Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Codificação das cores de uma point cloud através da sua divisão em filamentos

Bruno José Bergamaschi Kumer Reis

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia de Computação

Orientador
Prof. Dr. Eduardo Peixoto Fernandes da Silva

Brasília
2018

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Codificação das cores de uma point cloud através da sua divisão em filamentos

Bruno José Bergamaschi Kumer Reis

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia de Computação

Prof. Dr. Eduardo Peixoto Fernandes da Silva (Orientador)
CIC/UnB

Prof. Dr. Camilo Chang Dorea
CIC/UnB

Prof. Dr. Renan Utida Ferreira
FGA/UnB

Prof. Dr. José Edil Guimarães de Medeiros
Coordenador do Curso de Engenharia de Computação

Brasília, 10 de dezembro de 2018

Dedicatória

Eu dedico este trabalho à minha mãe Márcia Kumer e ao meu pai José Durval, aos meus quatro irmãos, Linda, Danilo, Ana Clara e Rosa, a todos os meus tios, avós e primos.

Agradecimentos

Agradeço imensamente à toda minha família por sempre ter me dado todo amor e apoio necessários em minhas escolhas e nunca terem me deixado faltar nada, sendo a principal razão por eu ter chegado até aqui. Agradeço em especial aos meus pais, Márcia e Durval, por sempre terem me incentivado em meus estudos e terem me dado carinho e amor incondicionais, sendo os responsáveis pelo homem que sou hoje; às minhas duas avós, Linda e Waldemira, por terem me proporcionado uma infância incrível e moldado o meu caratér e os meus valores; à minha irmã, Linda, por sempre ser minha companheira e me fazer sorrir todos os dias; e às minhas tias, Marta e Luciane, que sempre torceram por mim, me amaram e cuidaram em todos os momentos da minha vida.

Agradeço ao professor Eduardo Peixoto, sem dúvidas o melhor professor que eu tive nessa instituição, por toda sua orientação, paciência e disposição dedicados a mim e a esse trabalho nesses dois últimos semestres.

Agradeço à todos os Professores da UnB, em especial aos que passaram pela minha vida acadêmica, me passando seus ensinamentos e incentivando o meu amor ao estudo, se eu cheguei até aqui foi graças a eles.

Agradeço à Universidade de Brasília pela minha formação profissional e por todo aprendizado.

Agradeço também aos meus colegas de curso e amigos que tornaram a passagem por esse desafio mais leve e que se disponibilizaram a me ajudar sempre quando senti necessidade.

Resumo

Point cloud, ou nuvem de pontos, é uma representação tridimensional de uma cena que é possível ser visualizada por qualquer ângulo desejado. É uma tecnologia cuja captura está sendo cada vez mais desenvolvida e para que possa ser difundida em diversas aplicações na sociedade é necessário o aprimoramento de seu processamento e codificação. Assim, nesse trabalho será desenvolvido um codificador sem perdas das cores de uma point cloud completo, que consiste em dividir a point cloud em camadas, segmentar as camadas em filamentos unidimensionais e codificar as cores desses filamentos através da codificação diferencial e de Huffman. O trabalho desenvolvido não depende da forma que a geometria da point cloud é codificada, permitindo que uma vez que o destinatário tenha a geometria seja possível decodificar gerando exatamente os mesmos filamentos realizados na codificação e a codificação gerada é sem perdas, podendo ser interessante para aplicações que não tolerem perda de informação. Os resultados obtidos foram promissores, a taxa de compressão atingida média foi de 10 para 1, considerando o tamanho da point cloud como um todo e não só suas cores. Comparando com o codificador com perdas RAHT, que hoje é considerado o estado-da-arte na compressão de cores de uma point cloud, com um valor de parâmetro pequeno suficiente para que a imagem codificada seja o mais próximo da imagem original, o algortimo desenvolvido consegue gerar um arquivo menor e sem perdas, fazendo com que o trabalho possa ser competitivo com os devidos aprimoramentos. As possíveis melhorias futuras no trabalho desenvolvido se extendem desde a otimização na segmentação dos cortes para gerar os filamentos até a forma como as cores são codificadas e escritas no arquivo final.

Palavras-chave: Point clouds, compressão, cores, filamentos

Abstract

Point clouds are a three-dimensional representation of a scene that can be viewed by any desired angle. It is a technology whose capture is being improved considerably and for it to be diffused in diverse applications in society it is necessary to improve its processing and codification. In this work we will develop a complete lossless color encoder for point clouds, which consists of dividing the point cloud into layers and then segment these layers into one-dimensional filaments, encoding the colors from these filaments with Huffman and differential encoding. One of the merits of the work developed is that it does not depend on the way that the geometry of the point cloud is coded, allowing the use of other methods in the coding of the geometry of the point cloud. It also is a lossless method, allowing the use of this work for applications that cannot afford lossy methods for the used data. The results obtained were promissing, the average data compression ratio was 10 to 1, considering the size of the pointcloud as a whole, not only its colors. Comparing with the lossy encoder RAHT, which is considered the state of the art in color compression for point clouds, with a small enough parameter value for RAHT, making the coded image to be closest to he original imagem, our algorithm is able to generate a lossless smaller file, being able to be competitive with the appropriate improvements. One of the possible future improvements with the work that was developed is to find a better way for the segmentation of the layers or even changing the methodology used for coding the colors from the filaments, amongst other things.

Keywords: Point clouds, data compression, colors, filaments

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Com o crescimento das indústrias cinematográfica, de jogos eletrônicos e robótica e, mais recentemente, a indústria de impressoras 3D, aliadas a criação de dispositivos de captação de *point clouds* de baixo custo, como o Kinect (dispositivo desenvolvido pela Microsoft) [1] [2] [3], aprimoramentos na representação de cenas tridimensionais vêm sendo estudadas [1] [2]. Com isso, contínuos esforços vêm sendo realizados no processamento avançado de *point clouds*, entre eles seu processamento e codificação [4] [1] [5] [2].

Nos últimos anos diferentes tecnologias foram utilizadas e aprimoradas para a tentativa de reproduzir vistas em 3D [6], dando destaque ao 3D estereoscópico. Nessa técnica é realizada a análise de duas ou mais imagens obtidas em pontos diferentes de uma cena para através delas gerar a noção de perspectiva, de forma similar como os dois olhos do corpo humano funcionam [6] [7].

As vantagens da utilização de point clouds na busca pela representação da tridimensionalidade ao invés de outros métodos já desenvolvidos se deve ao fato de que elas permitem que se possa gerar uma cena e visualizá-la de qualquer posição, diferentemente de outras técnicas que se limitam a predição de diferentes quadros registrados, a utilização de uma point cloud permite que uma vista possa ser visualizada com muito mais liberdade, reproduzindo efetivamente a perspectiva, não uma predição limitada pela quantidade de frames capturados de uma cena. [6] [8]

Atualmente a tecnologia de point cloud não é amplamente difundida na sociedade pois sua captura ainda está sendo aprimorada e o seu tamanho é grande o suficiente para inviabilizar a rápida transmissão e processamento desse tipo de dado [8]. Nesse contexto, a compressão de point clouds é uma tecnologia habilitadora e convergente, onde a partir do momento em que obtivermos bons resultados com sua codificação e processamento, poderemos causar mudanças tecnológicas significativas na humanidade, podendo criar tecnologias derivadas em todos os campos do conhecimento.

Assim, no trabalho desenvolvido nessa monografia, iremos explorar especificamente a

parte da compressão de cores da point cloud de uma forma que até então não havia sido abordada, dividindo a point cloud em filamentos e realizando uma série de processos em cima desses filamentos para tentar explorar técnicas não usadas hoje no estado da arte. A compressão que será desenvolvida nesse trabalho é sem perdas e não depende da forma como é realizada a compressão da geometria, tornando-a bem diferente do que é utilizado hoje.

Dessa forma, essa monografia é dividida da seguinte maneira: o Capítulo 2 apresentará os fundamentos teóricos necessários para a compreensão do método que será desenvolvido; o Capítulo 3 apresentará de forma breve alguns dos métodos de compressão de cores de point cloud considerados como o estado-da-arte, tendo como objetivo deixar claro a diferença do que será elaborado nesse trabalho e o que já existe atualmente; o Capítulo 4 discorrerá sobre a metodologia utilizada neste trabalho, detalhando as etapas realizadas para a compressão das cores de uma point cloud por meio da divisão dela em filamentos; o Capítulo 5 apresentará os resultados obtidos utilizando a metodologia proposta e comparações com métodos que hoje são considerados como estado-da-arte; e o Capítulo 6 apresenta a Conclusão e algumas propostas para trabalhos futuros que possam ser interessantes na metodologia desenvolvida.

Capítulo 2

Fundamentos teóricos

Este capítulo tem como objetivo explicar os principais conceitos utilizados no processo de compressão de cores de uma point cloud via divisão de filamentos e compressão diferencial. Entre os principais conceitos abordados temos o de imagem, nuvem de pontos (do termo em inglês *point cloud*) e codificação diferencial e codificação de Huffman.

2.1 Imagem

Uma imagem pode ser definida como uma função de duas dimensões *f(x,y)* onde x e y representam coordenadas espaciais em um plano e a amplitude f em qualquer par de coordenadas *(x,y)* é chamada de escala de cinza ou intensidade da imagem naquele ponto [9].

As imagens podem ter suas coordenadas e sua amplitude tanto contínuas quanto finitas e discretas. Para a resolução do problema proposto nesse artigo teremos interesse em trabalhar somente com imagens que possuem propriedades finitas e discretas, as quais chamamos de *imagens digitais*. Podemos transformar qualquer imagem com propriedades contínuas em uma imagem digital, processo que denominamos de digitalização de uma imagem. O processo de digitalização de uma imagem consiste em duas etapas principais, a quantização e a amostragem. Enquanto a quantização define um conjunto finito de níveis de cinza para a representação da amplitude em uma dada coordenada, a amostragem discretizará o domínio de definição da função f(x,y), fazendo com que se crie uma grade de pontos igualmente espaçados entre si. [9].

Na representação da Figura 2.1(b) temos uma boa representação visual de uma imagem digitalizada monocromática. Nela, cada quadrado presente dentro da grade representa uma coordenada (x,y) da imagem e a intensidade da cor de cada quadrado possui uma amplitude, que na imagem mostrada é dada em escala de cinza. Denominamos esse conjunto de dados (coordenadas e intensidade) um pixel (do termo em inglês *picture*

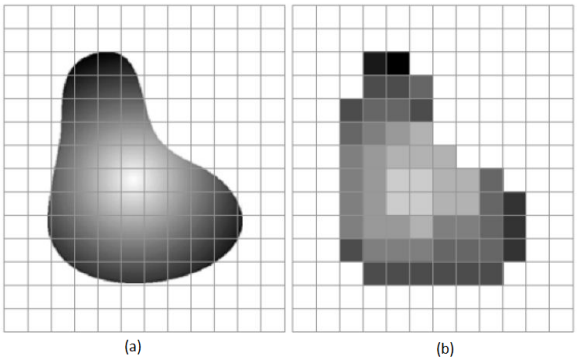

Figura 2.1: Demonstração visual da digitalização de uma figura (a) Imagem contínua projetada em uma grade de sensores (b) Resultado da amostragem e quantização da imagem (a).

element). Em imagens monocromáticas, como as da Figura 2.1, a intensidade I do *pixel* em uma coordenada (x_0, y_0) é dada por

$$I = (x_0, y_0) \tag{2.1}$$

A escala formada por todos os possíveis valores de I é denominada escala de cinza, que vai do valor 0 até $L-1$, onde L é definido pelo número n de bits utilizados para representar a intensidade de cada pixel, em que

$$L = 2^n, \tag{2.2}$$

onde a intensidade 0 representa a cor preta e a intensidade $L-1$ representa a cor branca, com qualquer valor entre esses dois representando uma cor na escala de cinza diferente. Para imagens coloridas, a intensidade é usualmente dada pelo sistema de cores aditivas RGB, em que a intensidade de cada cor R(vermelho), G(verde) e B(azul) são combinadas para reproduzir um largo espectro cromático. Cada pixel em uma imagem pode ter seus componentes RGB organizados na forma do vetor coluna

$$C = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}, \tag{2.3}$$

em que c_1 é a intensidade do pixel na imagem vermelha, c_2 intensidade do pixel na imagem verde e c_3 a intensidade do pixel na imagem azul. Dessa forma, quando falamos em uma imagem colorida RGB, além de termos sua dimensão $M \times N$, temos três imagens de dimensão igual, em que cada uma terá uma intensidade dos componentes RGB, que quando considerados como uma só imagem nos darão a imagem colorida original.

2.2 Nuvem de pontos

Uma nuvem de pontos é um conjunto de pontos que contêm dados em um determinado sistema de coordenadas. Uma nuvem de pontos é uma estrutura de dados usada para representar uma coleção de pontos multidimensionais [8]. Como o foco será o sistema de coordenadas tridimensional, para o propósito desse trabalho, uma nuvem de pontos é uma coleção de pontos localizados em um sistema de coordenadas XYZ em que a coordenada de um ponto específico dessa estrutura pode ser representada por um vetor coluna

$$PC = \begin{bmatrix} PC_1 \\ PC_2 \\ PC_3 \end{bmatrix} \tag{2.4}$$

onde PC_1 corresponde a variável x, PC_2 corresponde a variável y e PC_3 corresponde a variável z. Um ponto ainda pode possuir atributos adicionais como valores relativos a sua intensidade, informações térmicas, propriedades específicas (como orientação e escala), ou ainda qualquer informação abstrata [8]. Da mesma forma que nas imagens, podemos associar a determinado ponto em nosso sistema de coordenadas uma cor. Para que se possa ter uma nuvem de pontos colorida, o sistema de cores aditivas RGB é o mais utilizado. Da mesma forma que nas imagens, podemos representar as cores de determinado ponto da nuvem de pontos como um vetor coluna apresentado na Equação 2.3.

É interessante compreender a forma pela qual se dá a aquisição de uma nuvem de pontos para que se entenda a distribuição dos pontos no espaço tridimensional e também a análise que será feita em cima dessa geometria na parte da metodologia desse trabalho. Por mais que existam diversos sistemas que realizem a captura de uma nuvem de pontos, os métodos ópticos vem sendo os preferidos da comunidade acadêmica para o estudo do objeto pois oferecem uma forma eficiente, sem contato e de longa distância tanto em ambientes externos quanto internos [8] [3].

Podemos categorizar as técnicas ópticas baseando-se no princípio de medição que é utilizado, separando-as em técnicas passivas e ativas. Enquanto as técnicas passivas são usadas para cenas que possuem uma quantidade razoável de iluminação ambiente e visa apenas coletar a informação para gerar a nuvem de pontos, as técnicas ativas buscam manipular o ambiente, projetando padrões de luzes estruturados dentro ou fora do espectro de luz visível para aprimorar a aquisão de dados da nuvem de pontos [8]. Em ambas as categorias, o resultado obtido sempre será uma casca gerada pela reflecção da luz em que só a parte exterior daquilo que se deseja capturar é projetado no espaço tridimensional da nuvem de pontos, conforme podemos observar na Figura 2.2.

Assim, como vemos na Figura 2.2, a quantidade de pontos que possuem uma cor associada é mínima, uma vez que todo objeto representado na point cloud terá seu interior

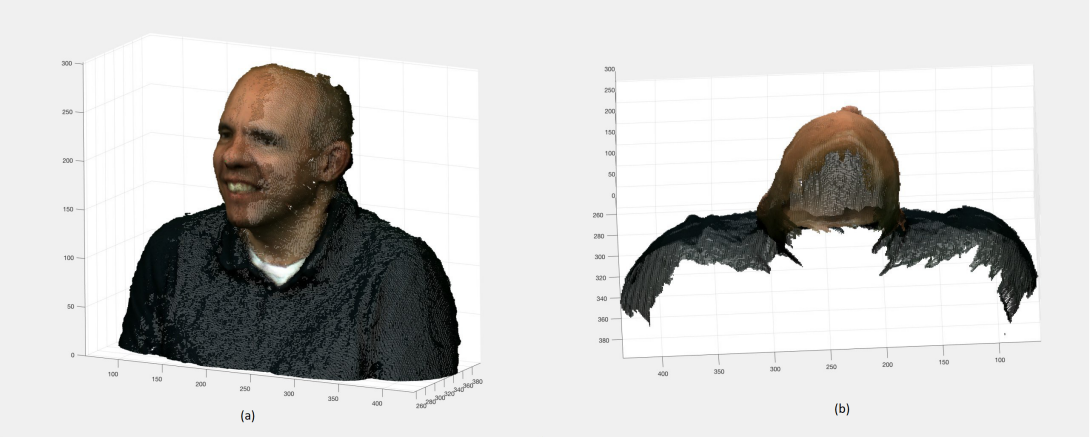

Figura 2.2: (a) Nuvem de pontos frontal (b) Mesma nuvem de pontos que a figura (a) vista de cima.

oco. Denominamos o elemento composto por coordenada e intensidade como voxels (do termo em inglês *volume element*, e dizemos que voxels com uma cor associada são voxels ocupados. Dessa forma, a estrutura de nuvem de pontos é massiva, uma vez que se colocarmos em perspectiva com o tamanho de uma imagem, e considerando que ela só possui em suas informações intensidade e coordenadas ela será pelo menos K vezes maior em espaço de armazenamento computacional que uma imagem de mesma dimensão nos eixos XY, considerando K como o tamanho do eixo Z da nuvem de pontos. Entretanto, como a distribuição de voxels ocupados é esparsa, a point cloud é em geral representada por uma lista [XYZ RGB].

