Resumo

Point clouds, ou nuvens de pontos, são um método de representação de imagens tridimensionais cada vez mais difundido. As imagens em 3D são úteis não só na criação de ambientes imersivos com fins de entretenimento, mas também nas mais diversas aplicações industriais, acadêmicas e até mesmo forenses. *Point clouds* são adaptadas a muitas tarefas de visão computacional, tendo destaque entre elas aplicações em carros autônomos. De fato, dados representados por nuvens de pontos gerados por sensores como o *LiDAR* já são utilizados para navegação e segurança destes veículos. O volume de dados deste tipo de aplicação vem crescendo exponencialmente, e estas informações precisam ser transmitidas de maneira eficiente por canais cuja capacidade é naturalmente limitada. Isto cria um desafio: como representar *point clouds* de maneira eficiente? Este trabalho visa resolver uma parte deste problema. Aqui é proposto um esquema de codificação sem perdas do sinal de cor de imagens tridimensionais representadas em *point clouds* baseado em um trabalho anterior. O procedimento desenvolvido utiliza informações oriundas da geometria das nuvens para agrupar pontos parecidos. As sequências de pontos assim geradas apresentam redundância no sinal de cor. Esta redundância é explorada com a utilização de um codificador diferencial. O sinal gerado por esse passo é por sua vez alimentado a um codificador aritmético adaptativo. Os algoritmos desenvolvidos foram implementados em linguagem de programação Python com o auxílio de bibliotecas de código aberto.

Palavras-chave: *point clouds*, compressão de dados, codificação de cor

Abstract

Point clouds are computer representations of three-dimensional objects. They are used in creating immersive environments in virtual reality; in the quality control of manufacturing processes; and even in crime scene investigations. One growing application of this technology is the acquisition and processing of environment data for autonomous vehicles, where point cloud representations generated by LiDAR systems are useful for computational vision tasks such as route planning and collision avoidance. The increasing volume of this type of data being acquired and processed creates a problem: how can this information be represented efficiently? Inherent limitations in storage and transmission could hinder the development of novel applications. Thus, the compression of point cloud data is crucial to the spread of those new technologies. This work aims to further the efforts already made to tackle this challenge. Here we present a compression scheme for a point cloud's color attributes. The scheme consists in processing the cloud's geometry and segmenting it into groups of points we call filaments, which are arrays of voxels that are transmitted sequentially. Each filament's color signal is fed to a differential encoder, the output of which is encoded using adaptive arithmetic compression. All the processes were implemented using open-source tools and most of the work was done in Python.

Keywords: point clouds, data compression, color encoding

Sumário

Lista de Figuras

Lista de Tabelas

o tarefa em duas partes: compressão de geometria e compressão do sinal dos atributos (por exemplo, cor). Este trabalho visa a implementação de um codificador para o sinal de cor de uma *point cloud*. O princípio dos esquemas propostos é utilizar as propriedades da geometria da nuvem para extrair a redundância das informações de cor de pontos próximos. Um trabalho anterior [13] propôs segmentar as nuvens de pontos em sequências de pontos próximos, que foram chamados de filamentos. Os filamentos foram codificados diferencialmente e o resultado desta etapa foi comprimido usando código de Huffman. O presente trabalho pretende melhorar o algoritmo de processamento de geometria apresentado no trabalho anterior, e melhorar as taxas obtidas empregando um codificador aritmético adaptativo ao invés do codificador de Huffman. Os algoritmos foram implementados em linguagem Python 3, com bibliotecas de código aberto, e foram testadas com imagens de uma base de dados disponível para fins acadêmicos na internet.[14] As imagens escolhidas para os testes podem ser vistas na figura 1.2

Organização do trabalho

O trabalho está organizado da seguinte maneira:

