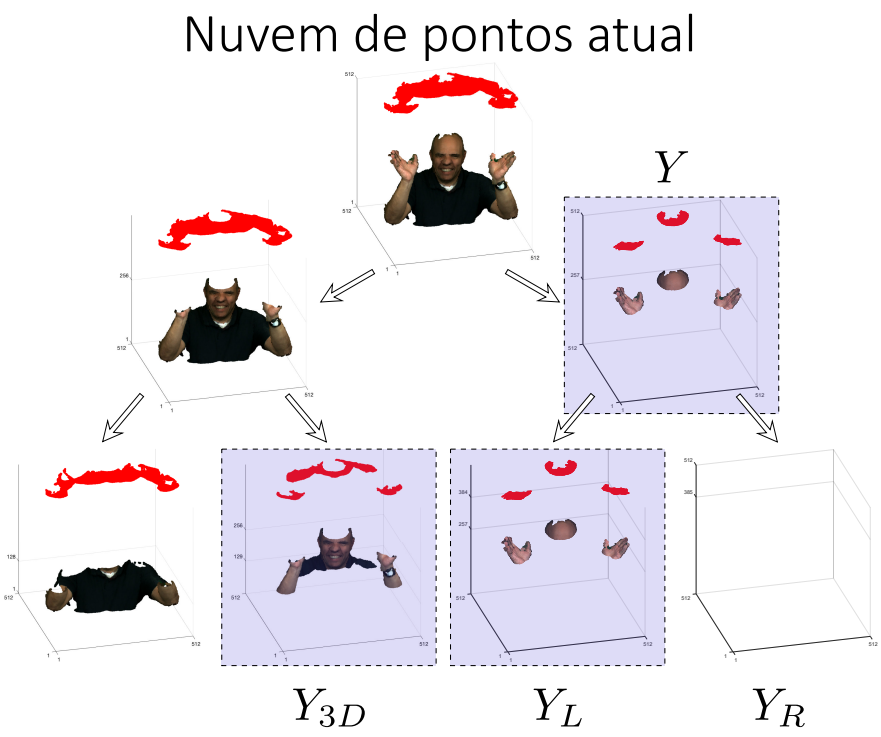
Input: $\mathbf{G}$: Matriz de ocupação.
$\mathbf{pG}$: Matriz de ocupação da nuvem de pontos previamente codificada.
$axis$: O eixo em que a divisão ocorre
n,m : Os intervalos inferior e superior
I : A silhueta do nível anterior
$Testa3D$: Parâmetro que indica se o Modo multi é ou não usado
$ModoFast$: Parâmetro que indica se o Modo multi rápido é ou não usado
P : Tamanho do intervalo para uso da Divisão em silhuetas unitárias

Output: $bitstream$: O arquivo com a representação compacta de G

```
1  begin
2      if I Não for passado como argumento then
3          I = silhouette(G,axis,n,m)
4          pI = silhouette(pG,axis,n,m)
5          encodeFirstImage(I,pI)
6      mid = ((m - n + 1) / 2) + n - 1
7      I_L = silhouette(G, axis, n, mid)
8      pI_L = silhouette(pG, axis, n, mid)
9      I_R = silhouette(G, axis, mid+1, m)
10     pI_R = silhouette(pG, axis, mid+1, m)
11     mIleft = I
12     mIRight = and(I,I_L)
13     if Testa3D e ModoFast then
14         encodeSilhouetteFast(I_L,mIleft,pI_L)
15     else
16         encodeSilhouette(I_L,mIleft,pI_L,Testa3D)
17     if Testa3D e ModoFast then
18         encodeSilhouetteFast(I_R,mIRight,pI_R)
19     else
20         encodeSilhouette(I_R,mIRight,pI_R,Testa3D)
21     if mid > n then
22         S4DCode(G, axis, n, mid, I_L)
23         S4DCode(G, axis, mid + 1, m, I_R)
24     if (n - m + 1) <= P then
25         singleModeEnc(G, axis, n, m, I)
26     bitstream = selectBestMode(rateDyadic,rateSingleMode)
```



(a)



(b)

Figura 5.1: Decomposição diádica S4D: (a) Nuvens de pontos prévia (b) Nuvens de pontos atual.

O processo de codificação de imagens é semelhante ao descrito no S3D. A principal diferença aqui é o uso de um conjunto diferente locais de contexto para codificação das silhuetas agora, 3 locais de contexto são retirados da própria silhueta (contextos 2D), 9 da silhueta previamente codificada (contextos 3D), e 1 local de contexto da silhueta da nuvem de pontos anterior com a mesma posição do *pixel* sendo codificado(o novo contexto 4D). Esses locais de contexto são mostrados na Figura 5.2.

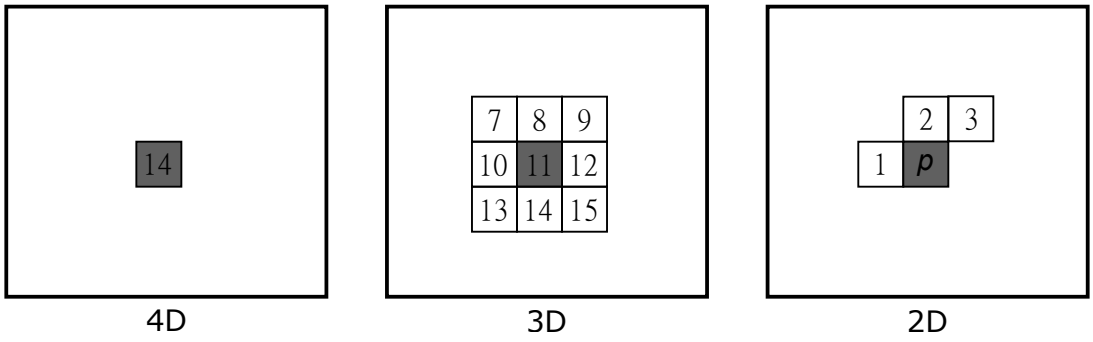


Figura 5.2: Locais de contexto usados no S4D.

5.1.2 Modo multi

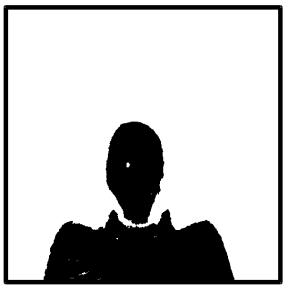
Foi observado durante o teste do algoritmo do S4D que algumas nuvens de pontos analisadas não obtiveram ganhos com relação ao S3D, e algumas inclusive obtiveram pior desempenho. Para resolver esse problema, foram adicionados os locais de contexto do S3D. Nessa solução o Modo multi (*Multi mode*), o algoritmo codifica a silhueta usando tanto o arranjo de locais de contexto do S3D quanto o do S4D, procedimento feito nas linhas 14 e 18 do algoritmo, e aquele que gerar o melhor resultado será adicionado e sinalizado no *bitstream* final. Ambos os contextos são atualizados independentemente de qual será escolhido.

5.1.2.1 Modo multi rápido

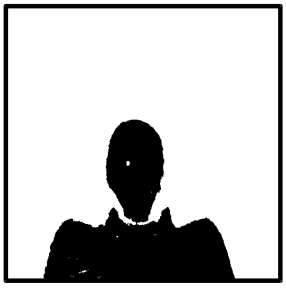
Um problema com essa abordagem é que ela possui um grande custo computacional, pois cada silhueta é codificada duas vezes.

Uma alternativa desenvolvida para seleção é o uso de um algoritmo pré-processamento, chamado aqui de Modo multi rápido (*Multi mode fast*) e mostrado nas linhas 15 e 21 do algoritmo, que consiste em percorrer toda a silhueta sendo codificada, anotando os valores dos *pixels*, computando as probabilidades do *pixel* ser 1 tanto para os contextos 3D como 4D. Após todo o pré-processamento, o algoritmo realiza uma predição sobre o valor de cada *pixel* da silhueta, considerando as probabilidades obtidas para cada contexto. Por exemplo se o contexto para um dado bit indica 80% de probabilidade do valor do bit ser 0, o algoritmo prediz que ele será 0. As duas predições com os dois conjuntos de locais de contexto são verificadas, e aquela que possuir a menor quantidade de *pixels* diferentes do valor original da silhueta é aquela que será utilizada na codificação Essa opção é sinalizada ao decodificador.

Um exemplo descrevendo essa aplicação é mostrado na Figura 5.3, onde a imagem de cima é a imagem real a ser codificada, a imagem do meio é a imagem predita pelo codificador usando contextos 3D e a inferior usando contextos 4D. Para essa silhueta, o codificador 3D errou em 708 bits enquanto o 4D errou em 473, logo, o codificador com contextos 4D será escolhido. Neste exemplo, isto se provou uma escolha acertada, pois para se codificar a silhueta usando os contextos 4D são gastos 2145 bits, ao contrário dos 3136 que seriam usados caso fossem escolhidos usar os contextos 3D.