2.3 Entropia da informação

A entropia da informação é a taxa média em que uma informação é produzida por uma fonte que produz dados de forma aleatória. Dado que uma fonte produz símbolos A_i de um conjunto de símbolos possíveis A, então a entropia é o número médio de símbolos binários necessários para codificar a informação que sai de uma fonte. Shannon mostra que o melhor que um codificador sem perdas pode fazer é codificar a saída de uma fonte com um número médio de bits igual a entropia da fonte [10] [11].

A entropia H de uma fonte de informação S que gera N amostras independentes X é definida por Shanon como

$$H(S) = -\sum_{i=1}^{N} P(X_i) \log_2 P(X_i) \tag{2.5}$$

De forma geral, a entropia para uma fonte qualquer (dependente ou independente), pode ser escrita como [10]

$$H(S) = -\lim_{n\to\infty} \frac{1}{n} G_n \tag{2.6}$$

onde

$$G_n = -\sum_{i_1=1}^{i_1=m} \sum_{i_2=1}^{i_2=m} \cdots \sum_{i_n=1}^{i_n=m} P(X_1 = i_1, X_2 = i_2, \ldots, X_n = i_n) \tag{2.7}$$

$$\log P(X_1 = i_1, X_2 = i_2, \ldots, X_n = i_n) \tag{2.8}$$

em que $\{X_1, X_2, \ldots, X_n\}$ é uma sequência de tamanho N da fonte. Note que a Equação 2.7 se torna a Equação 2.5 se as amostras são independentes e identicamente distribuídas. De forma geral, não é possível saber a entropia de uma fonte física, dessa forma temos que estima-la. A estimativa de uma entropia depende das suposições que fazemos a respeito da sequência gerada pela fonte [10]. Considere a seguinte sequência:

$$1 \quad 2 \quad 3 \quad 2 \quad 3 \quad 4 \quad 5 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 8 \quad 9 \quad 10 \tag{2.9}$$

Assumindo que a frequência da ocorrência de cada número é refletida de forma precisa no número de vezes que ele aparece na sequência, podemos estimar a probabilidade da ocorrência de cada símbolo da seguinte forma [10]:

$$P(1) = P(6) = P(7) = P(10) = \frac{1}{16} \tag{2.10}$$

$$P(2) = P(3) = P(4) = P(5) = P(8) = P(9) = \frac{2}{16} \tag{2.11}$$

Assumindo que a sequência seja independentemente e identicamente distribuída, a entropia pode ser calculada pela Equação 2.5 de primeira ordem, como

$$H(S) = -\sum_{i=1}^{10} P(i) \log_2 P(i) \tag{2.12}$$

Resultando em uma entropia para essa fonte de 3.25 bits. Isso significa que o melhor codificador possível para essa sequência poderia apenas codificar ela a 3.25 bits por amostra. De uma maneira geral, fontes que possuem uma distribuição de probabilidade dos elementos gerados mais uniforme são mais difíceis de se comprimir, uma vez que todos os símbolos possuem a mesma probabilidade, resultando em uma alta entropia. Enquanto isso, fontes que possuem distribuição de probabilidade com mais picos, como por exemplo a laplaciana, são mais fáceis de se comprimir, uma vez que com símbolos muito mais prováveis, é reduzida significativamente a entropia da fonte.

2.4 Codificação diferencial

A codificação diferencial é um método de compressão do tipo sem perdas (do termo em inglês *lossless*) [10]. A premissa do algoritmo consiste no fato em que fontes como imagens e fala tem uma alta correlação de amostra para amostra, dessa forma, podemos codificar e transmitir a diferença entre as amostras da sequência ao invés do seu valor original, tentando assim reduzir a quantidade de bits necessárias para o envio de uma sequência de dados [10].

O algoritmo consiste em para uma dada sequência, escrever a primeira amostra igual ao valor original e todos os valores seguintes são codificados como diferenças entre as amostras. De forma simplificada, o valor codificado para um elemento K localizado na posição N de uma sequência de entrada será a amostra correspondente a posição $N-1$ menos a amostra localizada na posição N, com exceção da primeira amostra de um vetor [10]. O valor Y gerado pelo algoritmo de uma amostra X localizada em uma posição N pode ser expresso na forma

$$Y[N] = X[N] - X[N-1] \tag{2.13}$$

Para a reconstrução exata da sequência original basta que a partir do segundo elemento da sequência codificada se some o valor da posição em que se deseja encontrar o código com a posição imediatamente interior. Para demonstrar a codificação diferencial vamos aplica-la na sequência apresentada na Equação 2.14, obtendo

$$1 \quad 1 \quad 1 \quad -1 \quad 1 \quad 1 \quad 1 \quad -1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad -1 \quad 1 \quad 1 \tag{2.14}$$

Essa sequência foi construída usando somente dois valores com probabilidades $P(1) = \frac{13}{16}$ e $P(-1) = \frac{3}{16}$. Realizando o cálculo, a entropia nesse caso fica igual a 0.70 bits por símbolo, reduzindo em 2.55 bits por símbolo da sequência original.

Dessa forma, quando falamos em sinais, o codificador diferencial apresenta seu melhor desempenho em imagens de baixa frequência, uma vez que a transição de intensidade entre os pixels é mais suave e portanto mais previsível, fazendo com que as predições feitas pelo algoritmo tenham valor baixo. E o algoritmo tem o seu pior desempenho para imagens de alta frequência, uma vez que a transição da intensidade entre os pixels tende a ser abrupta e resulta em predições com valores altos, necessitando de uma maior quantidade de bits para representar as diferenças na codificação.

Assim, o decodificador diferencial visa reduzir a quantidade de amostras em dada entrada transformando distribuições que se aproximam do tipo uniforme em distribuições mais próximas de uma laplaciana, zerando a grande maioria dos possíveis elementos, reduzindo assim o número de bits necessários para enviar os dados.

2.5 Codificação de Huffman

A codificação de Huffman é um método de compressão não ambíguo e do tipo sem perdas. A ideia do algoritmo consiste em atribuir códigos de tamanho variável para diferentes caracteres, de forma que os caracateres que forem mais frequentes terão os menores códigos e os menos frequentes terão os maiores códigos.

Os códigos de tamanho variável utilizados são chamados de códigos prefixo, ou seja, os códigos atribuídos pela codificação de Huffman não serão prefixos de nenhum outro código dado a um caractere no mesmo conjunto de dados que se deseja comprimir, garantindo assim que não haja ambiguidade ao decodificar um *bitstream*. [10]

O algoritmo consiste em duas partes principais: a montagem da árvore de Huffman e em seguida a travessia pela árvore para atribuir cada código ao seu respectivo caractere de entrada. Para a montagem da árvore, pega-se todos os caracteres que são únicos presentes no dado que se deseja codificar e se atribui um nó a eles, juntamente com a sua frequência no dado de entrada. Colocam-se todos esses nós em uma fila em que a prioridade será dada aos nós com menor frequência e, enquanto houver mais que um nó nessa fila, pega-se os dois nós com menor frequência e cria-se um nó pai para ambos que terá a frequência resultante dos filhos e o insere na lista de prioridade novamente. Assim, quando restar apenas um nó, teremos a árvore completamente montada. Uma vez com a árvore montada basta percorrer ela do nó que contêm o caractere que se deseja codificar até a raiz para gerar o respectivo código. Chama-se a associação dos símbolos e códigos gerada ao final do algoritmo de dicionário.

A entropia h_i em bits de cada símbolo s_i com probabilidade de ocorrência não nula w_i pode ser dada por

$$h(s_i) = \log_2 \frac{1}{w_i} \tag{2.15}$$

e a entropia H da codificação de Huffman é dada por um somatório ponderado, de todos os caracteres com probabilidade não nula w_i, da entropia de cada símbolo, expresso na forma

$$H = \sum_{w_i > 0} w_i h(s_i) \tag{2.16}$$

que pode ser reescrito como

$$H = \sum_{w_i > 0} w_i \log_2 w_i \tag{2.17}$$

2.6 Estado da Arte

Esta sessão tem como objetivo apresentar brevemente alguns dos métodos que hoje são considerados como estado-da-arte para a compressão das cores de uma point cloud. Seu intuito é mostrar como que o método utilizado nesse trabalho visa experimentar técnicas diferentes do que é considerado como estado-da-arte.

Para a compressão de point clouds, existem métodos já propostos [4] [12] [13] [5] [14] [15] [16] que irão comprimir tanto a geometria quanto suas cores e métodos que somente visam comprimir as cores de uma point cloud [4] [12] , deixando sua geometria para ser comprimida por métodos já conhecidos como o da octree.

Os dois métodos que serão descritos nesse capítulo e que acredito que hoje sejam o estado-da-arte em compressão de cores de point cloud serão tirados dos "Compression of 3D Point Clouds Using a Region-Adaptive Hierarchical Transform" [4] e o "Point Cloud Codec for Tele-Immersive Video" [12], o primeiro visa comprimir somente as cores de uma point cloud, deixando a geometria para métodos conhecidos como o da octtree, enquanto o segundo trata tanto da geometria quanto das cores. Outro método que merece ser citado mas não será descrito por se assimilar com o RAHT é o "Gaussian Process Transform (GPT)" [17], que apresenta resultados de codificação melhores que o RAHT mas não será utilizado como comparação com o trabalho desenvolvido por seu código não ser aberto para a realização de testes.

2.6.1 Region-Adaptive Hierarchical Transform (RAHT)

De forma resumida, a compressão geométrica do RAHT é utilizada em outros métodos e consiste na montagem de uma octree. Para a montagem da octree, imaginemos um cubo com dimensões $S \times S \times S$. No primeiro nível da octree, o cubo é particionado em 8 cubos menores com dimensões $\frac{S}{2} \times \frac{S}{2} \times \frac{S}{2}$, da mesma forma descrita na Figura 2.3. Em um segundo nível, cada cubo pode ser particionado da mesma forma, gerando para cada um mais 8 cubos de dimensões $\frac{S}{4} \times \frac{S}{4} \times \frac{S}{4}$. Esse processo pode ser repetido N vezes, resultando em 2^{3L} voxels com dimensões $2^{-N}S \times 2^{-N}S \times 2^{-N}S$.

A montagem da octtree então se dá de forma que para cada nó corresponde a um nível de subdivisão da point cloud, sendo o nó raiz a point cloud inteira. Caso uma subdivisão da point cloud tenha voxels ocupados, o nó é marcado como 1 e subdivido novamente e caso esteja totalmente vazio é marcado como 0 e vira um nó folha da árvore. Esse processo pode ser descrito de forma resumida pela Figura 2.4

De uma forma resumida, para a codificação das cores o RAHT deriva da ideia de utilizar cores associadas em níveis inferiores da octtree a nodos presentes no próximo nível da árvore. A árvore é percorrida inversamente, dos voxels (nós inferiores) até a point

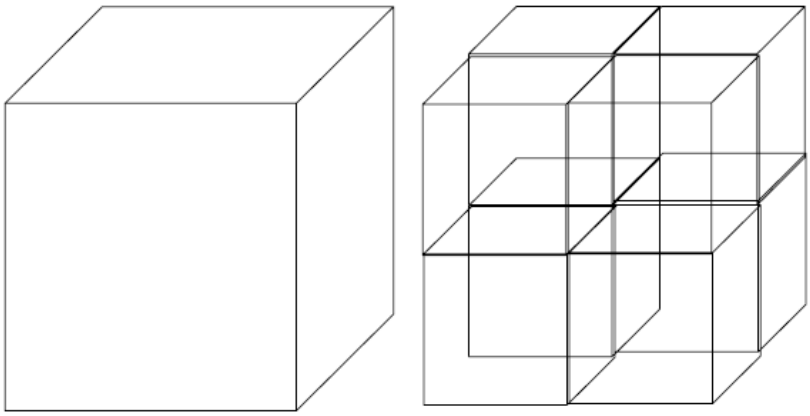

Figura 2.3: Em um nível particular, um cubo é dividido em 8 sub-cubos. Cada sub-cubo pode então ser dividido novamente gerando mais 8 sub-cubos.

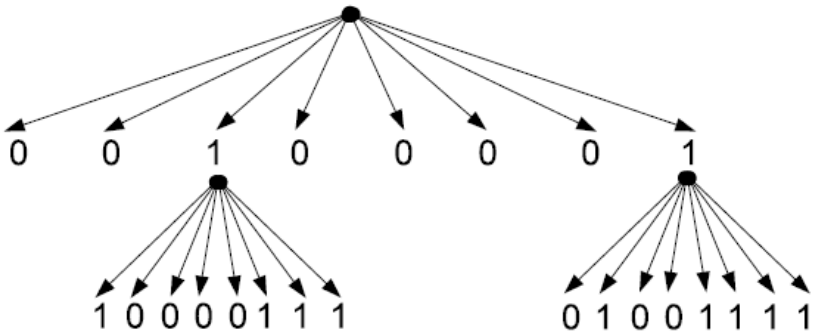

Figura 2.4: Cubos sem voxels ocupados são transformados em nós folha e cubos com voxels ocupados são subdividos novamente.

cloud como um todo (nó raiz), de forma que cada sub-cubo, a cada passo do algoritmo, é recombinado em sub-cubos maiores até chegar a point cloud completa.

Esse processo é feito através de uma série de transformadas envolvendo os pesos de cada conjunto de voxels da point cloud e quando os pesos forem uniformes, ou seja, um sub-cubo tiver todos seus voxels ocupados, o processo se reduz a uma transformada de Haar.

2.6.2 Point Cloud Codec for Tele-Immersive Video

A ideia por trás dessa metodologia é utilizar codecs comuns baseados em octrees (similares ao utilizado no RAHT) para a compressão de point clouds em 3D juntamente com codecs hibridos de vídeo como o MPEG-4 Part 10/AVC [12] e o High Efficiency Video Coding (HEVC) [12], que já tem incluido o compensador de movimento baseado em blocos.

A codificação da geometria ocorre de forma muito similar a do RAHT, em que o codificador irá recursivamente dividir a point cloud em oito blocos, em que somente blocos que contêm algum voxel ocupado continuarão sendo subdivididos, formando uma octree. Cada nível da octree é chamado de level de detalhamento (LOD), e o LOD final da point cloud é especificado nas configurações de bit da octree do codec, utilizando uma diferente configuração baseada na aplicação para qual ele será utilizado.

Para a compressão das cores da point cloud de uma forma eficiente, o codec tem integrado a ele um método baseado no mapeamento do grafo transversal da octree para uma grade de imagem de um JPEG, explorando a relação entre as cores nos $LODs$ finais da octree. A razão por trás do mapeamento se dá no fato de que os atributos de cor relacionados a uma point cloud vem de fontes naturais, de forma que geralmente sempre haverá uma relação entre pixels adjacentes.

De uma forma geral, tanto a metodologia usada na codificação geométrica como a utilizada nas cores são as escolhidas para esse codec pois são rápidas e servem bem para o seu foco, que são aplicações tele-imersivas 3D em tempo real.

Capítulo 3

Metodologia

Este capítulo apresenta o detalhamento de todos os métodos utilizados para a compressão das cores de uma point cloud através da divisão por filamentos e a aplicação posterior da codificação diferencial e codificação de Huffman.

Embora o método possa ser utilizado em qualquer point cloud tridimensional com componentes RGB, ele foi testado em frames de sequências dinâmicas de point clouds voxelizadas, conhecidas como Corpos Superiores Voxelizados da Microsoft (MVUB). Os frames utilizados para os testes podem estar em cubos de $512 \times 512 \times 512$ voxels ou em cubos de $1024 \times 1024 \times 1024$ voxels. E os atributos de cada voxel possuem somente os componentes vermelhos, verdes e azuis para cada superfície ocupada pelos corpos.