- No capítulo 2 são apresentados os princípios da representação de imagens tridimensionais em nuvens de pontos e do processamento de sua geometria. São detalhadas as noções necessárias de Teoria da Informação: é apresentado o conceito de entropia e discorre-se sobre os vários tipos de técnicas de compressão.
- O capítulo 3 trata das técnicas propostas para a resolução do problema. São analisados o custo computacional e a performance dos diferentes algoritmos de processamento de geometria estudados. Nesta seção também é são descritas as diferentes partes que integram o algoritmo de compressão: codificação diferencial e codificador aritmético adaptativo.
- No capítulo 4 são comentados os resultados do algoritmo para algumas nuvens de pontos. As taxas de compressão dos diferentes algoritmos são comparadas entre si, e os melhores resultados são comparados com os de trabalhos anteriores.
- Finalmente, o capítulo 5 analisa as vantagens e as limitações dos algoritmos propostos e aponta para possíveis futuros trabalhos.

2 Teoria e revisão bibliográfica

Este capítulo explica os conceitos básicos sobre nuvens de pontos, algoritmos de processamento de gemoetria e algoritmos de compressão de dados.

2.1 Nuvens de pontos

O processamento digital de imagens trata primeiramente de imagens bidimensionais. Em [15] uma imagem é definida como uma função que associa um ponto de uma parte do plano, denominado *pixel* a um valor de intensidade luminosa. Esta é a definição de uma imagem em preto-e-branco. Associando cada ponto a um valor de intensidade para cada uma das três cores primárias, é possível obter uma imagem colorida. Uma imagem tridimensional pode ser entendida como uma função que associa a ponto de um espaço de um dado conjunto tridimensional a um valor de intensidade luminosa, ou então a valores de cor. Uma diferença importante é o fato de que, enquanto imagens planas têm tipicamente como suporte uma região contínua e retangular do plano, o domínio de imagens tridimensionais é muitas vezes apenas um subconjunto pequeno e possivelmente descontínuo do espaço tridimensional.

2.1.1 Descrição teórica

Nuvens de pontos (em inglês *point clouds*) são uma maneira de representar objetos como um conjunto de pontos em um espaço tridimensional. É uma técnica de representação mais simples do que representações como redes poligonais (*polygon meshes*), já que não são transmitidas informações sobre as relações entre os pontos (*e.g.* arestas ou faces). Isto pode ser vantajoso para algumas aplicações já que a renderização é simplificada e a quantidade de informação transmitida é menor. Além da informação de posição, cada ponto da nuvem pode ter outros atributos, dependendo da aplicação, como cor, refletância, ou coordenadas do vetor normal à superfície.

É possível criar vídeos tridimensionais com sequências onde cada um dos quadros é uma *point cloud*. As nuvens de pontos estudadas neste trabalho são quadros de vídeos

2.2 Informação, entropia e compressão de dados

Como notam Thomas e Cover [20],"o conceito de informação é muito amplo para ser capturado em uma única definição". No entanto, informação pode ser entendida como tudo aquilo que se deseja transmitir, como textos, sinais de voz, imagens. Sayood [21] descreve a compressão de dados como "a arte ou a ciência de representar dados de maneira compacta". As técnicas de compressão modelam estruturas ou identificam padrões na informação, com objetivo de diminuir a redundância e permitir uma transmissão mais eficiente.

2.2.1 Entropia

Em teoria da informação, os dados a serem transmitidos são interpretados como uma sequência de símbolos emitidos por uma fonte. O comportamento da fonte é geralmente modelado como uma variável aleatória, cuja distribuição de probabilidade relaciona cada símbolo com a probabilidade de sua ocorrência. O conjunto de símbolos da fonte, que corresponde ao suporte da variável aleatória, é denominado alfabeto (que pode ser finito ou infinito). Já a taxa média de informação associada a uma variável aleatória pode ser medida com o conceito de entropia, que foi definido pela primeira vez no artigo seminal de Shannon [22].

Definição 1 *A entropia $H(X)$ de uma variável aleatória X é dada por*

$$H(X) = -\sum_{x \in X} p(x) \log_b p(x)$$

Esta medida é definida para uma base b. É usual tomar a base como 2, fazendo com que a entropia seja medida em *bits*. A entropia é o limite de compressão de um sinal. Isso significa que não é possível comprimir um sinal abaixo de sua entropia sem perder informação ou acrescentar informação oriunda de outra fonte.