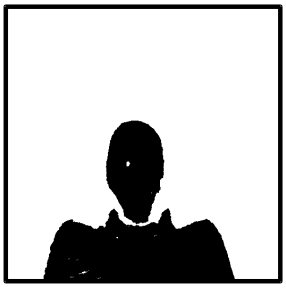


Silhueta a ser codificada
64566 símbolos precisam ser transmitidos



Predição com locais de contextos 3D

708 bits diferentes
3136 bits usados



Predição com locais de contextos 4D

473 bits diferentes
2145 bits usados

Figura 5.3: Exemplo do procedimento Modo multi rápido.

5.1.3 Resultados S4D

Uma primeira experimentação foi feita com o S4D, de maneira a avaliar o desempenho do codificador S4D padrão com o S3D, o Modo multi, e o Modo multi rápido. Foram codificados as primeiras 10 nuvens de pontos para cada uma das sequências do *8i Voxelized Full Bodies* [61] e

os resultados obtidos são mostrados na Tabela 5.1, todos em bpov.

Para 3 das sequências (*LongDress, Loot, RedandBlack*), o uso de locais de contexto *intra-frame* somente apresentaram melhores resultados, comparados com a aplicação S4D padrão, no entanto, para uma das sequências (*Soldier*), o uso dos contextos *inter-frame* apresentou uma melhora significativa. Os melhores resultados obtidos foram do Modo multi. Tal comportamento já era esperado, devido ao fato de que esse modo codifica previamente as silhuetas, a fim de determinar qual das composições de locais de contexto é a melhor. Já a aplicação Modo multi rápido conseguiu obter resultados semelhantes, possuindo uma complexidade semelhante aos codificadores que só avaliam locais de contexto *intra-frame* ou *inter-frame*, devido ao fato de que ele codifica as silhuetas uma vez apenas, fazendo o processo de seleção de composição de locais de contexto antes. Por ser a aplicação com a melhor relação entra valores de taxa e complexidade, o algoritmo S4D proposto desse trabalho é a aplicação Modo multi rápido.

O desempenho do S4D proposto é comparado com os algoritmos: (i) *Octree* Original; (ii) MPEG G-PCC TMC13 v7.0; (iii) o algoritmo *Silhouette 3D*; (iv) o P(Full) (*Parent Node Inheritance plus Super-Resolution*); (v-vii) o algoritmo proposto *Silhouette 4D*, testando somente um dos eixos (X,Y e Z respectivamente), dessa vez codificando 100 nuvens de pontos para cada sequência. Os *datasets* utilizados para avaliação do S4D foram o *Microsoft Voxelized Upper Bodies* [33], mais especificamente as sequências de nuvens de pontos de 9 bits *(Andrew9, David9, Phil9, Ricardo9 e Sarah9)* e o *8i Voxelized Full Bodies* [61] usando as sequências *(Longdress, Loot, RedandBlack e Soldier)* ambos os datasets estão disponíveis em [62], sendo que todos os valores obtidos foram computados em (bpov).

Analisando os valores da aplicação proposta com o S4D com uso de eixo fixo (X,Y ou Z), percebe-se que há um pequeno ganho em se realizar a seleção de eixos: 3.4% para o eixo X, 1.1% para o eixo Y e 2.9% para o eixo Z, em troca de um razoável custo computacional. Uma comparação direta de valores de taxa absolutos do S4D com o TMC13 e o codificador *inter-frame* P(Full), mostra que a aplicação proposta possui valores de taxas melhores para todas as sequências, sendo a sequência *Soldier*, aquela com maior ganho sobre o TMC13. Curiosamente o codificador *inter-frame* P(Full) também obteve resultados satisfatórios para essa sequência, o que indica que essa sequência de nuvens de pontos é uma das mais adequadas para se realizar codificação *inter-frame*. Comparando os valores da aplicação anterior S3D com o algoritmo proposto, também foi obtido ganho para todas as sequências, o que já era esperado devido a capacidade do algoritmo de escolher dinamicamente qual conjunto de contextos é o mais adequado.

5.2 *SILHOUETTE 4D* COM SELEÇÃO DE CONTEXTOS

Ambas as codificações de silhuetas propostas anteriormente, S3D e S4D, faziam uso de locais de contexto fixos para a codificação de todas as nuvens de pontos. O último trabalho proposto visa adicionar um algoritmo para seleção de contextos dentro do codificador S4D descrito ante-

Tabela 5.1: Comparação dos métodos propostos. Todas as taxas estão em bits por *voxel* ocupado (bpov).

Sequência	3D	4D	4D + 3D	4D + 3D fast
Longdress	0.95	1.03	0.95	0.95
Loot	0.91	0.95	0.89	0.90
RedandBlack	1.02	1.07	1.01	1.01
Soldier	0.96	0.79	0.79	0.80
Média	0.96	0.96	0.91	0.91

Tabela 5.2: Comparação com algoritmo de Estado da Arte. Todas as taxas estão em bits por *voxel* ocupado (bpov).

		Codificadores *intra-frame*			Codificadores *inter-frame*					Ganho do S4D sobre		
Banco de Dados	Sequência	**Octree**	**TMC13**	**S3D**	**P(Full)**	**P-X**	**P-Y**	**P-Z**	**Proposto**	**TMC13**	**S3D**	**P(Full)**
Microsoft Voxelized Upper Bodies [33]	*Andrew9*	2.58	1.14	1.12	1.37	1.09	1.10	1.08	1.08	−5.7%	−4.0%	−21%
	David9	2.62	1.07	1.06	1.31	1.09	1.06	1.05	1.05	−2.7%	−1.1%	−20%
	Phil9	2.64	1.18	1.14	1.42	1.18	1.14	1.13	1.13	−4.1%	−1.2%	−21%
	Ricardo9	2.59	1.10	1.04	1.34	1.05	1.03	1.03	1.02	−6.8%	−1.9%	−24%
	Sarah9	2.61	1.08	1.07	1.37	1.08	1.05	1.05	1.04	−3.5%	−2.8%	−24%
	Média	2.61	1.11	1.09	1.36	1.09	1.07	1.07	1.06	−4.6%	−2.8%	−22%
8i Voxelized Full Bodies [61]	*Longdress*	2.99	1.03	0.95	1.13	0.97	0.97	1.00	0.95	−7.5%	0.0%	−15.5%
	Loot	2.98	0.97	0.92	1.02	0.95	0.92	0.95	0.91	−6.0%	−0.5%	−11.1%
	RedandBlack	3.00	1.10	1.02	1.23	1.02	1.04	1.06	1.02	−7.2%	−0.1%	−17.0%
	Soldier	3.00	1.04	0.96	0.85	0.89	0.81	0.89	0.81	−22.8%	−15.9%	−5.6%
	Média	2.99	1.03	0.96	1.06	0.96	0.93	0.98	0.92	−10.9%	−4.2%	−12.3%

riormente de maneira a obter ganhos nas taxa de compressão previamente obtidas. O algoritmo 3 mostra o procedimento que é feito pelo S4D com seleção de contextos, a maioria do código é bem similar ao S4D, com algumas modificações. A primeira é o procedimento feito pelas funções nas linhas 2 e 3, que é o procedimento para seleção de locais de contexto que será explicado na seção seguinte, a segunda diferença é a exclusão do Modo multi desenvolvido no S4D, e a terceira é que as funções responsáveis pela codificação agora devem considerar os arranjos de locais de contexto previamente selecionados, que são diferentes para cada nuvem de pontos codificada. Já os locais de contexto elegíveis para serem selecionados neste codificador são 24, sendo 6 locais de contexto 2D, 9 locais de contexto 3D e 9 locais de contexto 4D, mostrados na Figura 5.4.

Aqui, usamos o algoritmo de seleção de contextos descrito no capítulo 4. Como o método guloso seria inviável, pois demandaria a codificação da nuvem de pontos dezenas de vezes, usou-se o método proposto na Seção 4.2, que utiliza a entropia de um processo de markov para realizar a seleção dos locais de contexto. Da mesma forma que ocorreu que ocorreu quando aplicamos o algoritmo na codificação de imagens, se formos levar em conta o comportamento da entropia, os valores irão sempre decair a medida que mais contextos são adicionados, o que na teoria significaria que usar o número máximo de contextos é sempre melhor. No entanto, como estes contextos são aplicados em um codificador aritmético, o codificador fica com uma quantidade

Algorithm 3: S4DCSCode()

Input: $\mathbf{G}$: Os pontos da nuvem de pontos sendo codificada.
$\mathbf{pG}$: Os pontos da nuvem de pontos previamente codificada.
$axis$: O eixo em que a divisão ocorre
n,m : Os intervalos inferior e superior
I : A silhueta do nível anterior
P : Tamanho do intervalo para uso da Divisão em silhuetas unitárias

Output: $bitstream$: O arquivo com a representação compacta de G

```
1  begin
       // Realiza a seleção de contextos
2      structTables = createContextTableInter(G)
3      structVector = generateAllContextVector(G,structTables)
4      if I Não for passado como argumento then
5          I = silhouette(G,axis,n,m)
6          pI = silhouette(pG,axis,n,m)
7          encodeFirstImage(I,pI)
8      mid = ((m - n + 1) / 2) + n - 1
9      I_L = silhouette(G, axis, n, mid)
10     pI_L = silhouette(pG, axis, n, mid)
11     I_R = silhouette(G, axis, mid+1, m)
12     pI_R = silhouette(pG, axis, mid+1, m)
13     mIleft = I
14     mIRight = and(I,I_L)
15     encodeSilhouette(I_L,mIleft,pI_L,structVector)
16     encodeSilhouette(I_R,mIRight,pI_R,structVector)
17     if mid > n then
18         S4DCSCode(G, axis, n, mid, I_L)
19         S4DCSCode(G, axis, mid + 1, m, I_R)
20     if (n - m + 1) <= P then
21         singleModeEnc(G, axis, n, m, I,structVector)
22     bitstream = selectBestMode(rateDyadic,rateSingleMode)
```