As sequências dinâmicas de point clouds voxelizadas das quais as point clouds foram retiradas podem ser encontrados na página JPEG Pleno Database: Microsoft Voxelized Upper Bodies - A Voxelized Point Cloud Dataset [18]. O banco de dados é composto apenas pela captura de sequências de frames de point clouds da parte superior de diferentes indivíduos dentro de um laboratório e foram feitas em câmeras estáticas RGBD, em 30 quadros por segundo, por um período de 7 a 10 segundos cada.

As implementações foram feitas no MATLAB R2017b [19], utilizando funções de Lidar and Point Cloud Processing do Computer Vision System Toolbox [20] para a leitura e processamento das point clouds e também funções do Huffman Coding do Communications Toolbox [21] para realizar a codificação de Huffman dos dados.

Este capítulo irá expor duas diferentes metodologias utilizadas no trabalho realizado, que por mais que sejam executadas de forma diferente, suas etapas assemelham-se bastante, sendo somente executadas em ordem ou em uma diferente organização dos dados da point cloud. Dessa forma, primeiro será explicado cada etapa da metodologia utilizada separadamente, com exemplos aplicados a uma point cloud em cada etapa e em seguida será explicado como essas etapas se integram para gerar duas metodologias distintas.

3.1 Etapas da metodologia

Esta seção visa explicar detalhadamente e mostrar com um exemplo real cada etapa pela qual a point cloud irá passar até ocorrer a escrita do arquivo codificado. A point cloud utilizada para os exemplos citados na metodologia é o arquivo frame0000.ply do conjunto de point clouds 'Ricardo9' que pode ser encontrado na referência [18]. Um fluxograma resumindo as etapas feitas durante a metodologia pode ser apresentado na Figura 3.1. Há quatro processos principais realizados na codificação da point cloud nesse trabalho, são eles: primeiro realizar a divisão da point cloud em camadas para facilitar a estrutura de dados que será trabalhada e também a obtenção de dados; o segundo passo consiste em realizar a divisão dos cortes gerados na etapa anterior em filamentos e obtenção de cores; em seguida, realizar a codificação diferencial e de Huffman das cores dos filamentos gerados para a sua posterior escrita; e por fim, realizar a escrita das cores codificadas dos filamentos para que se possa decodificar o arquivo depois.

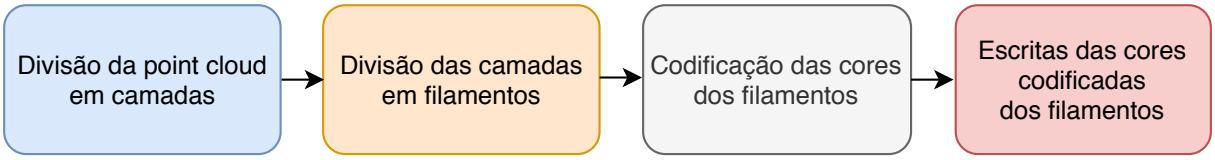

Figura 3.1: Diagrama com os passos para realizar a compressão das cores de uma point cloud com o método proposto

3.1.1 Divisão da point cloud em camadas

Esta seção detalhará os passos feitos na separação da point cloud em vários cortes. O objetivo é reestruturar a point cloud para que a leitura e análise de dados seja mais clara e o procedimento, por sua vez, mais ordenado. A point cloud será lida pelo codificador, dividida em vários cortes baseado no eixo de referência passado e as imagens serão binarizadas. Um fluxograma com as etapas da separação da point cloud em cortes é apresentado na Figura 3.2

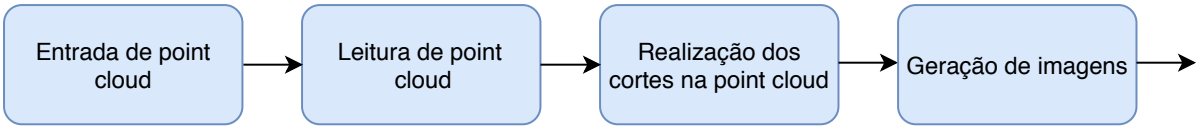

Figura 3.2: Diagrama com os passos para realizar a separação de uma point cloud em cortes

Inicialmente, recebe-se a entrada da point cloud com dimensões XYZ e cores RGB, que serão um conjunto de voxels tridimensionais a serem lidos. Uma vez lida, a point cloud pode ser acessada pelo algoritmo e pode ser vista no Matlab conforme a Figura 3.3.

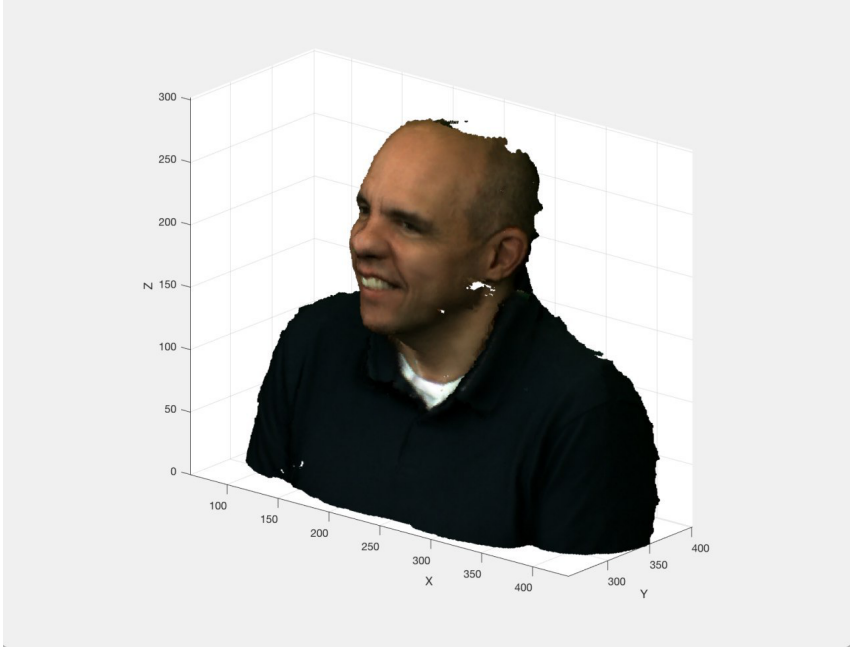

Figura 3.3: Point Cloud sem nenhum processamento

Em seguida, deve-se fazer cortes na point cloud para que se obtenha as estruturas em 2D desejadas. Gerar um corte na point cloud no contexto deste trabalho consiste em fixar um dos três eixos XYZ da point cloud e pegar todo o conjunto de pontos daquele eixo fixado, pegando dessa forma uma fatia bidimensional da point cloud e representar ela por uma imagem. Um exemplo contendo quatro cortes realizados na point cloud apresentada na Figura 3.3 pode ser visto na Figura 3.4.

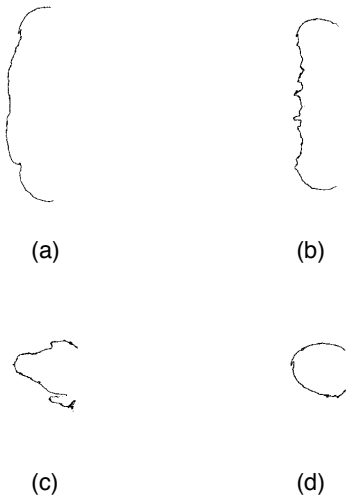

Figura 3.4: Cortes realizados na point cloud da Figura 3.3 (a) Corte realizado com o eixo z fixo em 0; (b) Corte realizado com o eixo z fixo em 90; (c) Corte realizado com o eixo z fixo em 150; (d) Corte realizado com o eixo z fixo em 270

É importante que se faça uma análise da point cloud para que se defina o melhor eixo para realizar os cortes. O eixo escolhido para realizar a divisão da point cloud em camadas neste trabalho foi o eixo Z, pois de forma geral, como as imagens tendem a estar em posição vertical, o esperado é que a divisão nesse eixo seja a que consegue manter os filamentos com as cores mais suaves (com menos variação), de forma que a expectativa é que seja mais fácil de se comprimir.

Na geração dos cortes é realizada uma binarização dos cortes gerados e a conversão destes em imagens de forma que voxels vazios tenham a intensidade do pixel com valor 1 e voxels ocupados tenham a intensidade do pixel com valor 0. As imagens (a), (b), (c) e (d) presentes na Figura 3.4 são exemplos de imagens binarizadas geradas por esse processo. Assim, para cada valor de Z teremos uma imagem binária com as dimensões XY que estarão com seus pixels com o valor 0 somente nos pontos que possuem cor na point cloud original. Esse processo é feito para a posterior obtenção de cores na point cloud.

3.1.2 Divisão das camadas em filamentos

Essa seção visa apresentar os detalhes de como é realizada a divisão em filamentos das imagens obtidas no procedimento anterior e a posterior obtenção de cores na point cloud original, além da motivação por trás da divisão de filamentos. O objetivo aqui é, para cada imagem gerada no passo anterior, segmentar todos seus filamentos e pegar as cores pertencentes a cada filamento para a posterior compressão. Um fluxograma com as etapas da divisão dos cortes em filamentos e obtenção das cores é apresentado na Figura 3.5

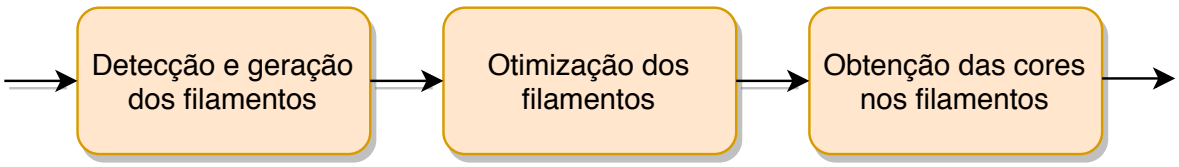

Figura 3.5: Diagrama com os passos para realizar a detecção dos filamentos e obtenção das cores

Com os cortes convertidos em imagens, deve-se agora detectar e segmentar os filamentos. Um filamento no contexto desse trabalho é uma lista de coordenadas próximas umas das outras em que duas coordenadas adjacentes sempre estarão dentro de uma distância R, definida nas configurações do algoritmo, que quando vistas a olho nu em um corte são identificadas como uma linha contínua. Um exemplo de um corte que idealmente só contêm um filamento quando observado a olho nú, pode ser visto na Figura 3.6.

É importante ressaltar que o algoritmo de detecção e geração de filamentos sempre que executado encontre exatamente os mesmo filamentos para um dado corte. Isso é

Figura 3.6: Representação de um filamento ideal no corte feito com o eixo z=0

fundamental uma vez que não é do nosso interesse passar informações a respeito da geometria na codificação das cores da point cloud, tornando o método mais flexível para utilizá-lo com outros métodos de compressão de geometria e reduzindo o tamanho do arquivo codificado final. Assim, fazendo com que para que se decodifique posteriormente as cores, basta rodar o algoritmo de detecção de filamentos novamente no destino, uma vez que a geometria deve ser codificada sem perdas [4].

Para a detecção e geração de filamentos o primeiro processo realizado em cada corte consiste em gerar uma lista de todos os filamentos existentes nele. A primeira coisa que a função de obtenção dos filamentos faz é receber o corte no qual serão detectados e gerados os filamentos. Uma vez recebido, identifica qual foi o eixo utilizado para realizar o corte e qual a coordenada do eixo que aquele corte representa.

Em seguida, gera uma tabela de coordenadas com todos os pontos que tem intensidade diferente do branco no corte. É importante notar que essa tabela é gerada percorrendo a imagem da esquerda para a direita e de cima para baixo. Então inicia-se um laço, enquanto essa tabela com as coordenadas não estiver vazia o laço continuará. Nesse loop pega-se uma linha da tabela de coordenadas, adiciona as coordenadas a lista que constitui um filamento e a exclui da tabela. Essa linha representa um ponto que terá seus vizinhos analisados por uma função auxiliar, que retornará o índice da tabela de coordenadas com o pixel que tem a menor distância do ponto que está sendo analisado e qual é essa distância. Se essa distância retornada for menor que o limiar que define qual o máximo da distância entre dois pixels em um mesmo filamento, a função retorna para o início e começa a análise de um novo pixel, caso seja maior que o limiar 10 (valor definido no

método), aquele filamento é dado como terminado e o algoritmo começa a analisar outro indice na tabela considerado como um novo filamento. A parte da repetição descrita pode ser visualizado de forma simplificada no Algoritmo 1.

```
initialization;
index ← 1;
j ← 1;
i ← 1;
limiar ← 10;
while not(isempty(tabela_de_pixels)) do
    lista_filamentos{j, i} ← tabela_de_pixels(index, :);
    tabela_de_pixels(index, :) = [];
    if not(isempty(tabela_de_pixels)) then
        [index, distancia] ←
          função_auxiliar(lista_filamentos{j, i}, tabela_de_pixels);
    else
        dist ← 0;
    end
    if distancia > limiar then
        j ← j + 1;
        i ← 1;
    end
end
```

Algoritmo 1: Descrição do loop para montagem dos filamentos

A função auxiliar utilizada no loop simplesmente irá traçar a distância do pixel sendo analisado com todas as coordenadas presentes na tabela de pixel, pegar a menor distância e retornar qual é o índice na tabela que contêm as coordenadas do pixel que tem essa distância e qual é ela. Ao executar a função elaborada no corte representado pela Figura 3.6, obtemos os filamentos representados na Figura 3.7.

Como podemos observar, foram cinco os filamentos detectados e não somente um filamento ideal como o esperado quando observamos a Figura 3.6. Dessa forma, é interessante realizar a tentativa de otimizações na obtenção dos filamentos para tentar chegar o mais próximo possível de um filamento só no caso desse corte em específico.

Diferentes alternativas foram pensadas na elaboração da metodologia para corrigir esse problema. A primeira tentativa realizada consistiu em mudar a forma como o algoritmo monta a tabela de coordenadas para ao invés de ser da esquerda para a direita e de

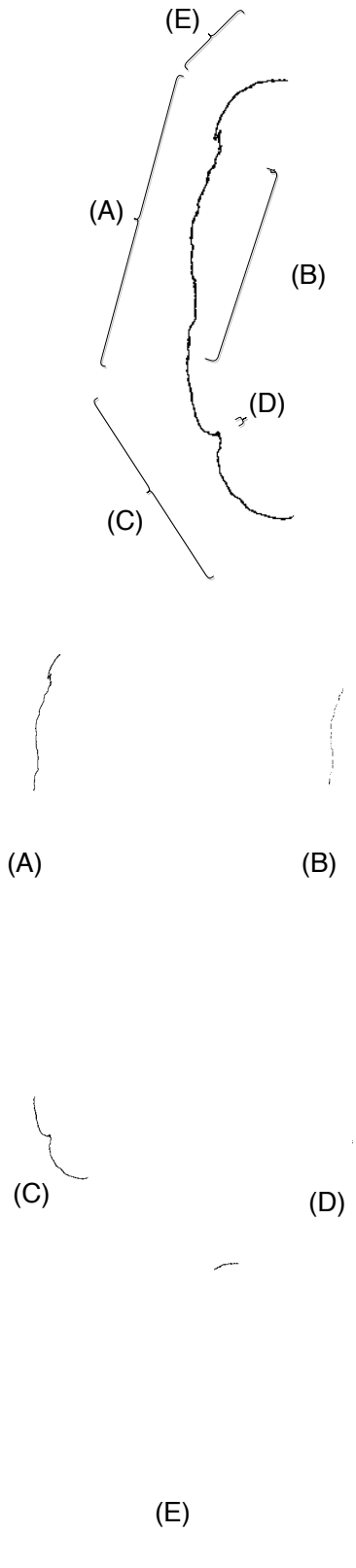

Figura 3.7: Filamentos detectados através do algoritmo aplicado ao corte relativo ao eixo z=0, com indicação da sua localização no corte. Filamento (D) é composto por 3 pixels.

cima para baixo, fosse diagonalmente percorrendo a imagem ou inversamente do sentido original. Essa ideia surgiu no fato de que como a montagem de filamentos começa pelo primeiro bit encontrado no percorrimento da imagem, situações como a que ocorrem na Figura 3.7 nos filamentos (A) e (E) sejam normais, em que o primeiro bit encontrado é o localizado no topo do filamento (A) e consistentemente nas primeiras iterações os bits mais próximos estão sempre no sentido para baixo, isolando assim o filamento (E) e tornando o que na teoria seria somente um filamento em dois.

Com a implementação de uma diferente abordagem na forma como o algoritmo identificaria os bits, observamos que por mais que essa implementação resolvesse parcialmente casos como o da Figura 3.7, piorava a identificação de outros cortes em que os pixels se encontrassem distribuídos de uma forma diferente e, mesmo quando melhorava, a geração de falhas como nos filamentos (B) e (D) poderiam vir a ocorrer aumentando ainda mais a quantidade filamentos, fazendo com que descartássemos essa possibilidade de otimização.