2.2.2 Técnicas de compressão e qualidade da reconstituição

As técnicas de compressão são métodos de representar informação sem redundância. A compressão tem dois componentes complementares: a codificação, que diminui o tamanho da informação, e a decodificação, que reconstitui os dados. Se os dados reconstituídos são idênticos aos dados originais, a compressão é dita sem perdas (*lossless compression*). Caso a reconstituição tenha perdas (*lossy compression*), é interessante ter uma medida da qualidade da reconstrução, ou seja, da distorção do sinal reconstituído. Uma medida que pode ser usada é o erro médio quadrático (MSE - *mean squared error*).

Definição 2 *O erro quadrático médio para uma reconstituição $\hat{X}$ de N valores de um sinal X é definido por*

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(X_i - \hat{X}_i\right)^2$$

Outra medida é a razão sinal-ruído de pico (PSNR - *peak signal-to-noise ratio*, medida em decibéis.

Definição 3 *A razão sinal ruído de uma reconstituição de um sinal cujos valores máximo e mínimo são X_{max} e X_{min} é dada por*

$$PSNR = 10 \cdot \log_{10}\left(\frac{(X_{max} - X_{min})^2}{MSE}\right)$$

2.2.3 Codificação de entropia

As técnicas de codificação de entropia são técnicas de compressão sem perdas que utilizam apenas as propriedades estatísticas da fonte, ou seja, as informações sobre a probabilidade de emissão dos diferentes símbolos. A codificação de entropia difere de outros esquemas de compressão que se adaptam à natureza semântica do conteúdo, explorando propriedades específicas de certos tipo de informação, tais como propriedades inerentes a sinais de vídeo, áudio, etc. Neste trabalho foi utilizado um tipo de codificação de entropia chamada a codificação aritmética, que será descrita mais a frente.

2.2.4 Modelagem de fonte

As técnicas de compressão são tipicamente compostas por uma sequência de vários estágios. [23] A primeira parte do processo de compressão consiste em extrair padrões existentes nos dados a serem transmitidos. Esta parte do processo assume certas propriedades sobre o sinal a ser comprimido e, por isso, deve ser adaptada à natureza do sinal. Caso esta etapa induza perdas, deve-se adaptar o algoritmo às necessidades do usuário do produto final do processo de compressão. Tome-se, por exemplo, um sinal de vídeo. Pixels de uma dada região de um quadro tendem a ser parecidos entre si: esta propriedade é denominada redundância espacial. Além disso, eles também tendem a ser parecidos com os pixels da mesma região de quadros anteriores (redundância temporal). É possível explorar essa redundância na compressão de imagens e vídeos[15]. Ainda no exemplo do sinal de vídeo, é importante levar em consideração as características do espectador: por exemplo, a visão humana é mais sensível a variações na luminância do que variações na cor. Sendo assim, caso perdas sejam induzidas neste exemplo, é melhor que o erro de reconstituição seja no sinal de cor e não no sinal de luminância.

A informação gerada por esse processo é modelada estatisticamente, gerando um fluxo de símbolos que alimenta um codificador de entropia. Neste trabalho, a redundância do sinal de cor foi explorada usando o processamento de geometria, que gera um sinal que é codificado utilizando codificação diferencial, cujo resultado é passado por um codificador de entropia. O codificador de entropia escolhido foi o codificador aritmético. Essas três técnicas são explicadas nas seções que seguem.

2.3 Processamento de geometria

No contexto da codificação de nuvens de pontos, o processamento de geometria consiste em utilizar a informação das coordenadas dos *voxels* ocupados para agrupar pontos próximos entre si. Esta técnica visa explorar o fato anteriormente citado de que *voxels* próximos têm cores parecidas. Assim, é possível transmitir a cor de *voxels* parecidos de maneira sequencial. Isso cria uma redundância no sinal, a qual pode ser aproveitada pela codificação diferencial, que está descrita na próxima seção. A sequências geradas por esta etapa foram denominadas "filamentos".