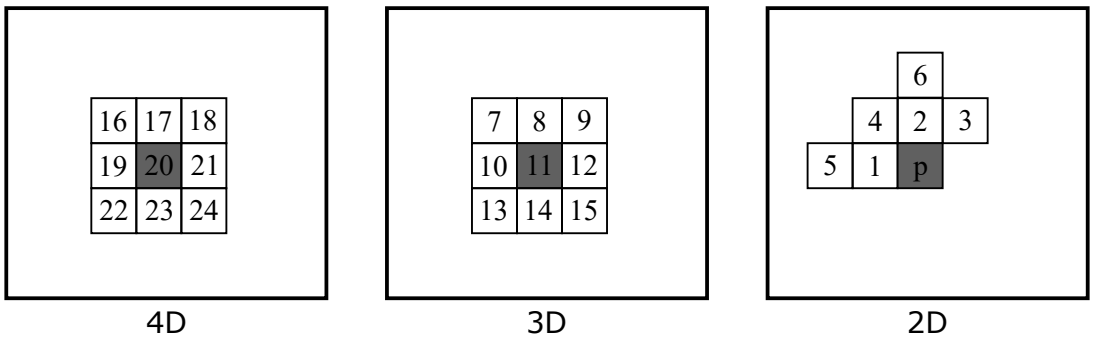


Figura 5.4: Locais de Contexto usados no S4D-CS.

imensa de contextos que não se adaptam adequadamente, prejudicando a eficácia da compressão. Esse processo pode ser visto na Figura 5.5, que avalia os valores de taxa para os primeiro quadro dos *datasets* de nuvem de pontos *Ricardo9* e *Longdress*. Pode-se então perceber que a medida em que mais contextos são adicionados a taxa decai, até um certo número de locais de contexto em que ela começa a subir novamente.

O processo de seleção começa então fazendo a primeira passagem na nuvem de pontos, de maneira a gerar o modelo probabilístico para cada possível contexto. Uma pequena mudança feita nessa versão do algoritmo, é que todos os contextos agora possuem probabilidade 50% para o valor 0 ou 1.

O algoritmo de seleção de contextos usado tem apenas duas diferenças com relação ao algoritmo apresentado na seção tal. A primeira, naturalmente, é que o algoritmo usa os locais de contexto da Figura 5.4. A segunda diferença é o critério de parada. Aqui, usamos o número de contextos no vetor de contextos como critério de parada, isto é, novos locais de contexto são adicionados até que 13 locais de contexto tenham sido adicionados, para nuvens de pontos de 9 bits, e 14 locais de contexto, para nuvens de pontos de 10 bits.

Uma versão simplificada também foi proposta, visando diminuir um pouco o custo computacional do código. Nessa versão, para a análise de seleção de contextos, a Decomposição diádica é feita até um certo nível, nas experimentações foi-se definido 4 como um bom estimador, reduzindo bastante o número de silhuetas a serem analisadas, pois agora são analisadas somente 31 silhuetas em contraste com as 1023 (para nuvens de pontos de 9 bits) ou 2047 (para nuvens de pontos de 10 bits), essa versão do código é chamada aqui de *S4D-CSPrune*, enquanto a versão do código que analisa toda a nuvem de pontos é a *S4D-CSFull*.

Depois da seleção, os vetores de contextos são armazenados na estrutura utilizada para a codificação das nuvens de pontos e são passados para o algoritmo do S4D, que dentro das funções de codificação aritmética, irá considerar somente esses locais de contexto. Esses locais de contexto são transmitidos para o decodificador, dentro do cabeçalho do arquivo final.

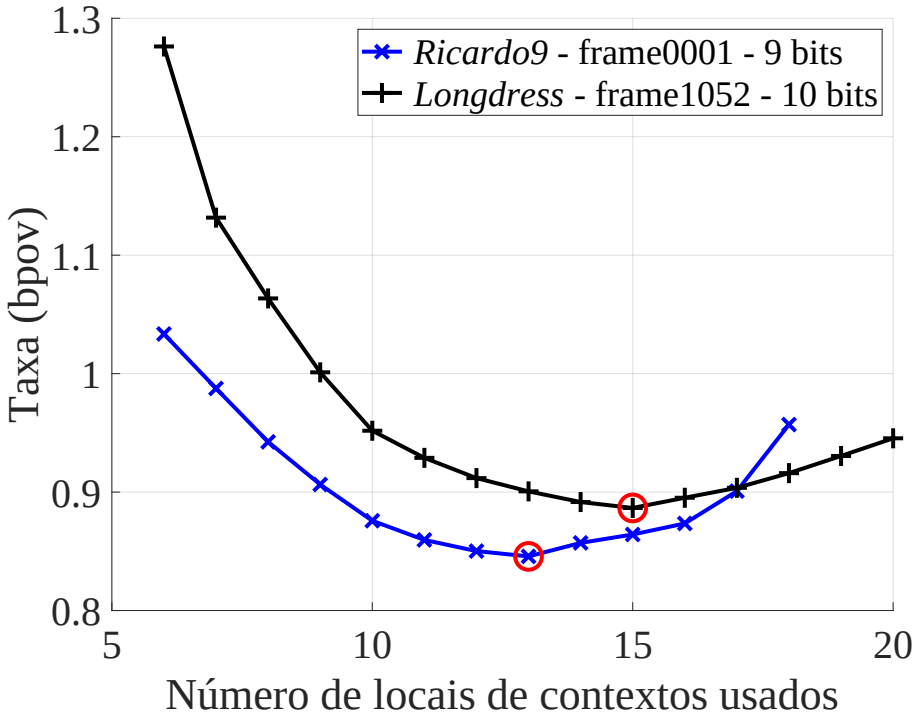


Figura 5.5: Exemplo de codificação variando número de contextos selecionados.

5.2.1 Resultados S4D-CS

Os experimentos feitos no algoritmo utilizam o conjunto de nuvens de pontos distribuídos pelo JPEG Pleno, mais especificamente o *Microsoft Upper Bodies* [33] e *8i Voxelized Full Bodies* [61], sendo a codificação feita nas primeiras cem nuvens de pontos de cada conjunto de dados. Os valores são computados em uma média, que representa a taxa para cada sequência. As nuvens de pontos do *dataset Microsoft Upper Bodies* são mostradas na Figura 5.6, enquanto as silhuetas do *8i Voxelized Full Bodies* estão na Figura 5.7. Ambos os *datasets* possuem somente nuvens de pontos voxelizadas.

A Tabela 5.3 mostra os resultados comparando o S4D [11] com o *S4D-CSFull* e o *S4D-CSPrune* descritos anteriormente. Analisando os valores de taxa, podemos confirmar que a aplicação com seleção de contextos tem um ganho bastante significativo em todas as sequências, incluindo aquelas nas quais não eram obtidos ganhos com uso de contextos *inter-frame* no código do S4D original, no caso as sequências *Longdress* e *RedandBlack*. Também percebe-se que o algoritmo de seleção rápida *S4D-CSPrune* possui resultados bastante semelhantes aos da versão completa do algoritmo *S4D-CSFull*, logo o *S4D-CSPrune* será a adotado como a aplicação proposta final.

Outro ganho verificado foi em relação ao impacto da escolha de eixo para a codificação. Para o S4D a escolha de eixo possui um grande impacto nos valores de taxa obtidos, em média

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

Figura 5.6: Nuvens de pontos 9 bits: (a) *Andrew9* (b) *David9* (c) *Phil9* (d) *Ricardo9* (e) *Sarah9*.

(a) (b)

(c) (d)

Figura 5.7: Nuvens de pontos 10 bits: (a) *Longdress* (b) *Loot* (c) *RedandBlack* (d) *Soldier*.

10%, portanto, era necessário que todos os três eixos fossem testados para melhor eficiência na compressão. Para o algoritmo de seleção de contextos, no entanto, os valores de taxa para codificação nos três eixos são bastante semelhantes, reduzindo bastante a disparidade média, logo o efeito da escolha de eixos é reduzido aqui, deixando a possibilidade de uma versão mais simples do código com escolha prévia de eixos. Para efeitos de comparação, entretanto, a seleção de eixos ainda é realizada aqui.

A Tabela 5.4 mostra os resultados do algoritmo proposto (*S4D-CSPrune*) em comparação com outros codificadores *intra-frame* sem perdas: (i) Algoritmo Octree original; (ii) MPEG G-PCC TMC13 v2.0 ; (iii) *Silhouette 3D* S3D; e os codificadores *inter-frame* (iv) P(Full) *Parent Node Inheritance plus Super-Resolution* e o (v) *Silhouette 4D* S4D. Essa tabela faz os cálculos de ganho do algoritmo proposto usando:

$$G_i = \frac{T_{proposto} - T_i}{T_i}, \tag{5.1}$$

com $T_{proposto}$ sendo o valor de taxa do algoritmo proposto para cada nuvem de pontos, T_i o valor para o método sendo usado como âncora e G_i o ganho percentual.