Uma segunda tentativa de otimização consistiu em pegar as coordenadas das pontas de todos os filamentos gerados e fazer uma nova função de verificação de distância entre eles. Novamente, para casos como o da Figura 3.7 nos filamentos (A) e (E) houve a junção em somente um filamento, mas fez com que em alguns cenários houvessem incosistências na estrutura do filamento como um todo, de forma que os melhores resultados obtidos na codificação foram sem a realização dessa otimização, fazendo com que também a descartássemos. Dessa forma, os filamentos foram deixados da forma como foram detectados e o algoritmo está pronto para iniciar a obtenção das cores nos filamentos.

Com os filamentos gerados para a imagem referente ao corte, para cada imagem o algoritmo percorrerá a lista que contêm todos os filamentos e para cada coordenada presente na lista que constitui um filamento o algoritmo agora irá associar uma cor, com os seus três canais RGB. Para isso há uma função que receberá a lista de coordenadas, o eixo em que o corte foi realizado na point cloud para a geração das imagens, a point cloud original e a coordenada do ponto de corte. A função irá comparar as coordenadas do filamento gerado com a point cloud original e retornará os três canais de cores na mesma ordem em que as coordenadas se encontravam na lista em uma matriz $N \times 3$, em que N corresponde ao número de coordenadas presentes na lista e cada coluna corresponde a um canal de cor RGB.

3.1.3 Codificação das cores dos filamentos

Essa seção visa apresentar os detalhes de como é realizada a codificação diferencial das cores obtidas dos filamentos, a motivação por trás da codificação diferencial e também a codificação de Huffman. O objetivo aqui é, para cada canal de cor presente em cada filamento realizar a compressão diferencial seguida de uma compressão de Huffman. Um

fluxograma com as etapas da codificação das cores do filamentos é apresentado na Figura 3.8

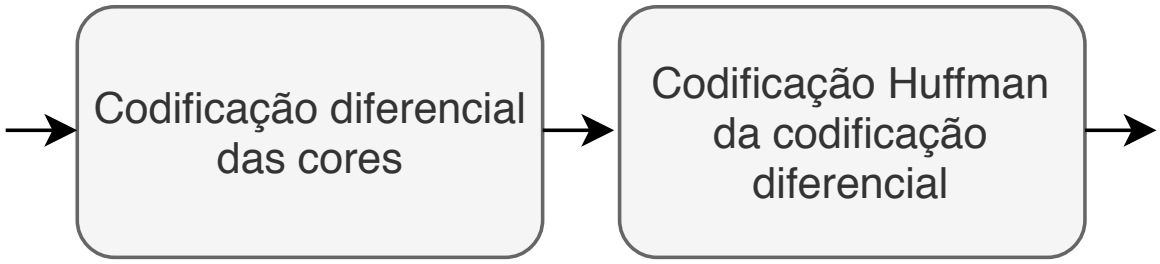

Figura 3.8: Diagrama com os passos para realizar a codificação das cores dos filamentos

Com os três canais de cores obtidos para todos os filamentos gerados em um corte, o algoritmo irá realizar a codificação diferencial das cores desses canais. A codificação consiste em, para cada filamento, manter o primeiro valor de cada canal de cor idêntico ao obtido pela comparação direta com a point cloud e todos os valores conseguintes do primeiro serão predições dos valores anteriores tomando o primeiro elemento como partida. A predição é feita através de somas e subtrações de forma que o elemento $N + 1$ sempre conterá o número que deve ser somado ao elemento N, seja ele positivo ou negativo, para obter o valor real correspondente a cada coordenada de filamento.

Dessa forma, uma vez que há a expectativa da grande maioria das cores de um dado filamento serem suaves (com valores próximos), espera-se diminuir a quantidade de diferentes valores para os canais de cores presentes no filamento. Assim, ao aplicar a codificação de Huffman sobre os dados, espera-se melhorar significativamente os resultados de uma compressão utilizando esse método, uma vez que um conjunto de informações com a menor quantidade de variáveis possível gerará o menor arquivo possível para um arquivo codificado por codificação de Huffman.

Em seguida, com os canais de cores comprimidos diferencialmente o algoritmo irá agora realizar a compressão de Huffman desses dados. A compressão de Huffman feita é a mais básica possível, utilizando a função de codificação do próprio Matlab, sem agrupamento de símbolos. Uma vez realizada, a função já retorna o bitstream pronto e o dicionário para posterior decodificação.

3.1.4 Escrita das cores codificadas dos filamentos

Essa seção visa apresentar os detalhes de como foi realizada a codificação dos dicionários gerados pela compressão de Huffman e como os bitstream das cores e dos dicionários foram organizados para a escrita. Um fluxograma com as etapas da montagem dos bitstream e escrita das cores codificadas é apresentado na Figura 3.9.

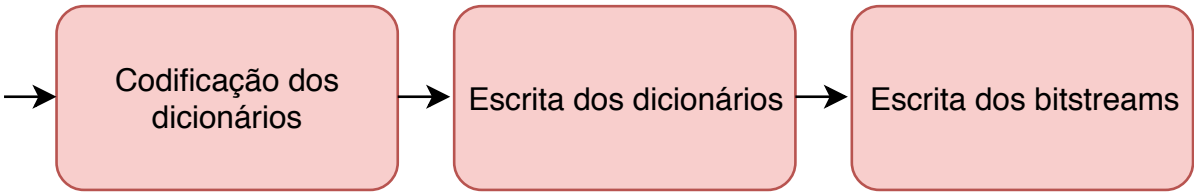

Figura 3.9: Diagrama com os passos para realizar a codificação dos dicionários gerados e da escrita do bitstream final

Com as cores de todos dos filamentos codificadas pela codificação de Huffman, deve-se codificar os dicionários para realizar a transmissão. Para realizar a codificação dos dicionários foi utilizado um esquema inspirado na metodologia de *run length coding*. É importante dizer que as entradas do dicionário encontram-se em ordem crescente organizadas pelos símbolos e seus símbolos podem variar entre -255 até 255, devido a codificação diferencial.

A primeira coisa que o algoritmo faz é armazenar no primeiro byte do bitstream que representará o dicionário quantos símbolos aquele dicionário possui. Em seguida, o algoritmo toma como base o valor -255 e calcula qual é o valor que deve ser somado a ele para chegar ao primeiro símbolo do dicionário. Assim, no próximo byte ele escreve o tamanho do código gerado para aquele símbolo, repetindo o processo até o dicionário ser totalmente percorrido. Ao final, escreve todos os códigos do dicionário em sequência.

É possível que o valor que tenha que ser somado ao valor base seja maior que 255 (capacidade de um byte), caso isso aconteça, ele escreverá 255 em um byte e no byte seguinte escreverá zero, para indicar que não foi encontrado o símbolo, escrevendo assim quanto falta a partir da soma de 255. A estrutura geral do bitstream pode ser observada na Figura 3.10

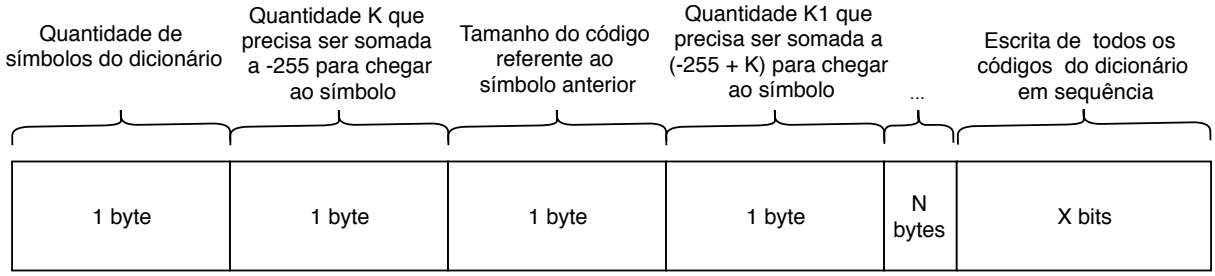

Figura 3.10: Representação visual do bitstream dos dicionários gerados pela codificação de Huffman.

Com o código gerado, o dicionário é imediatamente escrito na bitstream seguido da bitstream que ele irá decodificar. A ordem de codificação dos canais de cores é sempre vermelho, verde e azul. O bitstream escrito depois do dicionário codificado é o gerado pela função de codificação de Huffman do Matlab, mas antes dele é adicionado um cabe-

çalho de 2 bytes que dirá a quantidade de bits total do bitstream daquele canal para a posterior decodificação. Dessa forma, a estrutura geral do bitstream pode ser observada na Figura 3.11

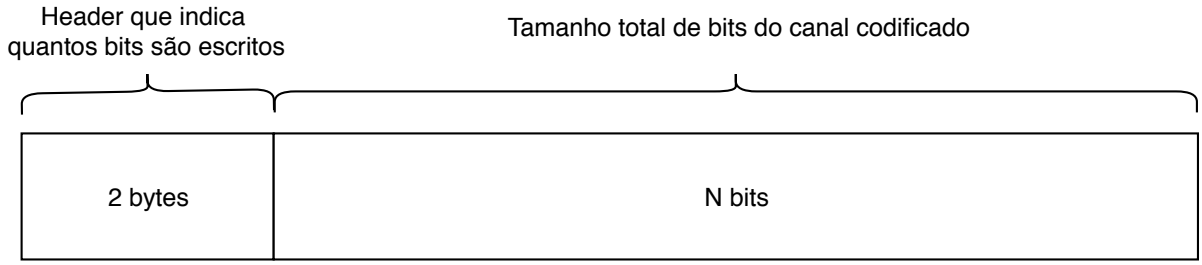

Figura 3.11: Representação visual do bitstream dos canais de cores.

3.2 Integração das etapas

Essa seção tem como objetivo apresentar as duas formas distintas pelas quais as etapas apresentadas foram organizadas para gerar duas metodologias diferentes. Uma das metodologias iremos chamar de metodologia três dicionários por filamento (MTDF), pois gera três dicionários para cada filamento de um determinado corte e a metodologia três dicionários por corte (MTDC), que somente gera três dicionários por corte.

3.2.1 Metodologia três dicionários por filamento (MTDF)

A metodologia três dicionários por filamento foi a primeira abordagem ao problema utilizada. Os procedimentos realizados para sua elaboração são feitos na mesma ordem em que são apresentados na seção 4.1. Sua principal característica consiste no fato de que para cada filamento em um corte, cada canal de cor será codificado diferencialmente, passará pela codificação de Huffman e terá um dicionário para ser decodificado, totalizando três dicionários por filamento em um corte. Fazendo com que um corte que apresente N filamentos tenha um total de $3 \times N$ dicionários. Dessa forma, o bitstream final será a junção da Figura 3.10 e da Figura 3.11 para cada canal de cor de cada filamento.

3.2.2 Metodologia três dicionários por corte (MTDC)

A metodologia três dicionários por corte foi uma metodologia desenvolvida para tentar reduzir a quantidade de dicionários que a MTDF possui, que reduz significativamente o tamanho do arquivo final. Para entender melhor como esse método funciona, observe a Figura 3.12

Canal de cor vermelha dos filamentos para um corte

Filamento 1	233	234	235	236	237	
Filamento 2	0	0	0			
Filamento 3	133	134	135	136	135	
Filamento 4	132	133	133	132		
Filamento 5	77	77	77	78	77	76

Figura 3.12: Representação visual do canal de cor vermelha de todos os filamentos de um determinado corte.

A Figura 3.12 representa uma situação comum dos canais de cores de determinado corte logo após a detecção dos filamentos e obtenção de cores. Logo em seguida, o algoritmo passará a compressão diferencial em cada um desses filamentos, resultando na Figura 3.13

Canal de cor vermelha codificados diferencialemente de todos filamentos de um corte

Filamento 1	233	1	1	1	1	
Filamento 2	0	0	0			
Filamento 3	133	1	1	1	-1	
Filamento 4	132	1	0	-1		
Filamento 5	77	0	0	1	-1	-1

Figura 3.13: Representação visual do canal de cor vermelha de todos os filamentos de um determinado corte codificados diferencialmente.

No MTDF cada um desses filamentos teria um dicionário para o canal de cor vermelha, o Filamento 1 com um dicionário de 2 símbolos, o Filamento 2 com um dicionário de 1

símbolo, o Filamento 3 com um dicionário de 3 símbolos, o Filamento 4 com um dicionário de 4 símbolos e o Filamento 5 com um dicionário de 4 símbolos.

O que foi observado é que com a codificação diferencial, a partir do primeiro elemento de cada filamento, os valores são semelhantes e na grande maioria das vezes se repetem em quase todos os filamentos. Dessa forma no MTDC é gerado um bitstream a mais que reduzirá significativamente a quantidade de entradas no dicionário.

A geração desse bitstream se dá na primeira coluna dos filamentos conforme mostrado na Figura 3.13, uma vez que a probabilidade do primeiro elemento de cada filamento se repetir é extremamente baixa e na grande maioria das vezes ele será o único presente em todo conjunto de dados após a codificação diferencial. O bitstream gerado utiliza seu primeiro byte para indicar quantos filamentos aquele conjunto de dados terá, depois utiliza $N \times bytes \times 2$ em que N será o valor escrito no primeiro byte, utilizando dois bytes por filamento para indicar quantos bits representam as cores de cada filamento e em seguida utilizará um byte para indicar qual é o valor do primeiro elemento de cada filamento. Dessa forma, a estrutura geral do bitstream que codifica os primeiros elementos de cada filamento pode ser observada na Figura 3.14

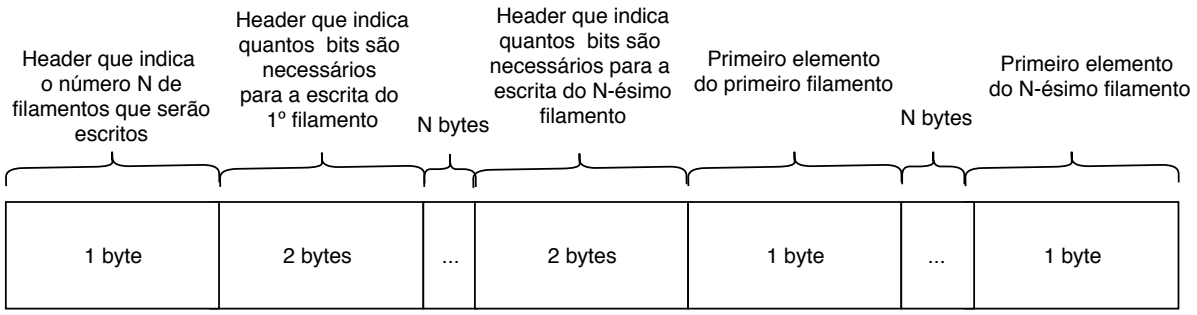

Figura 3.14: Representação visual do bitstream contendo os primeiros elementos de cada filamento.

Com o bitstream dos primeiros elementos gerado aplica-se o Huffman agora para todos os outros dados que não sejam os primeiros elementos dos filamentos, gerando somente um dicionário por canal de cor por corte. No exemplo citado na Figura 3.13, seria gerado somente um dicionário com 3 símbolos, ao invés dos 5 dicionários distintos pela MTDF, reduzindo significativamente a quantidade de bits utilizadas no arquivo codificado final.

Assim, o bitstream final gerado para essa metodologia é primeiramente o apresentado na Figura 3.14, seguido do bitstream apresentado na Figura 3.10 e por fim o bitstream apresentado na Figura 3.11 para cada canal de cor representando todos os filamentos de cada corte. Tornando essa metodologia especialmente interessante por transformar um corte com vários filamentos em uma representação única com só um dicionário por canal

de cor, aproximando-se do que seria a representação ideal de um só filamento por corte, amenizando os problemas citados na parte de otimização dos filamentos.

Capítulo 4

Resultados

Este capítulo tem por intuito expor os Resultados obtidos com ambas as metodologias apresentadas nesse trabalho e compará-las com um dos métodos de compressão de cores de point cloud que hoje é considerado como estado-da-arte, o RAHT. Serão apresentados também algumas informações gerais a respeito do método, como quantidade de filamentos e seu tamanho para a realização de estudos futuros e aprimoramento do método utilizado.

O algoritmo foi testado em 6 point clouds diferentes. Todas as point clouds utilizadas são frames de vídeos que podem ser encontradas no JPEG Pleno Database [18]. Os frames utilizados para a geração de resultados podem estar em cubos de até $512 \times 512 \times 512$ voxels ou em cubos de até $1024 \times 1024 \times 1024$ voxels. E os atributos de cada voxel possuem somente os componentes vermelhos, verdes e azuis para cada superfície ocupada pelos corpos.