2.3.1 Encontrando os vizinhos mais próximos

Será visto no capítulo de metodologia que para gerar os filamentos, um dos passos possíveis é encontrar os n vizinho mais próximos (*kNN*, ou k *nearest neighbours*). Um algoritmo de força bruta para encontrar os k vizinhos mais próximos tem complexidade espacial $\mathcal{O}(n)$. Aplicando-o para todos os pontos da nuvem, obtemos uma complexidade $\mathcal{O}(n^2)$. No entanto, é possível realizar um pré-processamento dos pontos, gerando uma estrutura denominada árvore K-D, que particiona o plano em regiões contendo quantidades iguais de pontos. Usando essa árvore a procura pelos k vizinhos mais próximos tem complexidade $\mathcal{O}(\log n)$. Esta estrutura foi usada para encontrar rapidamente os pontos vizinhos na geração de filamentos, como será descrito na metodologia. O algoritmo da biblioteca FLANN [24] é uma heurística que permite fazer isso de forma mais rápida, mas não garante uma seleção exata [25].

2.3.2 Curvas preenchedoras de espaço

As curvas preenchedoras de espaço (*Space-filling curves*, ou *SFC*s) são curvas fractais que preenchem um hipercubo unitário [26]. Elas podem ser interpretadas como uma função sobrejetiva e contínua que leva do intervalo unitário $[0, 1]$ a um hipercubo unitário $[0, 1]^n$. A sobrejetividade de tais funções implica que o quadrado unitário é completamente

recoberto pela curva. Além disso, como essas funções são contínuas, elas preservam a localidade. Isso significa que se dois pontos estão próximos no domínio, suas respectivas imagens estão próximas no espaço $\mathbf{R}^n$. O inverso não é sempre verdadeiro: dois pontos próximos no espaço $\mathbf{R}^n$ podem ter suas imagens afastadas no domínio $\mathbf{R}$. Estas curvas são o limite de uma sequência de aproximações discretas H_j que associam um conjunto de racionais $\{\frac{1}{N^n}, \frac{1}{N^n}...1\}$ ao conjunto de pontos do espaço $[0,1]^k$. Esta família de curvas começou a ser estudada no final do século XIX e sua teoria foi desenvolvida por matemáticos como Peano, Hilbert e Sierpiński.

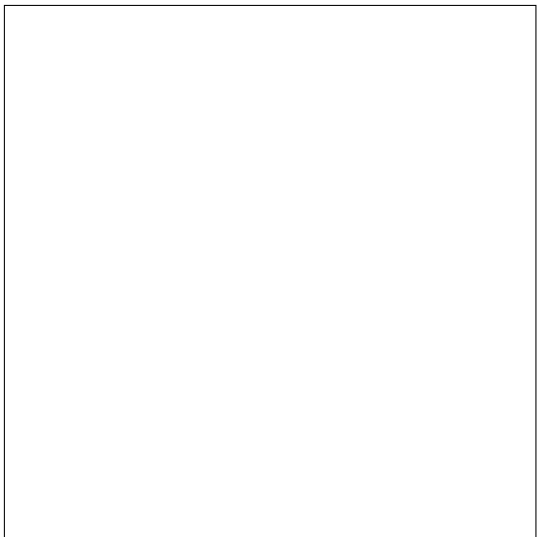
As iterações das curvas preenchedoras de espaço podem ser obtidas usando sistemas de reescrita chamados sistemas de Lindenmayer[27]. Sistemas de Lindenmayer são um tripla (V, ω, P) em que V é um alfabeto finito, ω é um axioma e P é o conjunto de regras de reescrita. Cada iteração do sistema gera uma nova cadeia de caracteres seguindo as regras de reescrita contidas em P. A primeira iteração é chamada *axioma* e é dada por ω. A cadeia de caracteres obtida a cada iteração pode ser interpretada geometricamente como uma sequência de instruções de desenho onde cada símbolo pode corresponder a um comando de gráficos-tartaruga (*turtle graphics*).