Pela Tabela 5.4 vemos que a aplicação do S4D já possuía ganhos significativos em cima do estado da arte G-PCC, em especial nas sequências do dataset *Full Bodies*, e a aplicação com seleção de contextos expande ainda mais esses ganhos, que vão entre 7.2% na sequência *RedandBlack* até 19.4% para a sequência *Soldier*.

Para fins de comparação, mostramos os resultados do S4D-CS, do MPEG-G-PCC funcionando no modo sem perdas, e também do MPEG-G-PCC funcionando no modo com perdas, em alta qualidade (PSNR 70.4 dB e 90% dos pontos originais) na Tabela 5.5. Nessa comparação, apenas o S4D opera no modo *inter-frame*, usando uma nuvem de pontos de referência. Entretanto, mostramos aqui que o S4D, operando no modo sem perdas, possui um desempenho competitivo com o G-PCC, operando no modo com perdas.

As Figuras de 5.8 até 5.16 mostram os valores de taxa para as 100 nuvens de pontos analisadas para cada sequência usando os métodos de compressão definidos anteriormente. Um comportamento visto em quase todas as sequências, é o padrão no desenho das curvas de taxas para cada nuvem de pontos. Esse comportamento mostra que os métodos de compressão funcionam de maneira semelhante para os mesmos quadros, ou seja, os quadros de nuvens de pontos que possuem piores desempenhos para um dos métodos, também obtêm piores desempenhos para os outros. A exceção à esse comportamento pode ser vista sequência *Soldier* na Figura 5.16, pois os codificadores *intra-frame* (S3D e TMC13) obtiveram valores de taxa praticamente constantes para todas as nuvens de pontos, enquanto os três codificadores *inter-frame* (S4D, S4D-CS e P(Full)), possuíam comportamentos semelhantes.

Os gráficos também mostram que a semelhança vista nos valores de média de taxa entre a aplicação S3D e S4D, também são aplicadas quando se analisa os valores para cada quadro das sequências individualmente. A representação dos valores de taxa nas figuras também evidenciam o ganho da aplicação S4D-CS, que é a aplicação que possui o melhor valor de taxa para todos os quadros de nuvens de pontos de todas as sequências analisadas.

Verificamos então a complexidade da aplicação S4D-CS comparada com o S4D através de codificações de uma série de nuvens de pontos usando os dois algoritmos. Essa comparação é válida pois ambas as aplicações foram desenvolvidas usando o MATLAB versão 2018a. Foi concluído que o S4D-CS possui 18% a mais de complexidade que o S4D padrão, pelo fato de existir um código de seleção embutido nele. O decodificador no entanto foi 12% mais rápido, esse ganho é obtido devido a retirada do algoritmo Modo multi presente no S4D.

5.3 CONCLUSÃO DO CAPÍTULO

Neste capítulo apresentamos dois algoritmos para compressão de nuvens de pontos dinâmicas, o S4D e o S4D-CS. O segundo, que utiliza o algoritmo desenvolvido de seleção de contextos,

apresentou o melhor desempenho para todas as nuvens de pontos analisadas, indicando que esse é, hoje, o estado da arte para compressão sem perdas de geometria de nuvens de pontos. O próximo capítulo conclui a dissertação.

Tabela 5.3: Comparação entre as variações do *Silhouette 4D*. Todas as taxas estão em bits por *voxel* ocupado (bpov).

		S4D				S4D-CSFull				S4D-CSPrune			
Banco de Dados	Sequência	**X**	**Y**	**Z**	**Best**	**X**	**Y**	**Z**	**Best**	**X**	**Y**	**Z**	**Best**
Microsoft Voxelized Upper Bodies	*Andrew9*	1.09	1.10	1.08	1.08	0.95	0.97	0.96	0.95	0.95	0.96	0.96	0.95
	David9	1.09	1.06	1.05	1.05	0.95	0.95	0.94	0.94	0.95	0.95	0.94	0.94
	Phil9	1.18	1.14	1.13	1.13	1.03	1.03	1.02	1.02	1.03	1.03	1.02	1.02
	Ricardo9	1.05	1.03	1.03	1.02	0.92	0.91	0.91	0.90	0.92	0.91	0.91	0.90
	Sarah9	1.08	1.05	1.05	1.04	0.92	0.93	0.92	0.92	0.93	0.93	0.92	0.92
	Média	1.09	1.07	1.07	1.06	0.96	0.96	0.95	0.95	0.96	0.96	0.95	0.95
8i Voxelized Full Bodies	*Longdress*	0.97	0.97	1.00	0.95	0.88	0.90	0.92	0.88	0.88	0.91	0.92	0.88
	Loot	0.95	0.92	0.95	0.91	0.84	0.85	0.87	0.84	0.83	0.85	0.87	0.83
	RedandBlack	1.02	1.04	1.06	1.02	0.94	0.96	0.98	0.94	0.95	0.97	0.98	0.95
	Soldier	0.89	0.81	0.89	0.81	0.66	0.67	0.65	0.65	0.66	0.67	0.65	0.65
	Média	0.96	0.93	0.98	0.92	0.83	0.85	0.85	0.83	0.83	0.85	0.86	0.83

Tabela 5.4: Comparação com algoritmos de Estado da Arte. Todas as taxas estão em bits por *voxel* ocupado (bpov).

		Codificadores Intra			Codificadores Inter			Ganhos do **S4D + Context Selection** sobre			
Banco de Dados	Sequência	**Octree**	**TMC13**	**S3D**	**P(Full)**	**S4D**	**Proposto**	**TMC13**	**S3D**	**P(Full)**	**S4D**
Microsoft Voxelized Upper Bodies [33]	*Andrew9*	2.58	1.14	1.12	1.37	1.08	0.95	−16.2%	−15.2%	−30.3%	−11.7%
	David9	2.62	1.07	1.06	1.31	1.05	0.94	−11.9%	−11.1%	−28.3%	−10.1%
	Phil9	2.65	1.17	1.14	1.42	1.13	1.02	−12.6%	−10.4%	−28.1%	−9.4%
	Ricardo9	2.60	1.09	1.04	1.34	1.02	0.90	−16.9%	−13.3%	−32.6%	−11.6%
	Sarah9	2.61	1.07	1.07	1.37	1.04	0.92	−14.2%	−14.3%	−32.7%	−11.9%
	Média	2.61	1.11	1.09	1.36	1.06	0.95	−14.4%	−12.9%	−30.4%	−10.9%
8i Voxelized Full Bodies [61]	*Longdress*	2.99	1.02	0.95	1.13	0.95	0.88	−13.4%	−7.4%	−21.7%	−7.3%
	Loot	2.98	0.96	0.92	1.02	0.91	0.84	−12.6%	−8.7%	−18.4%	−8.2%
	RedandBlack	3.00	1.08	1.02	1.23	1.02	0.94	−12.9%	−7.2%	−22.9%	−7.2%
	Soldier	3.00	1.03	0.96	0.85	0.81	0.65	−37.2%	−32.2%	−23.9%	−19.4%
	Média	2.99	1.02	0.96	1.06	0.92	0.83	−19.0%	−13.9%	−21.7%	−10.5%

Tabela 5.5: Comparação para quadros específicos. Todas as taxas estão em bits por *voxel* ocupado (bpov).

	TMC13		
Sequência	Sem perdas	Com perdas (70.4 dB)	Proposto
Longdress 1300	1.02	0.92	0.89
Loot 1200	0.98	0.88	0.86
RedandBlack 1550	1.11	0.99	0.98
Soldier 0690	1.04	0.94	0.89

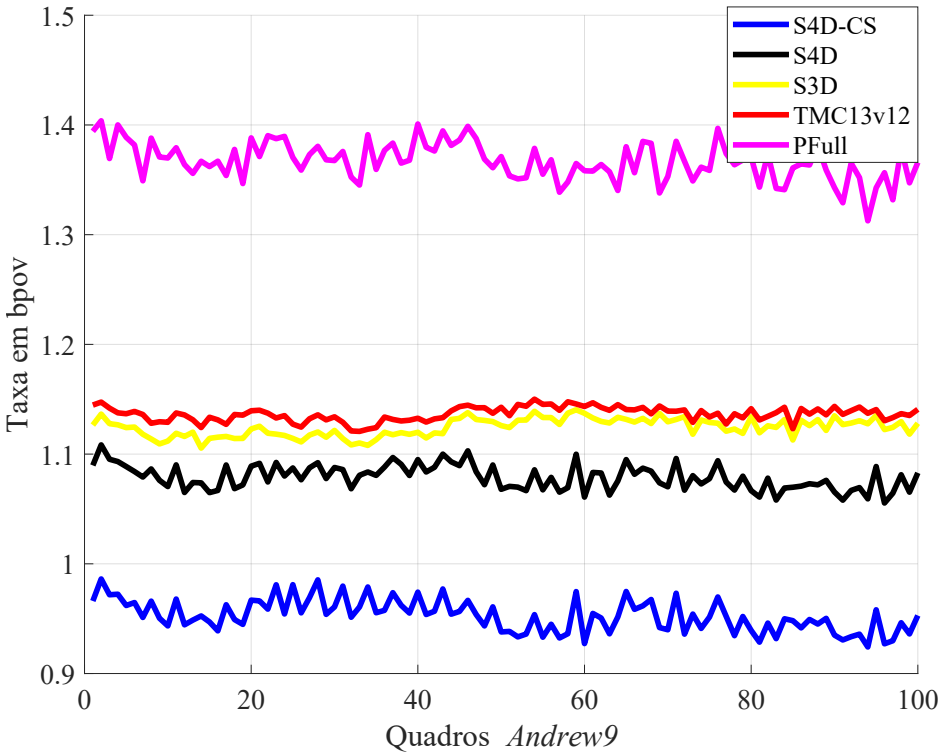


Figura 5.8: Valores de taxa para a sequência *Andrew9*.