Diferentes frames de diferentes vídeos foram utilizados para testar o algoritmo de forma que a disposição de voxels nas point clouds fossem as mais variadas o possível. A point cloud 1 foi escolhida por ter as cores distribuídas de forma uniforme, sendo um bom exemplo do melhor caso para a utilização do algoritmo desenvolvido; a point cloud 2 foi escolhida por apresentar os membros estendidos, de forma que o esperado é que se gere múltiplos filamentos em um mesmo corte; a point cloud 3 foi escolhida pela alta variação de cores na camisa listrada e também pelo ruído apresentado, sendo um dos piores casos para o algoritmo desenvolvido; a point cloud 4 foi escolhida pela alta variação de cores na camisa quadriculada, sendo um dos piores cenários para o algoritmo desenvolvido; a point cloud 5 foi escolhida por apresentar os membros estendidos, gerando múltiplos filamentos por corte; e por fim, a point cloud 6 foi escolhida por ser bem maior que as point clouds anteriores e ter uma grande variação de cores não linear no vestido, diferentemente das camisas listradas e quadriculadas apresentadas na point cloud 4 e 5.

As implementações foram feitas no MATLAB R2017b [19] e testadas em um notebook com macOS High Sierra versão 10.13.6, equipado com um processador 2,9 GHz Intel Core

i5, 8 GB de memória e placa de vídeo Intel Iris Graphics 550 1536MB.

Para cada point cloud será apresentado um histograma da distribuição do tamanho de todos os filamentos da point cloud, uma tabela indicando qual o tamanho do arquivo final, a PSNR e a taxa em bits por voxel ocupado (bpov) com diferentes metodologias aplicadas nas cores do point cloud e também um gráfico da taxa em bpov pelo PSNR dos arquivos codificados pelo RAHT com diferentes deltas para ver a partir de qual delta o método desenvolvido obtém melhores resultados que o RAHT.

É importante lembrar que ambas as metodologias apresentadas neste trabalho são *lossless* e inerentemente terão um resultado em termos de tamanho em bytes utilizados na codificação piores do que o RAHT, que não é *lossless*. Dessa forma, iremos comparar os resultados obtidos sempre com o RAHT com um o parâmetro delta de codificação o menor possível, fazendo com que se tenha a menor quantidade de perdas possível, que por mais que se aproxime de uma compressão lossless, ainda terá distorção entre a imagem decodificada e a codificada.

Tabela 4.1: Resultados para a point cloud 1.

Método de codificação	**Resultado (em KBs)**	**Resultado (em bpov)**	**PSNR (dB)**
MTDF	458	17.04	∞
MTDC	370	13.76	∞
RAHT (delta = 2)	176	6.53	50.79
RAHT (delta = 1)	262	9.78	53.68
RAHT (delta = 0.5)	331	12.35	56.98
RAHT (delta = 0.25)	410	15.27	60.35
RAHT (delta = 0.15)	465	17.35	62.13
RAHT (delta = 0.1)	500	18.64	64.87

4.1 Point Cloud 1

Esta primeira point cloud é o primeiro frame do vídeo Ricardo9 retirado da JPEG Pleno Database, o vídeo foi gravado em laboratório e possui dimensões de $512 \times 512 \times 512$ voxels com componentes RGB. A Figura 4.1 apresenta duas perspectivas da point cloud utilizada para o teste, a Figura 4.2 apresenta o histograma com a distribuição dos tamanhos dos filamentos de toda a point cloud, a Figura 4.3 apresenta um gráfico de PSNR por taxa em bpov dos arquivos codificados com o RAHT com diferentes deltas e o MTDC e a Tabela 4.1 apresenta um comparativo dos métodos expostos nesse trabalho com um dos métodos considerados como o estado-da-arte, o RAHT. A point cloud possui 214656 voxels ocupados.

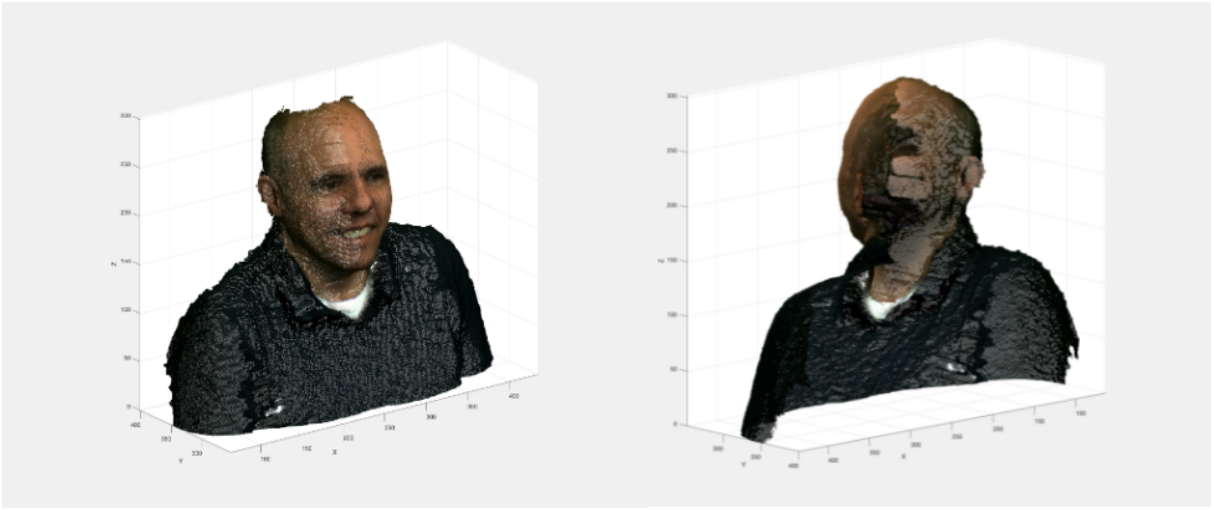

Figura 4.1: Primeiro frame do vídeo Ricardo9 retirado do JPEG Pleno Database.

O esperado para uma point cloud como a da estrutura apresentada na Figura 4.1 é que a metodologia apresente um bom resultado, uma vez que todos os cortes que serão feitos no eixo Z ao olho nu gerarão idealmente só um filamento. Assim, pelo histograma apresentado fica evidente a necessidade de realizar uma melhor aquisição e otimização

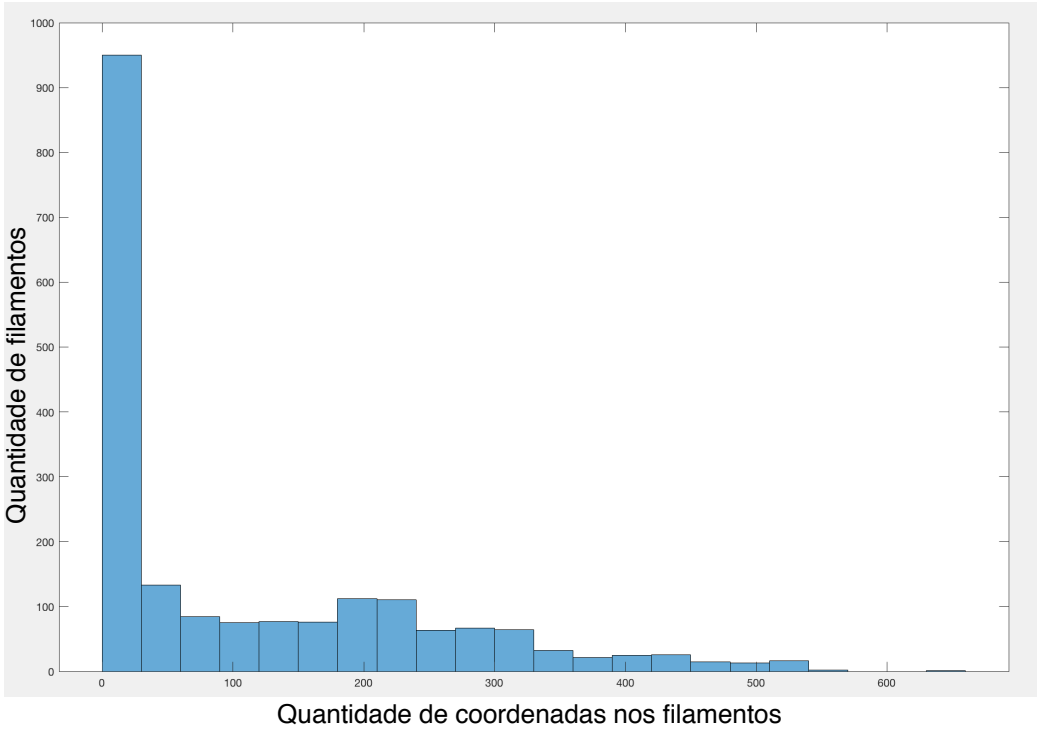

Figura 4.2: Histograma do tamanho dos filamentos da point cloud 1.

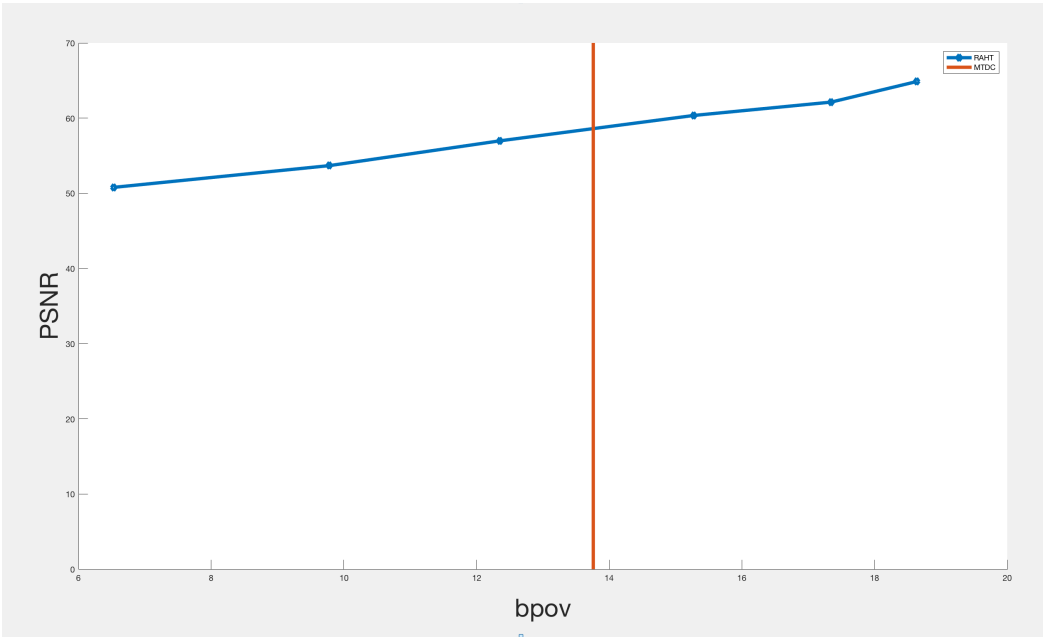

Figura 4.3: Gráfico de comparação da taxa do RAHT com diferentes deltas com o MTDC aplicados a point cloud 1

dos filamentos, uma vez que olhando a imagem a olho nu não vemos nem de perto a quantidade de pequenos filamentos soltos apresentada.

Como podemos observar pelos resultados, o MTDC se saiu bem melhor do que o

MTDF como já era esperado e o resultado para a aplicação do RAHT com o delta igual a 0.25, ficou 40 KBs maior que o método de compressão desse trabalho, de forma que para valores de delta do RAHT menores que esse o algoritmo desenvolvido apresenta um melhor desempenho que o estado da arte.

Tabela 4.2: Resultados para a point cloud 2.

Método de codificação	**Resultado (em KBs)**	**Resultado (em bpov)**	**PSNR (dB)**
MTDF	846	20	∞
MTDC	614	14.56	∞
RAHT (delta = 2)	300	7.1	50.80
RAHT (delta = 1)	432	10.23	53.69
RAHT (delta = 0.5)	543	12.85	57
RAHT (delta = 0.25)	663	15.68	60.30
RAHT (delta = 0.15)	748	17.72	62.12
RAHT (delta = 0.1)	803	19.03	64.76

4.2 Point Cloud 2

A segunda point cloud é o frame de número 91 do vídeo Ricardo9 retirado da JPEG Pleno Database, o vídeo foi gravado em laboratório e possui dimensões de $512 \times 512 \times 512$ voxels com componentes RGB. A Figura 4.4 apresenta duas perspectivas da point cloud utilizada para o teste, a Figura 4.5 apresenta o histograma com a distribuição dos tamanhos dos filamentos de toda a point cloud, a Figura 4.6 apresenta um gráfico de PSNR por taxa em bpov dos arquivos codificados com o RAHT com diferentes deltas e o MTDC e a Tabela 4.2 apresenta um comparativo dos métodos expostos nesse trabalho com um dos métodos considerados como o estado-da-arte, o RAHT. A point cloud possui 337803 voxels ocupados.

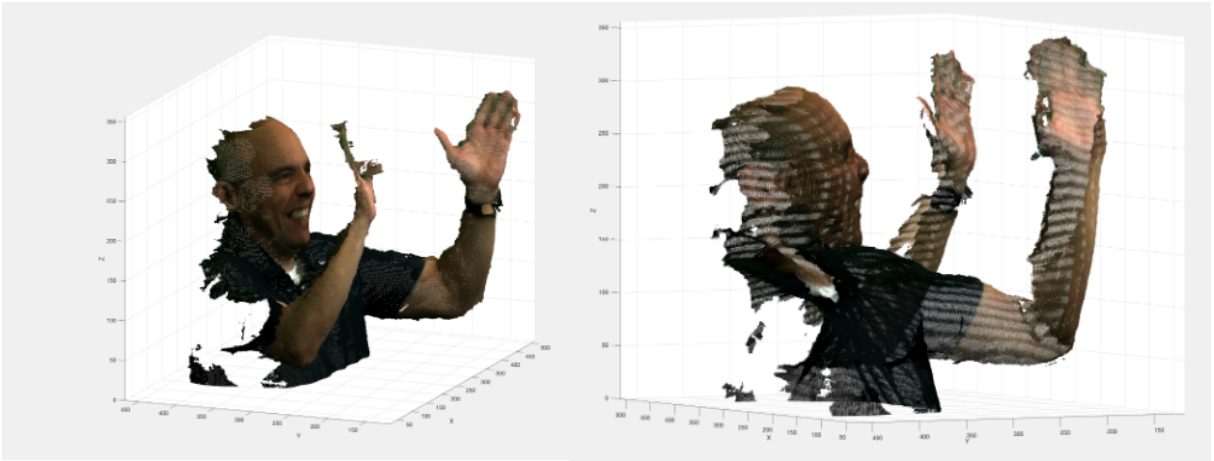

Figura 4.4: Frame de número 91 do vídeo Ricardo9 retirado do JPEG Pleno Database.

A estrutura da point cloud 2 apresentada na Figura 4.4 é um pouco diferente do que a point cloud 1 apresentada na Figura 4.1, os braços levantados fazem com que a transição das cores entre a camisa e a pele fique mais brusca, potencialmente gerando dicionários maiores mas o esperado ainda é que a metodologia apresente um bom resultado, uma vez

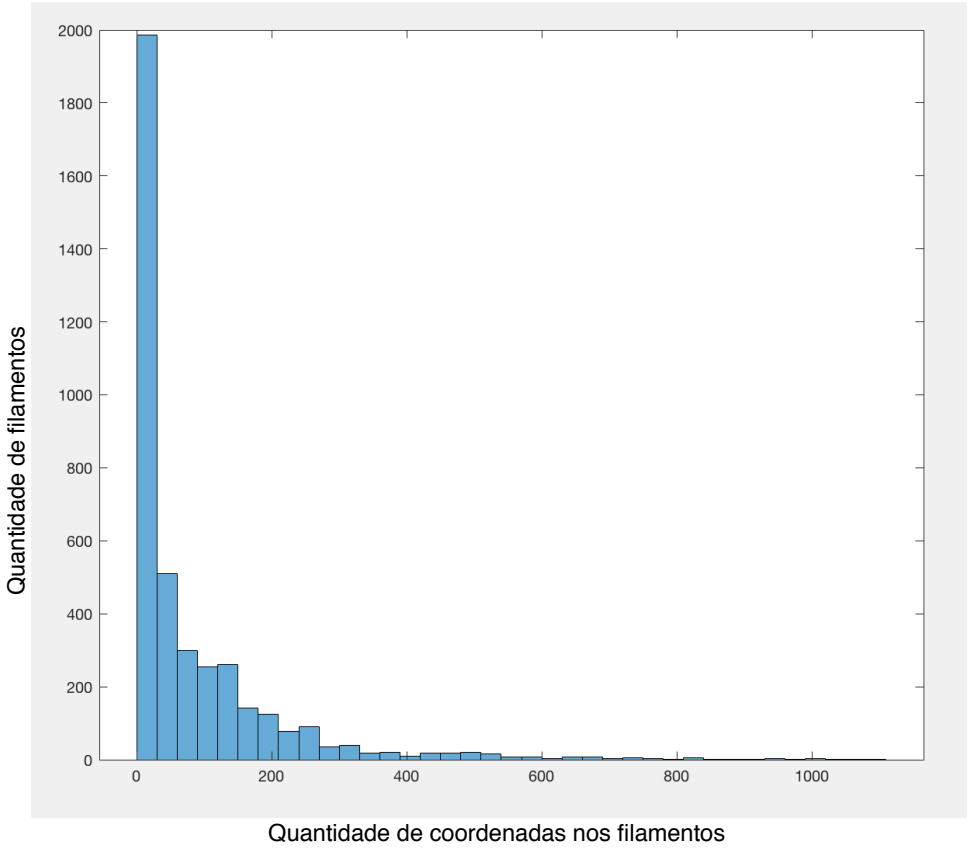

Figura 4.5: Histograma do tamanho dos filamentos da point cloud 2.