Um exemplo de SFC é a curva de Hilbert, cujas primeiras iterações podem ser vistas na figura 2.4. Ela pode ser gerada pelo seguinte sistema de Lindenmayer:

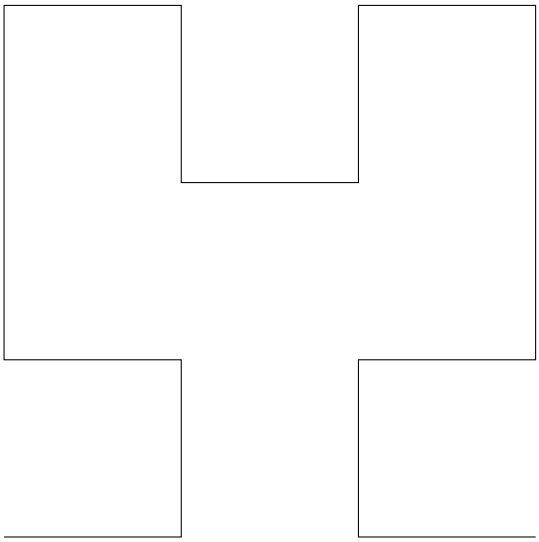
$$
\begin{aligned}
&V = \{L, R, F, +, -\} \\
&\omega = L \\
&P \begin{cases} p_1 : L \rightarrow +RF - LFL - FR+ \\ p_2 : R \rightarrow -LF + RFL - FR+ \end{cases}
\end{aligned}
$$

Com a seguinte interpretação:

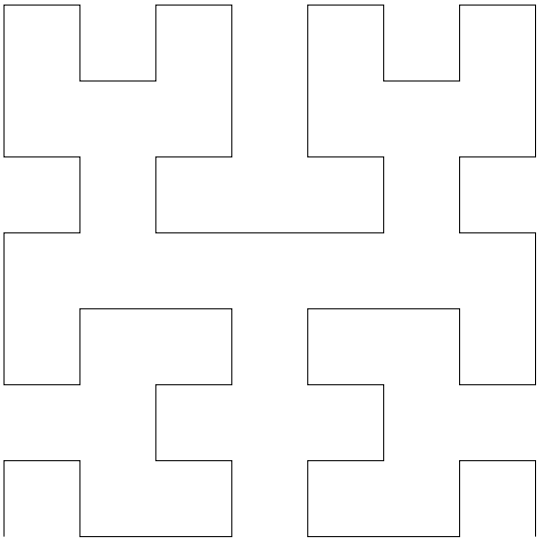
- $F \rightarrow$ Dar um passo a frente
- $- \rightarrow$ Virar 90º para a direita.
- $+ \rightarrow$ Virar 90º para a esquerda.
- L e R são ignorados.