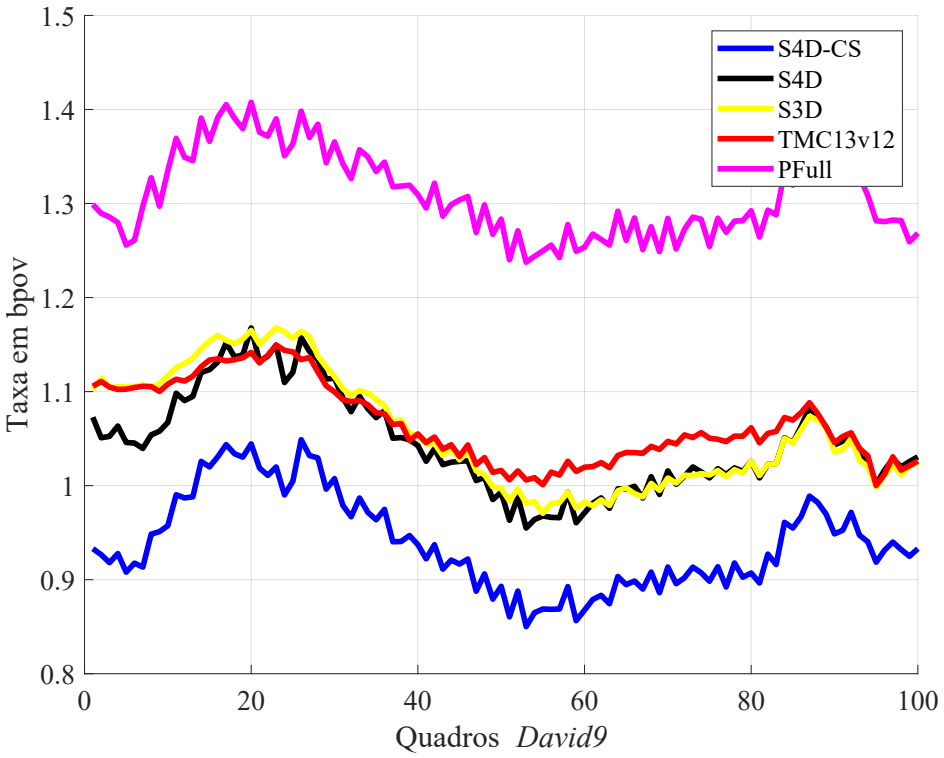


Figura 5.9: Valores de taxa para a sequência *David9*.

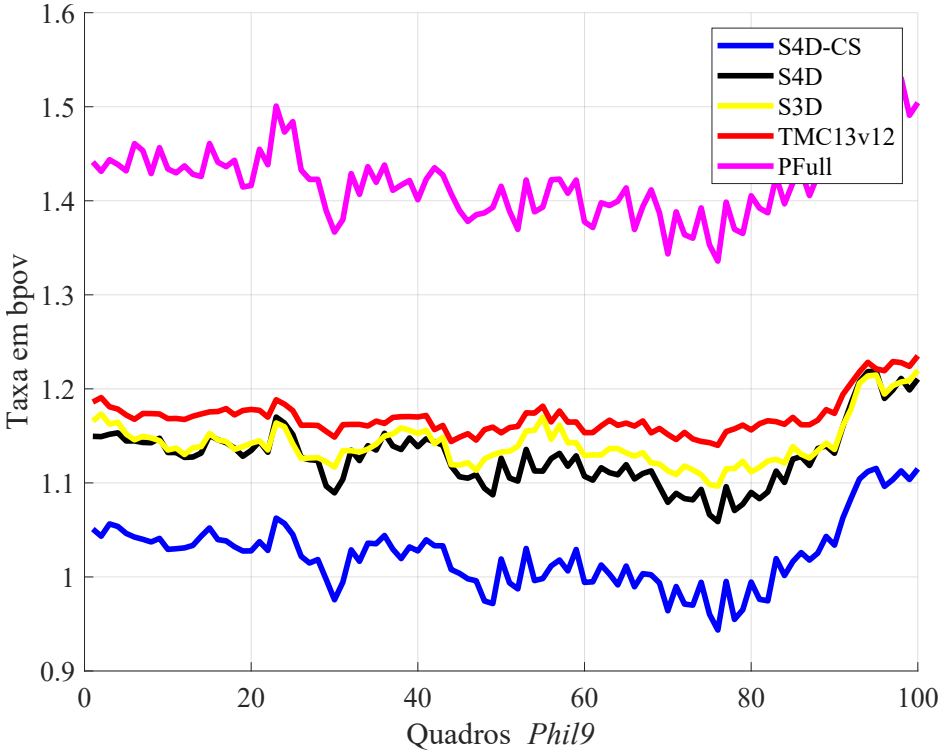


Figura 5.10: Valores de taxa para a sequência *Phil9*.

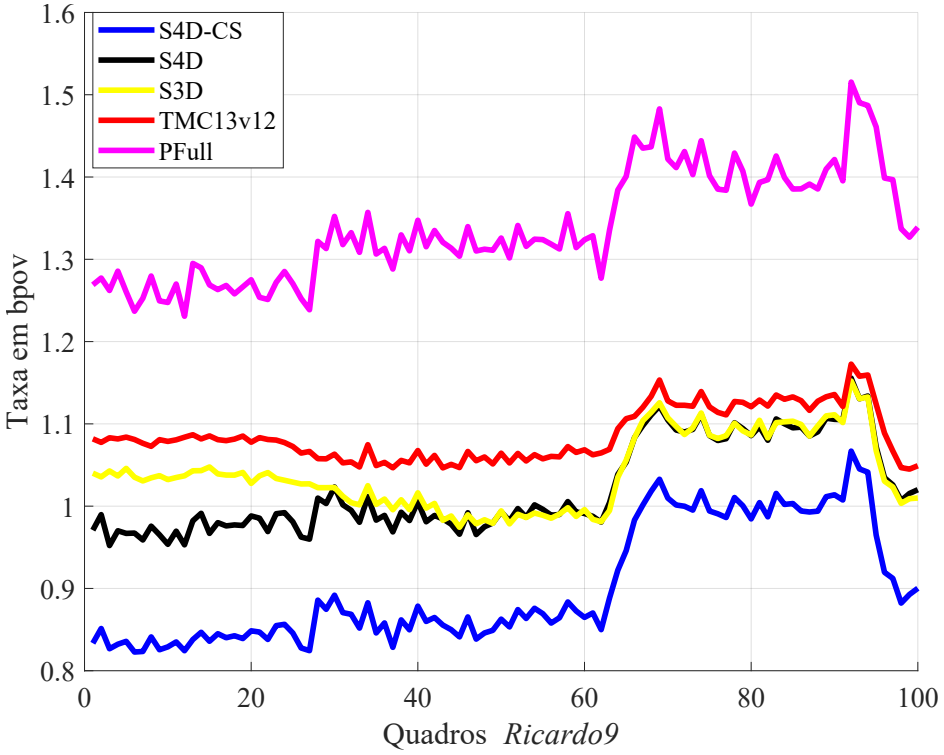


Figura 5.11: Valores de taxa para a sequência *Ricardo9*.

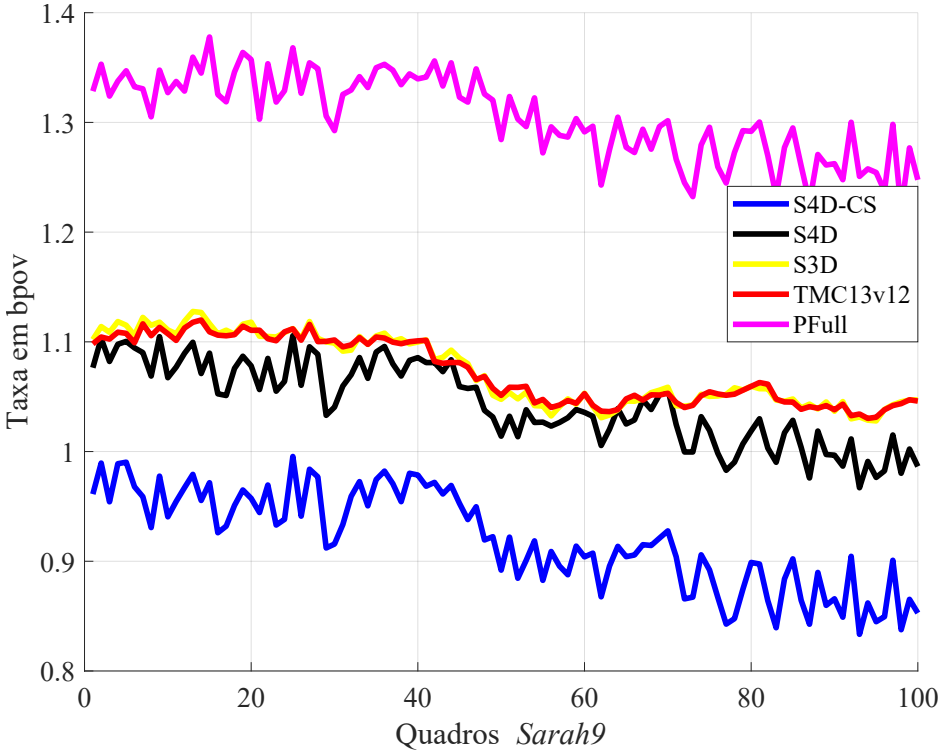


Figura 5.12: Valores de taxa para a sequência *Sarah9*.