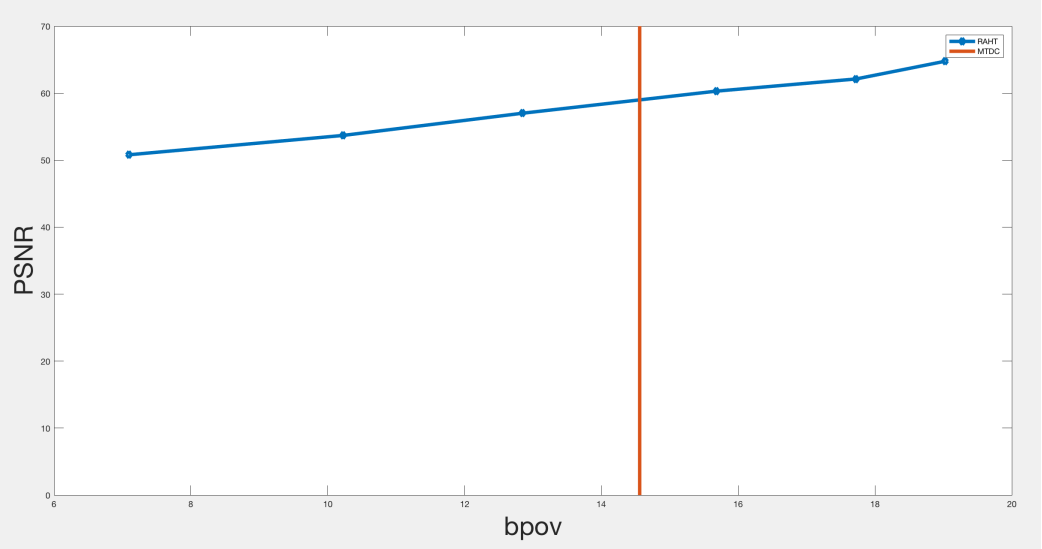

Figura 4.6: Gráfico de comparação da taxa do RAHT com diferentes deltas com o MTDC aplicados a point cloud 2

que passada a mudança brusca de cores entre pele e camisa a transição das cores volta a ser constante.

Pelo histograma apresentado fica mais evidente a necessidade de realizar uma melhor aquisição e otimização dos filamentos, uma vez que olhando a imagem a olho nú não vemos nem de perto a quantidade de pequenos filamentos soltos apresentada, sendo a quantidade dessa point cloud mais do que o dobro da point cloud anterior.

Como podemos observar pelos resultados, o MTDC se saiu bem melhor do que o MTDF como já era esperado e o resultado para a aplicação do RAHT com o delta igual a 0.25 ficou 49 KBs maior que o melhor método de compressão feito nesse trabalho, de forma que para valores de delta do RAHT menores que esse o algoritmo desenvolvido apresenta um melhor desempenho que o estado da arte.

Tabela 4.3: Resultados para a point cloud 3.

Método de codificação	**Resultado (em KBs)**	**Resultado (em bpov)**	**PSNR (dB)**
MTDF	1552	32.24	∞
MTDC	1067	22.16	∞
RAHT (delta = 2)	565	11.72	50.55
RAHT (delta = 1)	709	14.72	53.43
RAHT (delta = 0.5)	851	17.67	56.84
RAHT (delta = 0.25)	994	20.62	59.91
RAHT (delta = 0.15)	1102	22.86	61.85
RAHT (delta = 0.1)	1178	24.46	63.95

4.3 Point Cloud 3

A terceira point cloud é o frame de número 116 do vídeo Phil9 retirado da JPEG Pleno Database, o tamanho do frame é de 8.6 MBs e o vídeo foi gravado em laboratório e possui dimensões de $512 \times 512 \times 512$ voxels com componentes RGB. A Figura 4.7 apresenta duas perspectivas da point cloud utilizada para o teste, a Figura 4.8 apresenta o histograma com a distribuição dos tamanhos dos filamentos de toda a point cloud, a Figura 4.9 apresenta um gráfico de PSNR por taxa em bpov dos arquivos codificados com o RAHT com diferentes deltas e o MTDC e a Tabela 4.3 apresenta um comparativo dos métodos expostos nesse trabalho com um dos métodos considerados como o estado-da-arte, o RAHT. A point cloud possui 385467 voxels ocupados.

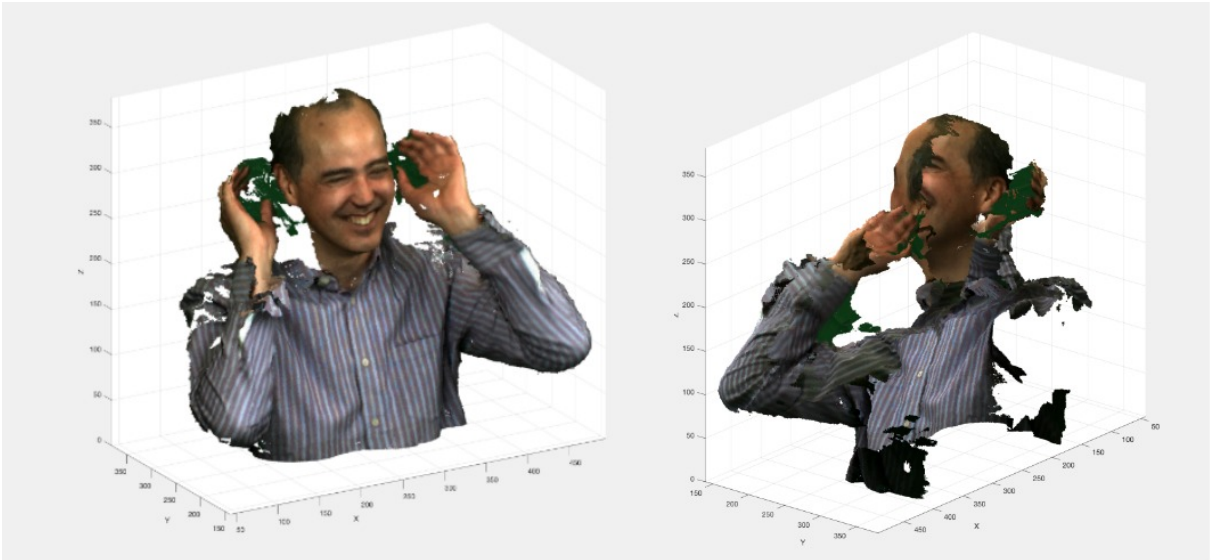

Figura 4.7: Frame de número 116 do vídeo Phil9 retirado do JPEG Pleno Database.

A estrutura da point cloud 3 apresentada na Figura 4.7 possui bastante ruído e tem uma característica de cores que se difere das duas point clouds anteriores, a camisa por ser

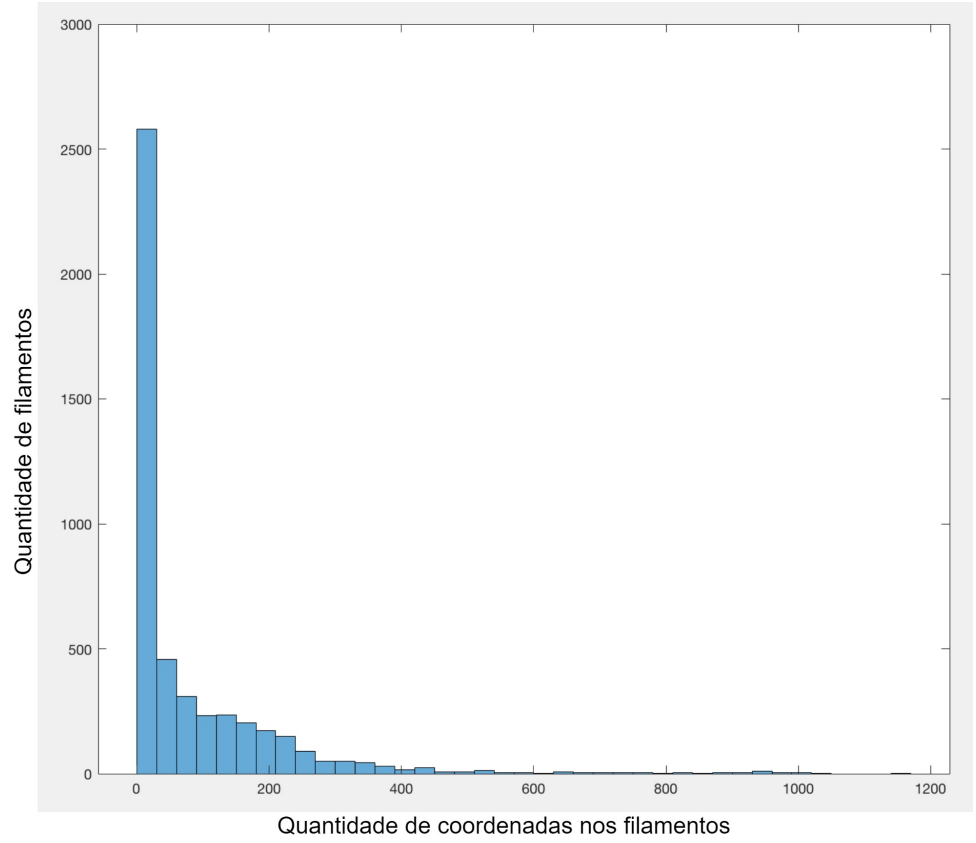

Figura 4.8: Histograma do tamanho dos filamentos da point cloud 3.

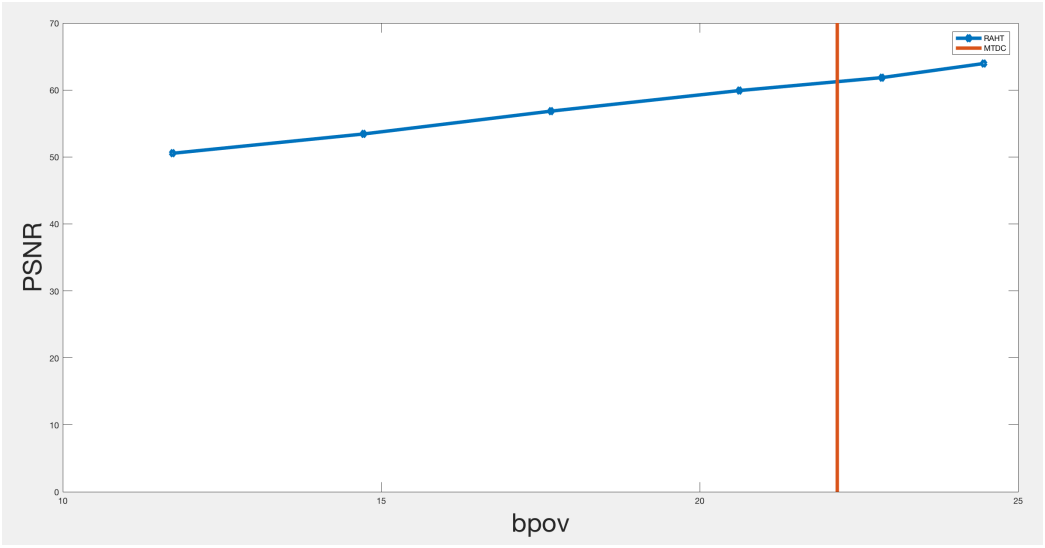

Figura 4.9: Gráfico de comparação da taxa do RAHT com diferentes deltas com o MTDC aplicados a point cloud 3

listrada fará com que os as cores dos filamentos oscilem constantemente, potencialmente piorando o desempenho da codificação diferencial e gerando um arquivo maior, não só isso a point cloud 3 também é bem mais densa em voxels do que as duas point clouds

anteriores.

Pelo histograma apresentado fica evidente a necessidade de realizar uma melhor aquisição e otimização dos filamentos, uma vez que olhando a imagem a olho nu não vemos nem de perto a quantidade de pequenos filamentos soltos apresentada.

Como podemos observar pelos resultados, o MTDC se saiu bem melhor do que o MTDF como já era esperado e o resultado para a aplicação do RAHT com o delta igual a 0.15, ficou 35 KBs maior que o melhor método de compressão feito nesse trabalho, de forma que para valores de delta do RAHT menores que esse o algoritmo desenvolvido apresenta um melhor desempenho que o estado da arte.

Tabela 4.4: Resultados para a point cloud 4.

Método de codificação	**Resultado (em KBs)**	**Resultado (em bpov)**	**PSNR (dB)**
MTDF	1142	32.88	∞
MTDC	852	24.56	∞
RAHT (delta = 2)	443	12.78	50.54
RAHT (delta = 1)	546	15.73	53.43
RAHT (delta = 0.5)	648	18.68	56.88
RAHT (delta = 0.25)	750	21.63	59.93
RAHT (delta = 0.15)	828	23.87	61.87
RAHT (delta = 0.1)	883	25.45	63.89

4.4 Point Cloud 4

A quarta point cloud é o frame de número 263 do vídeo Andrew9 retirado da JPEG Pleno Database, o tamanho do frame é de 6.2 MBs e o vídeo foi gravado em laboratório e possui dimensões de 512 × 512 × 512 voxels com componentes RGB. A Figura 4.10 apresenta duas perspectivas da point cloud utilizada para o teste, a Figura 4.11 apresenta o histograma com a distribuição dos tamanhos dos filamentos de toda a point cloud, a Figura 4.12 apresenta um gráfico de PSNR por taxa em bpov dos arquivos codificados com o RAHT com diferentes deltas e o MTDC e a Tabela 4.4 apresenta um comparativo dos métodos expostos nesse trabalho com um dos métodos considerados como o estado-da-arte, o RAHT. A point cloud possui 277583 voxels ocupados.

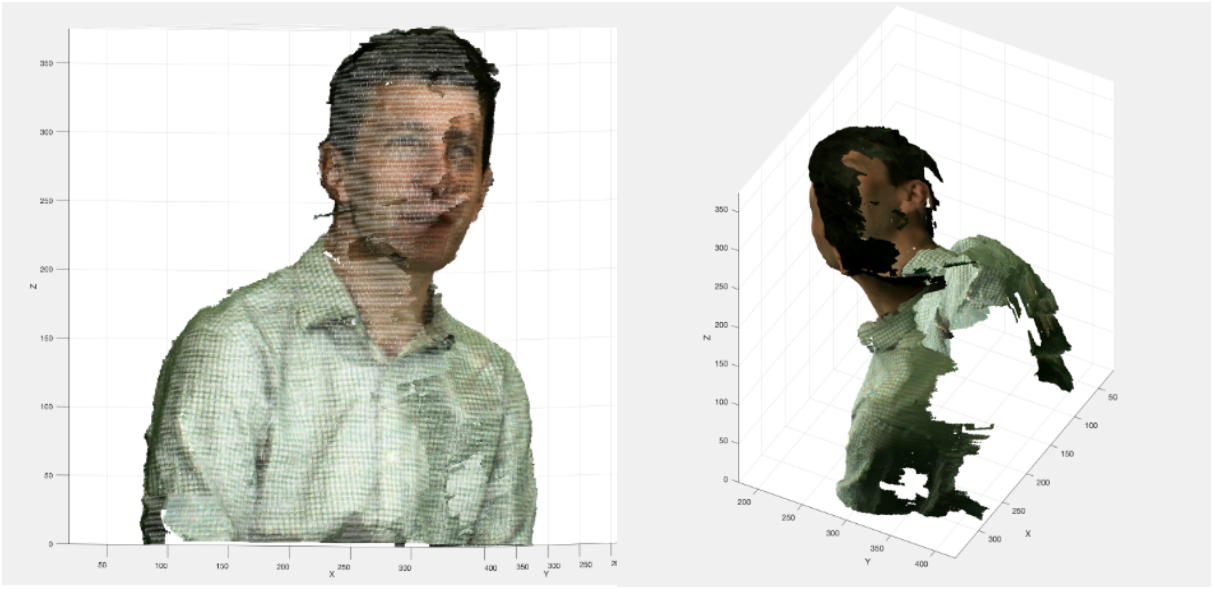

Figura 4.10: Frame de número 263 do vídeo Andrew9 retirado do JPEG Pleno Database.

A estrutura da point cloud 4 apresentada na Figura 4.10 possui menos ruído que a anterior uma vez que o vídeo só tem movimentos faciais e tem uma característica de

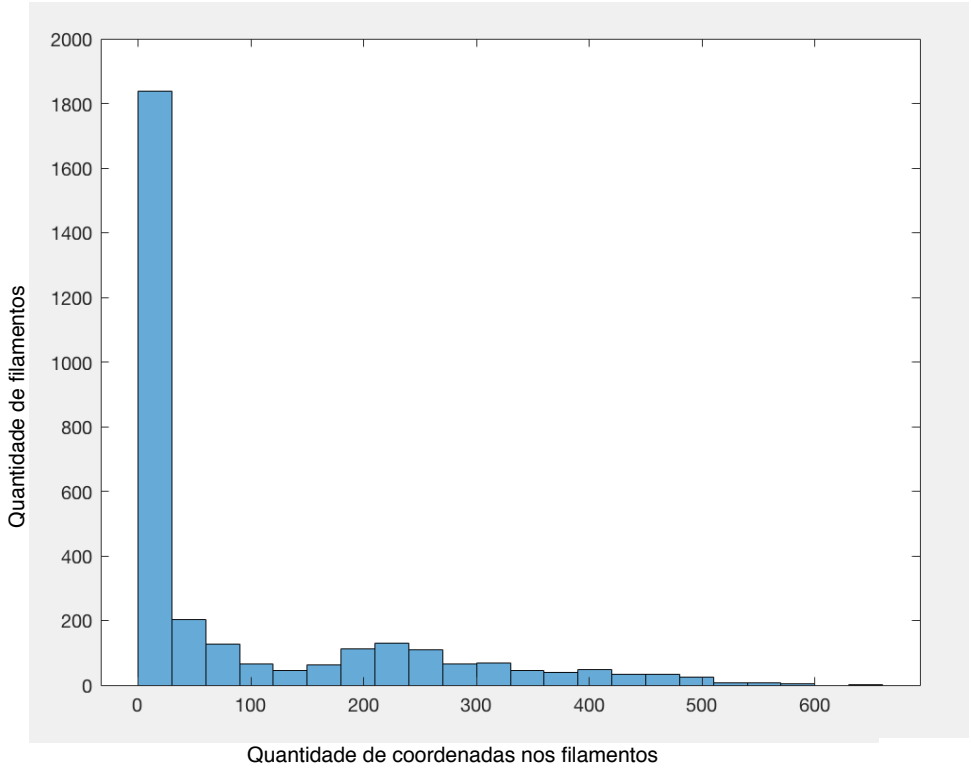

Figura 4.11: Histograma do tamanho dos filamentos da point cloud 4.

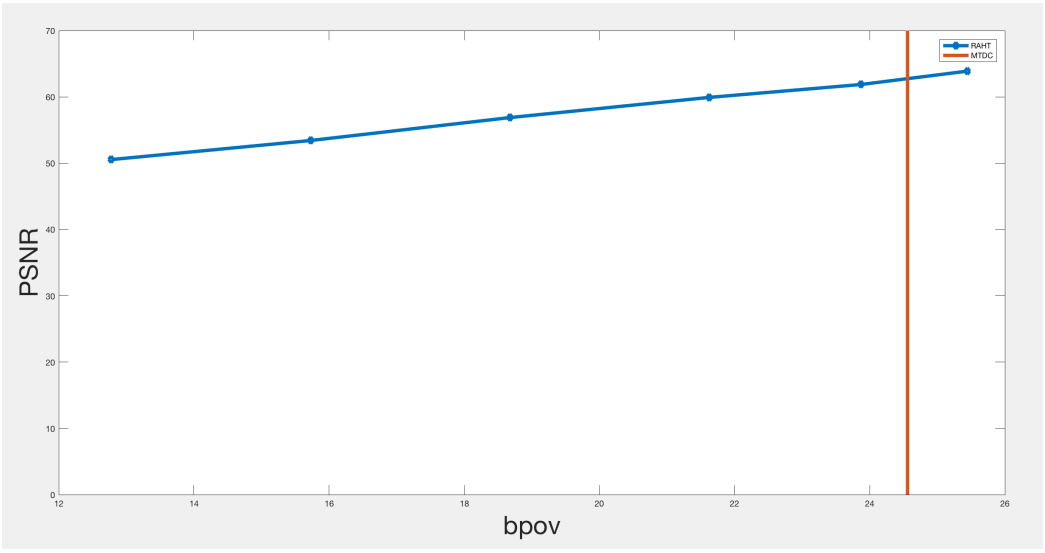

Figura 4.12: Gráfico de comparação da taxa do RAHT com diferentes deltas com o MTDC aplicados a point cloud 4

cores que se difere das duas primeiras point clouds, a camisa por ser quadriculada fará com que os as cores dos filamentos oscilem constantemente, potencialmente piorando o desempenho da codificação diferencial e gerando um arquivo maior.

Pelo histograma apresentado fica evidente a necessidade de realizar uma melhor aquisição e otimização dos filamentos.

Como podemos observar pelos resultados, o MTDC se saiu bem melhor do que o MTDF como já era esperado e o resultado para a aplicação do RAHT com o delta igual a 0.1, ficou 31 KBs maior que o melhor método de compressão feito nesse trabalho. No entanto, o resultado do MTDC obteve uma taxa maior que 24 bpov, sendo mais interessante enviar as cores da point cloud sem compressão, uma vez que 24 bits por voxel para as cores é o mesmo que enviar sem compressão.

Tabela 4.5: Resultados para a point cloud 5.

Método de codificação	**Resultado (em KBs)**	**Resultado (em bpov)**	**PSNR (dB)**
MTDF	810	21.28	∞
MTDC	627	16.48	∞
RAHT (delta = 2)	252	6.63	51.31
RAHT (delta = 1)	361	9.49	54.14
RAHT (delta = 0.5)	454	11.92	57.50
RAHT (delta = 0.25)	543	14.25	60.75
RAHT (delta = 0.15)	606	15.92	62.58
RAHT (delta = 0.1)	648	17.03	64.99

4.5 Point Cloud 5

A quinta point cloud é o frame de número 146 do vídeo Sarah9 retirado da JPEG Pleno Database, o tamanho do frame é de 6.3 MBs e o vídeo foi gravado em laboratório e possui dimensões de $512 \times 512 \times 512$ voxels com componentes RGB. A Figura 4.13 apresenta duas perspectivas da point cloud utilizada para o teste, a Figura 4.14 apresenta o histograma com a distribuição dos tamanhos dos filamentos de toda a point cloud, a Figura 4.15 apresenta um gráfico de PSNR por taxa em bpov dos arquivos codificados com o RAHT com diferentes deltas e o MTDC e a Tabela 4.5 apresenta um comparativo dos métodos expostos nesse trabalho com um dos métodos considerados como o estado-da-arte, o RAHT. A point cloud possui 304597 voxels ocupados.

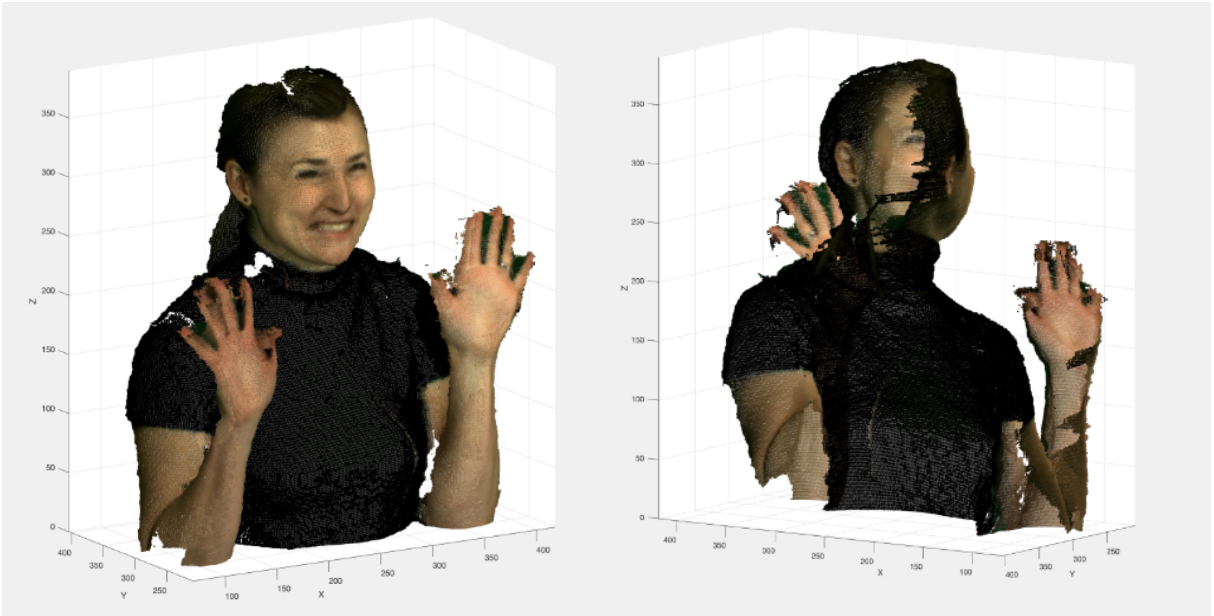

Figura 4.13: Frame de número 146 do vídeo Sarah9 retirado do JPEG Pleno Database.

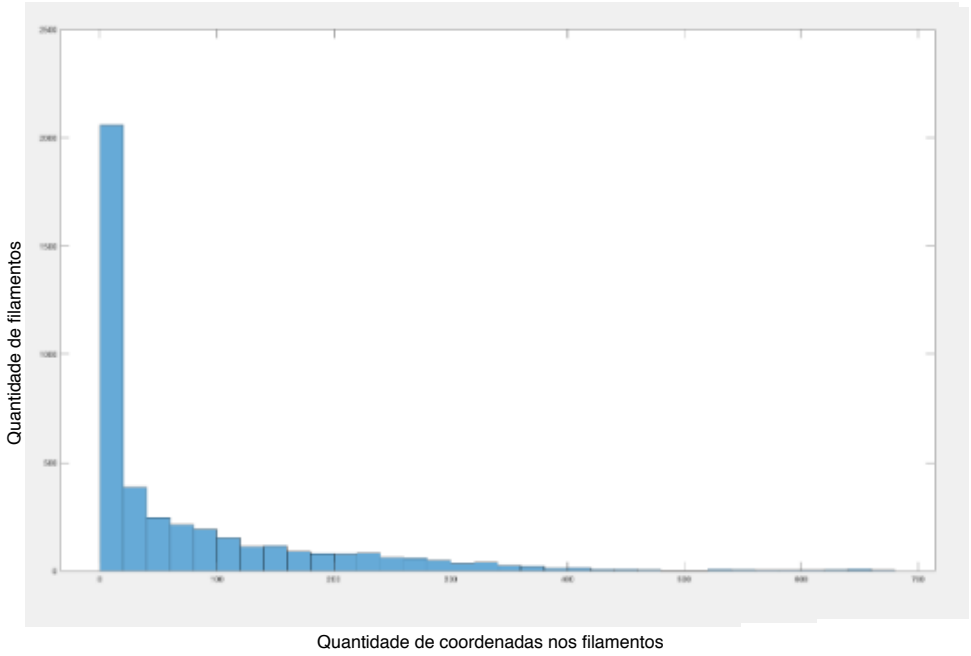

Figura 4.14: Histograma do tamanho dos filamentos da point cloud 5.

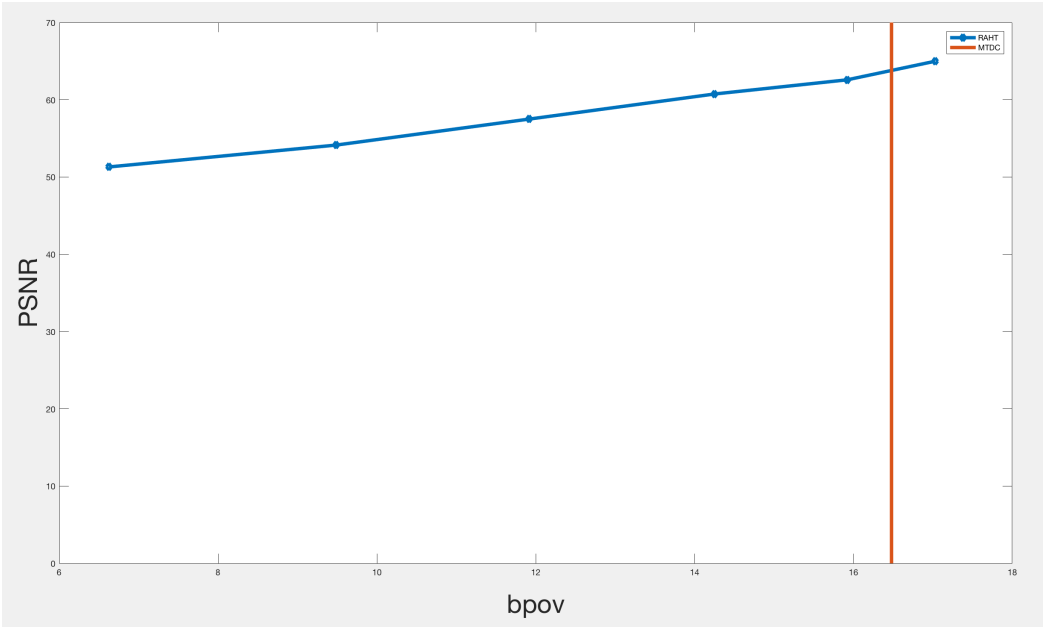

Figura 4.15: Gráfico de comparação da taxa do RAHT com diferentes deltas com o MTDC aplicados a point cloud 5

A estrutura da point cloud 5 apresentada na Figura 4.13 possui bastante ruído uma vez que o frame foi retirado no momento em que no vídeo a Sarah balança os dedos e os braços. Pelo histograma apresentado fica evidente a necessidade de realizar uma melhor aquisição e otimização dos filamentos, mesmo com os filamentos dos dedos que naturalmente serão pequenos a quantidade de filamentos pequenos ainda não é justificável.

Como podemos observar pelos resultados, o MTDC se saiu bem melhor do que o MTDF como já era esperado e o resultado para a aplicação do RAHT com o delta igual a 0.1 ficou 21 KBs maior que o melhor método de compressão feito nesse trabalho, de forma que para valores de delta do RAHT menores que esse o algoritmo desenvolvido apresenta um melhor desempenho que o estado da arte.

4.6 Point Cloud 6

A sexta point cloud é o frame de número 106 do vídeo Red and Black retirado da JPEG Pleno Database [22], o tamanho do frame é de 16.5 MBs e o vídeo foi gravado em laboratório e possui dimensões de $1024 \times 1024 \times 1024$ voxels com componentes RGB. A Figura 4.16 apresenta duas perspectivas da point cloud utilizada para o teste, a Figura 4.17 apresenta o histograma com a distribuição dos tamanhos dos filamentos de toda a point cloud, a Figura 4.18 apresenta um gráfico de PSNR por taxa em bpov dos arquivos codificados com o RAHT com diferentes deltas e o MTDC e a Tabela 4.6 apresenta um comparativo dos métodos expostos nesse trabalho com um dos métodos considerados como o estado-da-arte, o RAHT. A point cloud possui 765626 voxels ocupados.