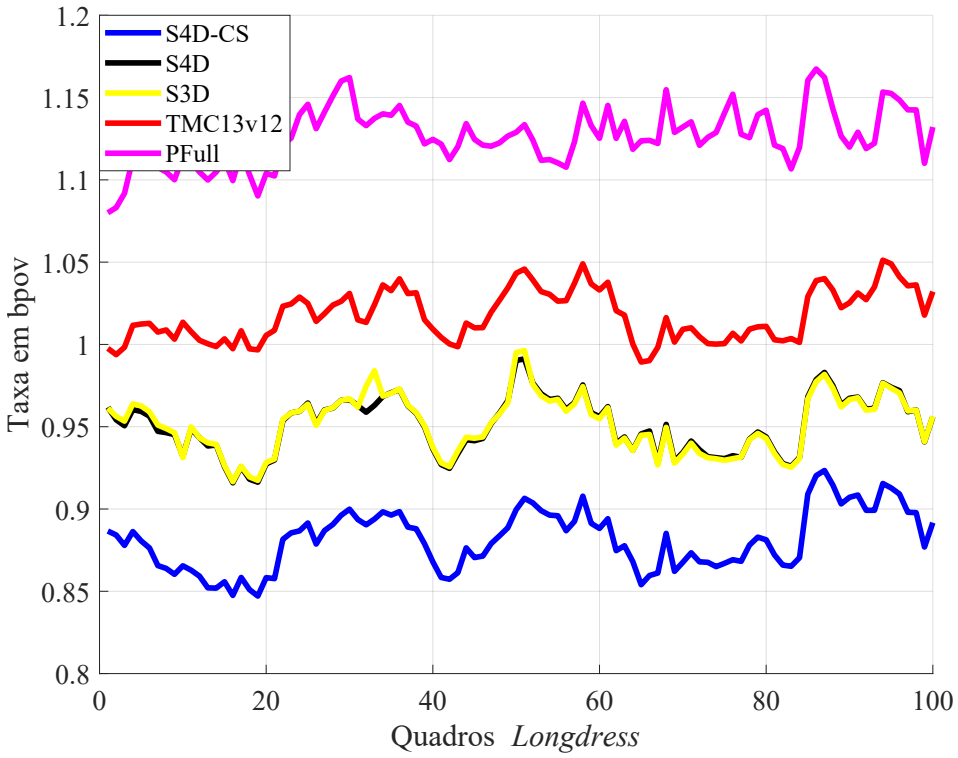


Figura 5.13: Valores de taxa para a sequência *Longdress*.

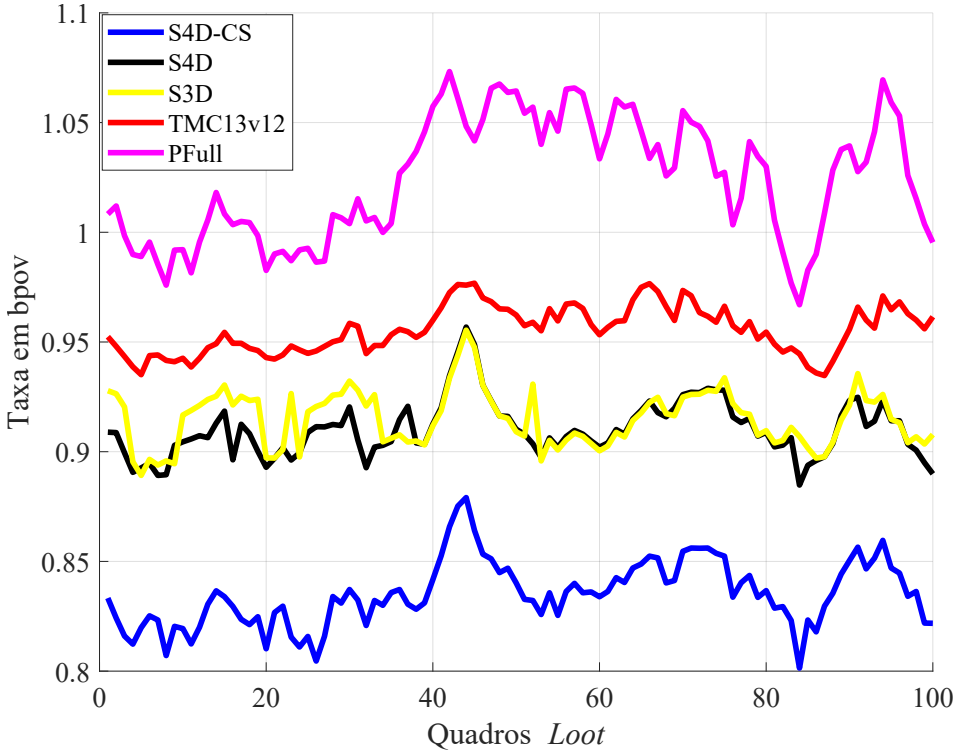


Figura 5.14: Valores de taxa para a sequência *Loot*.

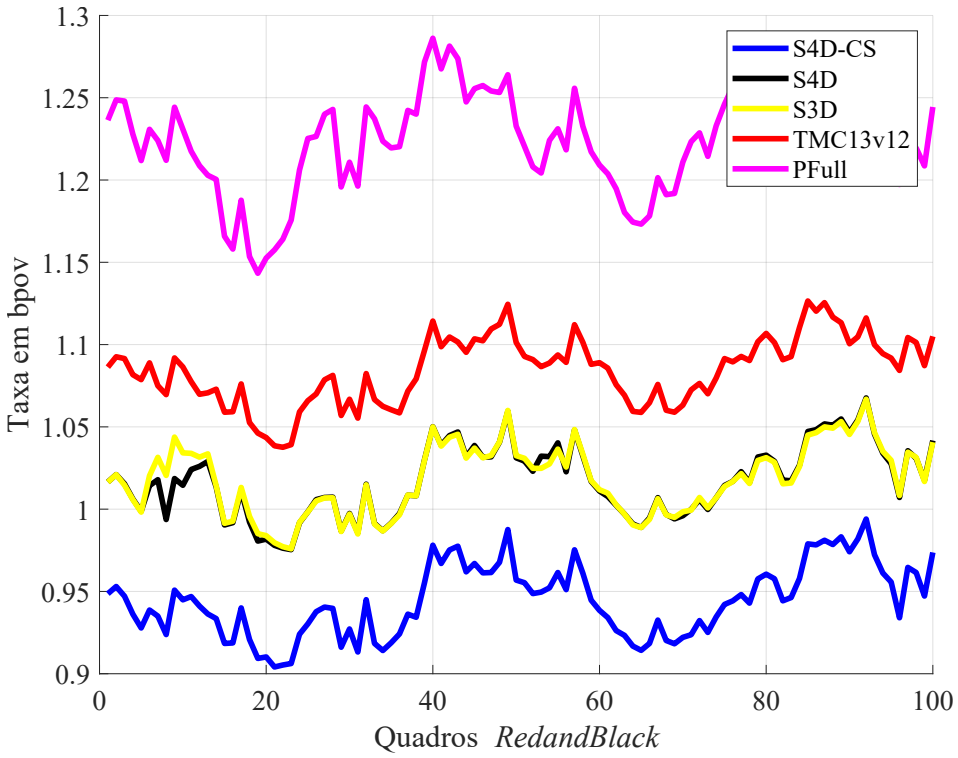


Figura 5.15: Valores de taxa para a sequência *RedandBlack*.

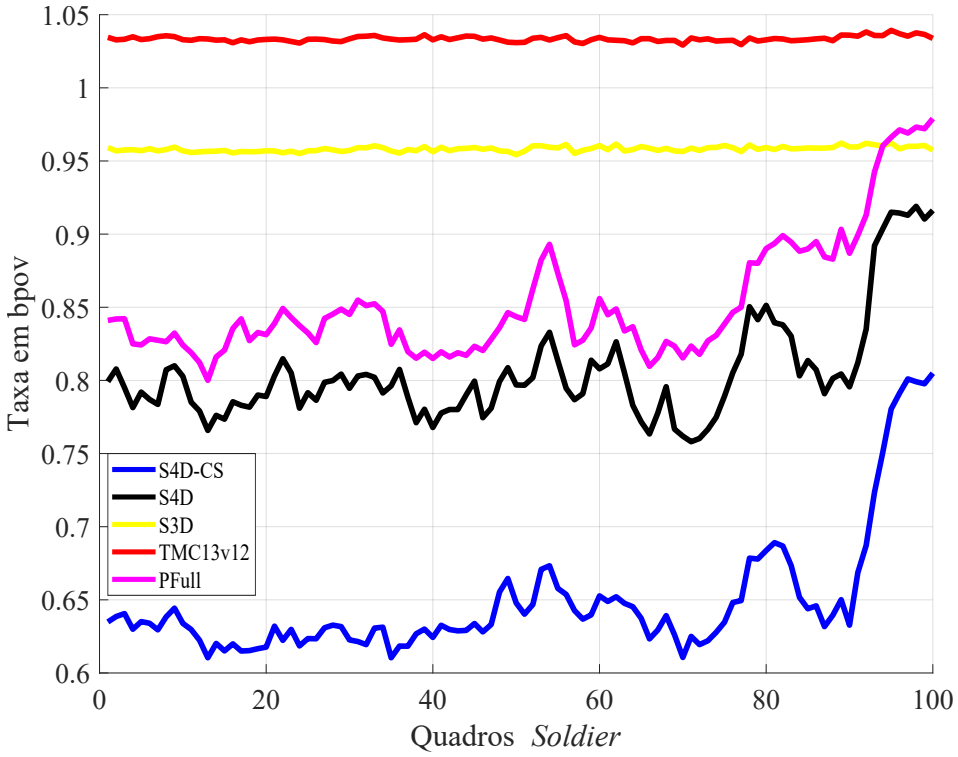


Figura 5.16: Valores de taxa para a sequência *Soldier*.

6 CONCLUSÃO

Um dos maiores desafios para a codificação de nuvens de pontos dinâmicas é a dificuldade para o mapeamento dos pontos da nuvem de pontos atual nas nuvens de pontos previamente codificadas. Antes desse trabalho, os melhores resultados de compressão de nuvens de pontos eram atingidos com uso de codificadores *intra-frame*, sendo que os codificadores *inter-frame*, como o P(Full), só possuíam desempenho superior em sequências de nuvens de pontos específicas.

Pensando nisso, três soluções foram desenvolvidas neste trabalho. A primeira é um algoritmo de pré-processamento de imagens, que seleciona uma combinação ideal de locais de contextos para cada imagem. Essa combinação é passada para uso em um codificador aritmético. O aplicativo proposto foi desenvolvido tendo em mente a baixa complexidade, evitando qualquer tipo de procedimento matemático complexo na seleção de locais de contexto, mantendo o tempo de execução do algoritmo praticamente o mesmo do codificador aritmético padrão (apenas 2 segundos a mais na média). Essa aplicação obteve valores de taxa superiores ao padrão JBIG para a maiorias das imagens que foram testadas, sendo uma alterativa viável para compressão de imagens binárias sem perdas, devido à simplicidade e eficiência da aplicação.

O segundo algoritmo é um codificador de geometria *inter-frame* sem perdas, chamado aqui de S4D, construído com base na aplicação S3D desenvolvida previamente, com a adição de um de local de contexto 4D. O S4D possui dois modos, um que utiliza apenas contextos 3D (ignorando a nuvem de pontos de referência), e outro que usa os contextos 3D e 4D. Apresentamos também um algoritmo rápido para seleção entre esses dois modos. Para algumas nuvens de pontos, os resultados obtidos foram bastante semelhantes ao S3D, que opera no modo *intra-frame*, e para outras a adição dos contextos 4D gerou ganhos significativos de compressão comparados com a aplicação anterior, esse foi o primeiro codificador *inter-frame* sem perdas que possui valores de taxa melhores que o padrão G-PCC.

No terceiro algoritmo, adaptamos um algoritmo de pré-processamento para seleção de contextos no S4D previamente desenvolvido. Esse algoritmo é capaz de definir um conjunto ideal de locais de contextos dentro de um total de 24, que irá garantir um bom desempenho de compressão. A nova aplicação tem ganhos bastante significativos se comparados como S4D, em média 10% para as sequências dos dois conjuntos de dados públicos do grupo MPEG, sendo o intervalo dos ganhos entre 7% e 21%. Até o presente momento, os resultados obtidos podem ser considerados estado da arte para a compressão sem perdas de geometria de nuvens de pontos.

Trabalhos futuros podem ser desenvolvidos em duas frentes principais. Para melhorar o desempenho de compressão, podem ser pesquisadas diferentes adaptações dos Modos de Decomposição diádica e Divisão em silhuetas unitárias. Para algumas nuvens de pontos, utilizar a Divisão em silhuetas unitárias mais cedo, agrupando um número maior de silhuetas, pode ser vantajoso. Além disso, o algoritmo hoje tem apenas duas opções: Decomposição diádica (gera apenas as

duas silhuetas filhas no nível imediatamente posterior) ou a Divisão em silhuetas unitárias (gera silhuetas unitárias diretamente). É possível adaptar outros modos, que não sejam um desses dois extremos, e é esperado que isso gere um ganho de compressão para algumas nuvens de pontos. Um modo como o *Direct Coding Mode* para codificar pontos extremos também pode apresentar ganhos neste codificador. Outro ponto para trabalhos futuros é adaptar o algoritmo para compressão durante a aquisição. Da forma como apresentado, tanto o S3D e o S4D necessitam de acesso a todos os pontos antes de se aplicar o algoritmo. Porém, é possível adaptar o algoritmo para que ele codifique pontos a medida que estes sejam adquiridos, principalmente se a aquisição tiver uma direção em um eixo. Para aumentar as possibilidades de aplicação do algoritmo, esta é uma direção de pesquisa futura bastante interessante. Outra ideia para trabalhos futuros é a adaptação do algoritmo para compressão de atributos da nuvem de pontos. Finalmente, é interessante também estudar a complexidade do algoritmo para que se possa preparar uma proposta para o MPEG do S3D ou S4D. Os resultados em termos de eficiência de compressão são bastante competitivos.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] D. Habermann, C. E. Vido, F. S. Osório, and F. Ramos, "Road junction detection from 3d point clouds," in *2016 International Joint Conference on Neural Networks (IJCNN)*, pp. 4934–4940, 2016.

[2] L. De Paula Veronese, J. Guivant, F. A. Auat Cheein, T. Oliveira-Santos, F. Mutz, E. de Aguiar, C. Badue, and A. F. De Souza, "A light-weight yet accurate localization system for autonomous cars in large-scale and complex environments," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 520–525, 2016.

[3] T. Podobnikar and A. Vrečko, "Digital elevation model from the best results of different filtering of a lidar point cloud," *Transactions in GIS*, vol. 16, no. 5, pp. 603–617, 2012.

[4] Q. Cheng, P. Sun, C. Yang, Y. Yang, and P. X. Liu, "A morphing-based 3d point cloud reconstruction framework for medical image processing," *Computer Methods and Programs in Biomedicine*, vol. 193, p. 105495, 2020.

[5] M. Sinko, P. Kamencay, R. Hudec, and M. Benco, "3d registration of the point cloud data using icp algorithm in medical image analysis," in *2018 ELEKTRO*, pp. 1–6, 2018.

[6] 3DG, "G-PCC codec description v4," tech. rep., ISO/IEC JTC 1/SC 29/WG 11 input document w18673, 2019.

[7] 3DG, "[V-PCC] V-PCC Test Model v7," tech. rep., ISO/IEC JTC 1/SC 29/WG 11 input document w18666, 2019.

[8] R. Rosário and E. Peixoto, "Intra-frame compression of point cloud geometry using boolean decomposition," in *2019 IEEE Visual Communications and Image Processing (VCIP)*, pp. 1–4, 2019.

[9] E. Peixoto, "Intra-frame compression of point cloud geometry using dyadic decomposition," *IEEE Signal Processing Letters*, pp. 1–1, 2020.

[10] E. Ramalho and E. Peixoto, "Context selection for lossless compression of bi-level images," *XXXVIII Simpósio Brasileiro de Telecomunicações e Processamento de Sinais*, 2020.

[11] E. Peixoto, E. Medeiros, and E. Ramalho, "Silhouette 4d: An inter-frame lossless geometry coder of dynamic voxelized point clouds," in *2020 IEEE International Conference on Image Processing (ICIP)*, pp. 2691–2695, 2020.

[12] E. Ramalho, E. Peixoto, and E. Medeiros, "Silhouette 4d with context selection: Lossless geometry compression of dynamic point clouds," *IEEE Signal Processing Letters*, vol. 28, pp. 1660–1664, 2021.

[13] ISO/IEC JTC 1 , "Coding of Still Pictures," Tech. Rep. N 1359, ISO/IEC, July 1999.

[14] M. D. G. C. WG 7, "G-PCC codec description v9," tech. rep., ISO/IEC JTC 1/SC 29/WG 7 N 0011, 2020.

[15] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[16] N. Abramson, *Information Theory and Coding*. McGraw-Hill, 1963.

[17] F. Jenilek, *Probabilistic Information Theory*. McGraw-Hill, 1968.