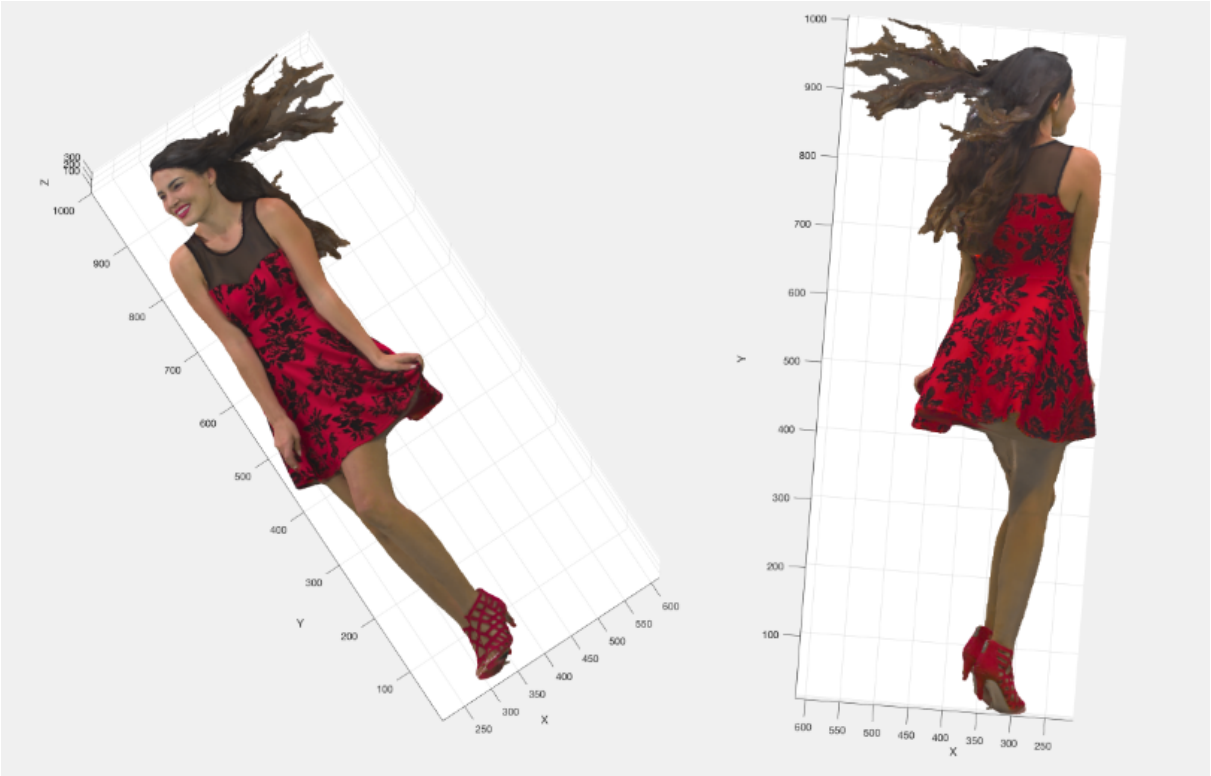

Figura 4.16: Frame de número 146 do vídeo Red and Black retirado do JPEG Pleno Database.

A estrutura da point cloud 6 apresentada na Figura 4.16 é bem diferente de todas as estruturas anteriores, uma vez que essa pega o corpo inteiro enquanto as outras só capturaram o tronco superior. O vestido da mulher possui muitas variações de cores e o cabelo ao vento gera uma série de filamentos diferentes dos padrões anteriores. Pelo histograma apresentado fica evidente a necessidade de realizar uma melhor aquisição e otimização dos filamentos, o número de filamentos pequenos nessa imagem é muito grande,

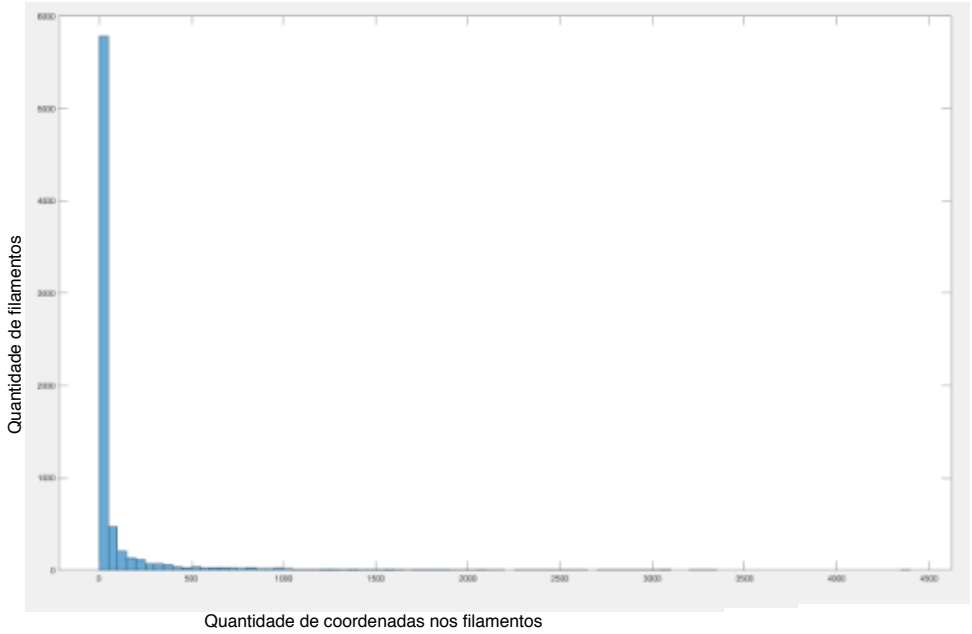

Figura 4.17: Histograma do tamanho dos filamentos da point cloud 6.

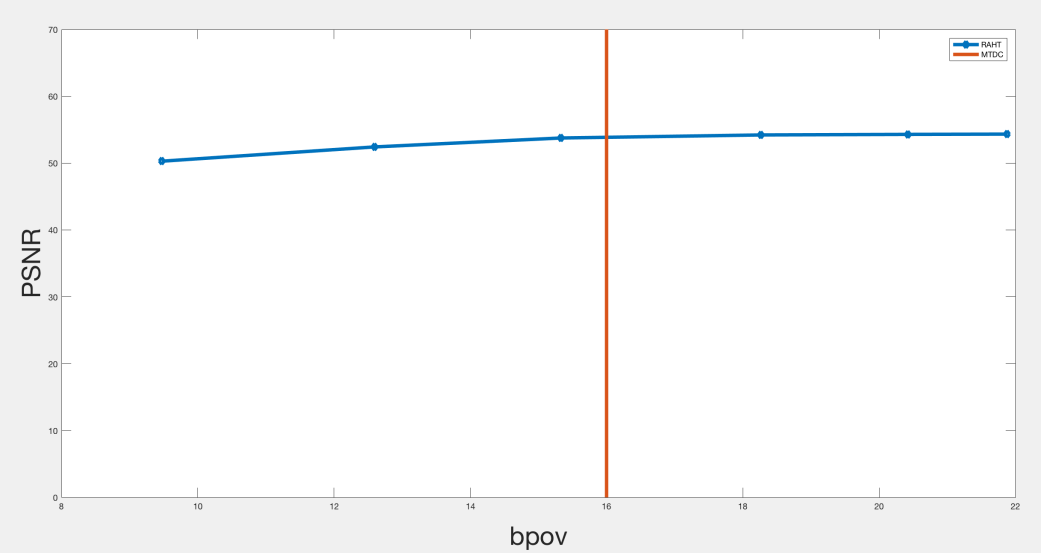

Figura 4.18: Gráfico de comparação da taxa do RAHT com diferentes deltas com o MTDC aplicados a point cloud 6

sendo maior que o dobro da terceira point cloud apresentada, prejudicando o desempenho da codificação significamente.

Como podemos observar pelos resultados, o MTDC se saiu bem melhor do que o MTDF como já era esperado e o resultado para a aplicação do RAHT com o delta igual a 0.25 ficou 213 KBs maior que o melhor método de compressão feito nesse trabalho, de forma que para valores de delta do RAHT menores que esse o algoritmo desenvolvido apresenta um melhor desempenho que o estado da arte.

Tabela 4.6: Resultados para a point cloud 6.

Método de codificação	**Resultado (em KBs)**	**Resultado (em bpov)**	**PSNR (dB)**
MTDF	2048	21.44	∞
MTDC	1535	16	∞
RAHT (delta = 2)	906	9.4	50.28
RAHT (delta = 1)	1205	12.59	52.42
RAHT (delta = 0.5)	1467	15.33	53.76
RAHT (delta = 0.25)	1748	18.26	54.21
RAHT (delta = 0.15)	1955	20.42	54.30
RAHT (delta = 0.1)	2094	21.88	54.34

Capítulo 5

Conclusão e Trabalhos Futuros

Este capítulo apresenta as conclusões preliminares a respeito do trabalho e aponta direções para trabalhos futuros.

5.1 Conclusões preliminares

Com a criação de novas tecnologias e o aprimoramento das já existentes, junto com o crescimento de indústrias que podem se beneficiar da utilização de point clouds para inovar o seu negócio, aliadas a criação de dispositivos de captação de point clouds de baixo custo, como o Kinect, e aprimoramento de dispositivos de captação de point clouds já existentes, é fundamental que o processamento desse tipo de dado seja aprimorado, uma vez que a point cloud é um meio de visualização de dados versátil e que pode ser utilizado em diversas aplicações dentro da sociedade. Suas aplicações são inúmeras, desde a visualização de projetos arquitetônicos e estruturas geológicas até a utilização na imersão de jogos eletrônicos e, mais recentemente, impressão 3D.

Nesse contexto, o presente trabalho visa apresentar uma diferente abordagem do estado-da-arte para a compressão de cores em uma point cloud. A metodologia realizada nesse trabalho foi desenvolvida do zero e utilizou de diversas técnicas de processamento de dados e codificação para chegar no resultado obtido. O método usado consistiu em realizar cortes na point cloud, em que para cada corte é realizada uma segmentação da camada obtida em filamentos unidimensionais, comprimindo as cores desses filamentos utilizando codificação diferencial e de Huffman. O método utilizado é sem perdas e não depende da forma como a geometria é codificada, podendo ser interessante o seu uso juntamente com outros métodos de codificação de geometria para a obtenção de um resultado ótimo.

O método foi testado em 6 point clouds diferentes, todas com diferentes densidades de pixel e, no geral, diferentes estruturas geométricas e distribuições de cores. Os resultados obtidos foram comparados com um método de compressão de cores de point cloud que

hoje é considerado o estado-da-arte, o RAHT. Os deltas utilizados no RAHT para a comparação foram bem pequenos, uma vez que ele é um método com perdas, e para que se tenha uma comparação entre os dois métodos justa ele deve se aproximar o máximo possível de um método de compressão sem perdas.

Os resultados obtidos foram promissores, por mais que os arquivos comprimidos pelo RAHT com delta igual a 1 tenham sido em média, 30% menores do que o do método MTDC, para deltas entre 0.10 e 0.25 o algoritmo desenvolvido nesse trabalho conseguiu superar o RAHT, que é o estado-da-arte. É importante também levar em consideração que o método desenvolvido nesse trabalho é sem perdas e por mais que os deltas utilizados com o RAHT tenham sido baixos, ainda há perdas como podemos ver pela PSNR calculada.

5.2 Trabalhos futuros

Esta seção apresenta algumas direções possíveis para a realização de trabalhos futuros.

5.2.1 Otimizar a segmentação dos cortes

Otimizações na segmentação dos filamentos dos cortes podem aprimorar ainda mais os resultados obtidos. Quando observamos os histogramas apresentados apresentados no resultado nitidamente há espaço para melhora na obtenção desses filamentos, uma vez que na grande maioria dos cortes filamentos que a olho nu aparentam somente ser um filamento acabam sendo divididos em múltiplos filamentos vide exemplo dado na metodologia.

Outra perspectiva interessante de seria entender como alterações nos parâmetros da função que segmenta os cortes alteram os resultados, desde a segmentação dos filamentos até o limiar de distância que define o quão distante dois pontos podem estar para fazer parte de um mesmo filamento.

5.2.2 Melhoria na escrita dos bitstreams do codificador

A escrita dos bitstreams da metodologia tiveram como principal enfoque uma rápida e simples decodificação. É possível melhorar a escrita dos dados obtidos de forma a ocupar a menor quantidade de bits possível, principalmente dos dicionários em que foi utilizado o método de run length coding.

A codificação das cores codificadas diferencialmente foi feita através do método de Huffman simples, sem agrupamento e não adaptativo. Alterações na forma como o método de Huffman foi aplicado ou até uma total troca de método pode ser benéfica para o desempenho da metodologia. Os dicionários para decodificar as cores ocupam uma parte considerável do bitstream e remover eles causaria um grande ganho de desempenho.

Há espaço também para melhoria na codificação de filamentos pequenos ou que apresentam uma mesma cor durante toda a sua estrutura, hoje no estado em que o programa se encontra ele codifica e escreve todos os filamentos da mesma forma.

5.2.3 Induzir perdas

A metodologia foi pensada para um codificador sem perdas, no entanto seria interessante observar qual seria o ganho de bits caso houvesse a indução de perdas no método, uma vez que isso possibilitaria atingir taxas ainda menores.

Referências

[1] Özbay, Erdal e Ahmet Çinar: *3d reconstruction technique with kinect and point cloud computing.* 3rd WORLD CONFERENCE on INFORMATION TECHNOLOGY (WCIT 2012), 3, 2013. 1

[2] Rusu, Radu Bogdan e Steve Cousins: *3d is here: Point cloud library (pcl).* 2011 IEEE International Conference on Robotics and Automation, 2011. 1

[3] N. Namitha, V.K. Jayasree, Vaitheeswaran Sm e M.K. Bharat: *Point cloud mapping measurements using kinect rgb-d sensor and kinect fusion for visual odometryo.* Procedia Computer Science, 89(4), 2016. 1, 5

[4] Queiroz, Ricardo L. de e Philip A. Chou: *Compression of 3d point clouds using a region-adaptive hierarchical transform.* IEEE Transactions on Image Processing, 25(8), 2016. 1, 10, 17

[5] Cha Zhang, Dinei Florencio e Charles Loop: *Point cloud attribute compression with graph transform.* Microsoft Research. 1, 10

[6] Shaoping Lu, Taijiang Mu e Songhai Zhang: *A survey on multiview video synthesis and editing.* TSINGHUA SCIENCE AND TECHNOLOGY, 21(6), 2016. 1

[7] Zamarin, Marco e Søren Forchhammer: *3d video compression and transmission.* Technical University of Denmark, 2018. 1

[8] Biolchini, Jorge, P Gomes Mian, A Candida Cruz Natali e G Horta Travassos: *Preliminaries of 3d point cloud processing.* System Engineering and Computer Science Department COPPE/UFRJ, Technical Report ES, 679(05), 2005. 1, 5

[9] Gonzalez, Rafael C. e Richard E. Woods: *Digital Image Processing (3rd Edition).* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006. 3

[10] Sayood, Khalid: *Introduction to Data Compression (3rd Edition).* Morgan Kaufmann, San Francisco, CA, 2006. 6, 7, 8, 9

[11] Shannon, Claude E.: *A mathematical theory of communication.* Bell System Technical Journal, 27, 1948. 6

[12] Rufael Mekuria, Kees Blom e Pablo Cesar: *Design, implementation, and evaluation of a point cloud codec for tele-immersive video.* IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, 27(4), 2017. 10, 11

[13] Julius Kammerl, Nico Blodow, Radu Bogdan Rusu Suat Gedikli Michael Beetz e Eckehard Steinbach: *Real-time compression of point cloud streams.* IEEE International Conference on Robotics and Automation, 2012. 10

[14] Garcia, Diogo C. e Ricardo L. de Queiroz: *Context-based octree coding for point-cloud video.* 10

[15] Lin Tang, Fei peng Da e Yuan Huang: *Compression algorithm of scattered point cloud based on octree coding.* IEEE International Conference on Computer and Communications, 2, 2016. 10

[16] Yan Huang, Jingliang Peng, C. C. Jay Kuo e M. Gopi: *A generic scheme for progressive point cloud coding.* IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, 14(2), 2008. 10

[17] Chou, Philip A. e Ricardo L. de Queiroz: *Gaussian process transforms.* 10

[18] Microsoft: *Jpeg pleno database: Microsoft voxelized upper bodies - a voxelized point cloud dataset.* https://jpeg.org/plenodb/pc/microsoft/, 2018. 13, 14, 27

[19] MATLAB$_R$2017b : *(version 9.3).TheMathWorksInc.*, $Natick, Massachusetts$, 2017.13, 27

[20] MATLAB: *Computer Vision System Toolbox (R2017b).* The MathWorks Inc., Natick, Massachusetts, 2017. 13

[21] MATLAB: *Communications Toolbox (R2017b).* The MathWorks Inc., Natick, Massachusetts, 2017. 13

[22] Eugene d'Eon, Bob Harrison, Taos Myers e Philip A. Chou: *"8i voxelized full bodies, version 2 – a voxelized point cloud dataset," iso/iec jtc1/sc29 joint wg11/wg1 (mpeg/jpeg) input document m40059/m74006*, January 2017. 44

Apêndice A

Código Fonte

O código fonte desenvolvido neste trabalho pode ser consultado em:
`https://github.com/Teodim/Coding_point_cloud_colors_with_filaments`