[18] R. Pasco, "Source code algorithms for fast data compression," 1976.

[19] J. J. Rissanen, "Generalized kraft inequality and arithmetic coding," *IBM Journal of Research and Development*, vol. 20, no. 3, pp. 198–203, 1976.

[20] J. Rissanen and G. G. Langdon, "Arithmetic coding," *IBM Journal of Research and Development*, vol. 23, no. 2, pp. 149–162, 1979.

[21] V. Kyrki, "Jbig image compression standard," 04 1999.

[22] P. G. Howard, F. Kossentini, B. Martins, S. Forchhammer, and W. J. Rucklidge, "The emerging jbig2 standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 7, pp. 838–848, 1998.

[23] F. Ono, W. Rucklidge, R. Arps, and C. Constantinescu, "Jbig2-the ultimate bi-level image coding standard," in *Proceedings 2000 International Conference on Image Processing (Cat. No.00CH37101)*, vol. 1, pp. 140–143 vol.1, 2000.

[24] A. J. Pinho, "A jbig-based approach to the encoding of contour maps," *IEEE Transactions on Image Processing*, vol. 9, no. 5, pp. 936–941, 2000.

[25] M. Abdat and M. G. Bellanger, "Combining gray coding and jbig for lossless image compression," in *Proceedings of 1st International Conference on Image Processing*, vol. 3, pp. 851–855 vol.3, 1994.

[26] K. Khursheed, N. Ahmad, M. Imran, and M. O'Nils, "Detecting and coding region of interests in bi-level images for data reduction in wireless visual sensor network," in *2012 IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pp. 705–712, 2012.

[27] B. Fowler, R. Arps, A. El Gamal, and D. Yang, "Quadtree based jbig compression," in *Proceedings DCC '95 Data Compression Conference*, pp. 102–111, 1995.

[28] B. Martins and S. Forchhammer, "Tree coding of bilevel images," *IEEE Transactions on Image Processing*, vol. 7, no. 4, pp. 517–528, 1998.

[29] P. Franti and E. Ageenko, "On the use of context tree for binary image compression," in *Proceedings 1999 International Conference on Image Processing (Cat. 99CH36348)*, vol. 3, pp. 752–756 vol.3, 1999.

[30] H. S. Malvar, "Fast adaptive encoder for bi-level images," in *Proceedings DCC 2001. Data Compression Conference*, pp. 253–262, Mar. 2001.

[31] S. Zahir and M. Naqvi, "A near minimum sparse pattern coding based scheme for binary image compression," in *IEEE International Conference on Image Processing 2005*, vol. 2, pp. II–289, 2005.

[32] L. Zhou and S. Zahir, "A new efficient algorithm for lossless binary image compression," in *2006 Canadian Conference on Electrical and Computer Engineering*, pp. 1427–1431, 2006.

[33] C. Loop, Q. Cai, S. O. Escolano, and P. A. Chou, "Microsoft Voxelized Upper Bodies – A Voxelized Point Cloud Dataset," tech. rep., ISO/IEC JTC1/SC29/WG11 m38673 ISO/IEC JTC1/SC29/WG1 M72012, Geneva, Switzerland, 2016.

[34] R. L. de Queiroz and P. A. Chou, "Compression of 3D point clouds using a region-adaptive hierarchical transform," *IEEE Transactions on Image Processing*, vol. 25, pp. 3947–3956, Aug. 2016.

[35] D. Meagher, "Geometric modeling using octree-encoding," *Computer Graphics and Image Processing*, vol. 19, pp. 129–147, 06 1982.

[36] D. C. Garcia, T. A. Fonseca, R. U. Ferreira, and R. L. de Queiroz, "Geometry coding for dynamic voxelized point clouds using octrees and multiple contexts," *IEEE Transactions on Image Processing*, vol. 29, pp. 313–322, 2020.

[37] D. C. Garcia and R. L. de Queiroz, "Intra-frame context-based octree coding for point-cloud geometry," in *2018 25th IEEE International Conference on Image Processing (ICIP)*, pp. 1807–1811, Oct. 2018.

[38] L. Cui, R. Mekuria, M. Preda, and E. S. Jang, "Point-cloud compression: Moving picture experts group's new standard in 2020," *IEEE Consumer Electronics Magazine*, vol. 8, no. 4, pp. 17–21, 2019.

[39] L. Tang, F. peng Da, and Y. Huang, "Compression algorithm of scattered point cloud based on octree coding," in *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, pp. 85–89, 2016.

[40] A. Dricot, F. Pereira, and J. Ascenso, "Rate-distortion driven adaptive partitioning for octree-based point cloud geometry coding," in *2018 25th IEEE International Conference on Image Processing (ICIP)*, pp. 2969–2973, Oct. 2018.

[41] P. de Oliveira Rente, C. Brites, J. Ascenso, and F. Pereira, "Graph-based static 3d point clouds geometry coding," *IEEE Transactions on Multimedia*, vol. 21, pp. 284–299, Feb 2019.

[42] M. Quach, G. Valenzise, and F. Dufaux, "Learning convolutional transforms for lossy point cloud geometry compression," in *2019 IEEE International Conference on Image Processing (ICIP 2019)*, pp. 4320–4324, Sep. 2019.

[43] A. Filali, V. Ricordel, N. Normand, and W. Hamidouche, "Rate-distortion optimized tree-structured point-lattice vector quantization for compression of 3d point clouds geometry," in *2019 IEEE International Conference on Image Processing (ICIP 2019)*, pp. 1099–1103, Sept. 2019.

[44] V. Ricordel and C. Labit, "Tree-structured lattice vector quantization," in *1996 8th European Signal Processing Conference (EUSIPCO 1996)*, pp. 1–4, 1996.

[45] V. Ricordel and C. Labit, "Vector quantization by packing of embedded truncated lattices," in *Proceedings., International Conference on Image Processing*, vol. 3, pp. 292–295 vol.3, 1995.

[46] I. Daribo, R. Furukawa, R. Sagawa, and H. Kawasaki, "Adaptive arithmetic coding for point cloud compression," in *2012 3DTV-Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON)*, pp. 1–4, Oct. 2012.

[47] W. Zhu, Y. Xu, L. Li, and Z. Li, "Lossless point cloud geometry compression via binary tree partition and intra prediction," in *2017 IEEE 19th International Workshop on Multimedia Signal Processing (MMSP)*, pp. 1–6, Oct. 2017.

[48] S. Milani, "Fast point cloud compression via reversible cellular automata block transform," in *2017 IEEE International Conference on Image Processing (ICIP)*, pp. 4013–4017, Sept. 2017.

[49] S. Limuti, E. Polo, and S. Milani, "A transform coding strategy for voxelized dynamic point clouds," in *2018 25th IEEE International Conference on Image Processing (ICIP)*, pp. 2954–2958, Oct. 2018.

[50] J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz, and E. Steinbach, "Real-time compression of point cloud streams," in *2012 IEEE International Conference on Robotics and Automation*, pp. 778–785, May 2012.

[51] D. C. Garcia and R. L. de Queiroz, "Context-based octree coding for point-cloud video," in *2017 IEEE International Conference on Image Processing (ICIP)*, pp. 1412–1416, Sept. 2017.

[52] M. Krivokuća, P. A. Chou, and M. Koroteev, "A volumetric approach to point cloud compression–part ii: Geometry compression," *IEEE Transactions on Image Processing*, vol. 29, pp. 2217–2229, 2020.

[53] P. A. Chou, M. Koroteev, and M. Krivokuća, "A volumetric approach to point cloud compression—part i: Attribute compression," *IEEE Transactions on Image Processing*, vol. 29, pp. 2203–2216, 2020.

[54] S. Milani, E. Polo, and S. Limuti, "A transform coding strategy for dynamic point clouds," *IEEE Transactions on Image Processing*, vol. 29, pp. 8213–8225, 2020.

[55] A. F. R. Guarda, N. M. M. Rodrigues, and F. Pereira, "Adaptive deep learning-based point cloud geometry coding," *IEEE Journal of Selected Topics in Signal Processing*, vol. 15, no. 2, pp. 415–430, 2021.

[56] C. Zhang, D. Florêncio, and C. Loop, "Point cloud attribute compression with graph transform," in *2014 IEEE International Conference on Image Processing (ICIP)*, pp. 2066–2070, 2014.

[57] P. A. Chou and R. L. de Queiroz, "Gaussian process transforms," in *2016 IEEE International Conference on Image Processing (ICIP)*, pp. 1524–1528, 2016.

[58] A. Haar, "Zur theorie der orthogonalen funktionensysteme. (erste mitteilung).," *Mathematische Annalen*, vol. 69, pp. 331–371, 1910.

[59] "Public-Domain Test Images.." [Online]. Disponível em: `https://homepages.cae.wisc.edu/~ece533/images/`.

[60] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[61] E. d'Eon, B. Harrison, T. Myers, and P. A. Chou, "8i Voxelized Full Bodies, version 2 – A Voxelized Point Cloud Dataset," tech. rep., ISO/IEC JTC1/SC29/WG11 m40059 ISO/IEC JTC1/SC29/WG1 M74006 Geneva, Switzerland, 2017.

[62] "JPEG Pleno database." [Online]. Disponível em: `https://jpeg.org/plenodb/`.