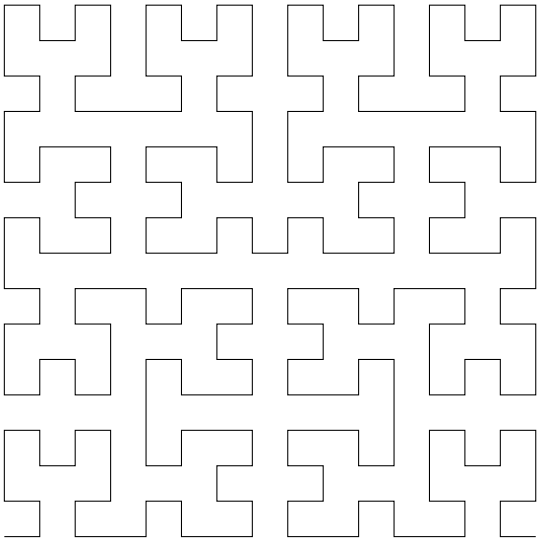
(a) ordem 1



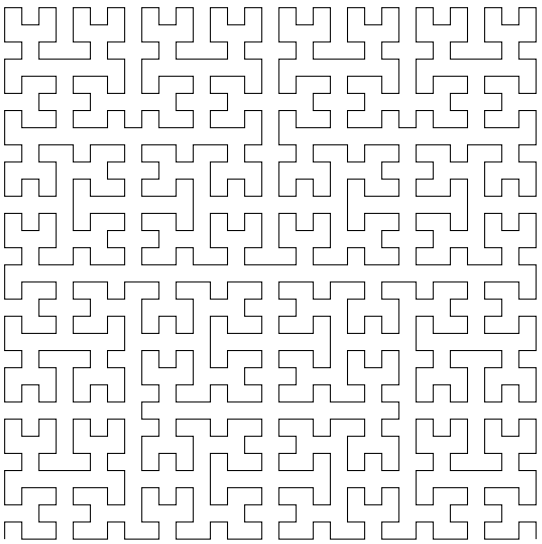
(b) ordem 2



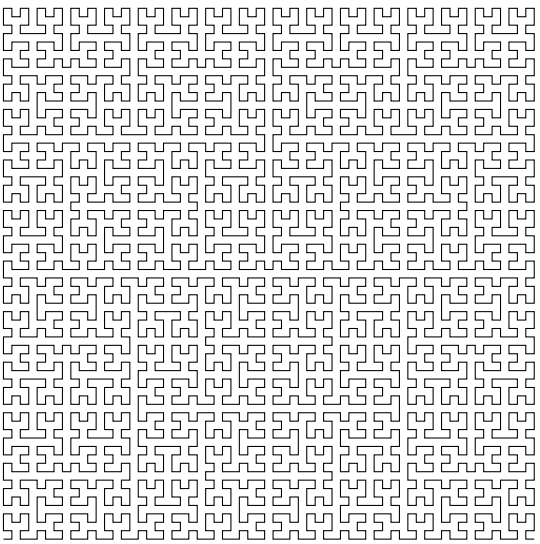
(c) ordem 3



(d) ordem 4



(e) ordem 5



(f) ordem 6

Figura 2.4: As primeiras iterações da curva de Hilbert. (geradas a partir de [28])

2.4 Codificação diferencial

A codificação diferencial aproveita a correlação entre dados consecutivos para comprimir a informação a ser transmitida. Ela consiste em usar os dados anteriores para estimar o próximo valor a ser transmitido. Este tipo de codificação é útil para codificar sinais que mudam devagar (sinais "passa-baixa"). Ao invés de transmitir o sinal, é transmitido o erro de estimação. Tipicamente este erro tem propriedades estatísticas que facilitam sua compressão por um codificador de entropia: ele tende a ter uma extensão menor e também uma variância menor.

2.4.1 Gerando uma estimativa

O método mais simples de estimar o próximo valor é simplesmente usar o último símbolo enviado como estimativa. É o que foi feito no trabalho anterior [13] e também no presente trabalho. No entanto, é possível criar estimativa usando funções mais complexas, que dependem de uma quantidade maior de símbolos anteriores. Um tipo de estatística bem estudado é uma função linear (média ponderada) de uma quantidade fixa de pontos anteriores. Os coeficientes ótimos podem ser calculados, como descrito no capítulo 11 do livro [21].

2.5 Codificador aritmético

A codificação aritmética é uma técnica de codificação de entropia. A sua taxa de compressão aproxima-se assintoticamente da entropia para fontes independente e identicamente distribuídas (iid). Ela consiste em um algoritmo que associa de maneira única uma mensagem a um subintervalo do intervalo $[0, 1)$. É escolhido um número, denominado *tag*, dentro do intervalo obtido. É enviada a *tag* acompanhada da quantidade de símbolos codificados. Nas subseções seguintes são descritos os princípios gerais do código aritmético e de sua implementação com precisão finita, denominada implementação inteira. Esta seção se baseia na explicação de [21], capítulo 5.

2.5.1 Descrição dos princípios do código aritmético

Seja $(a_1, a_2, ..., a_n)$ um arranjo do alfabeto n-ário de uma fonte de símbolos. A função de probabilidade acumulada associada a esse arranjo é dada por

$$F(k) = \sum_{i=0}^{k} p(a_i)$$

De maneira geral, pode-se associar a um intervalo $I_i = [l_i, u_i)$ uma partição

$$P_{I_i}(k) = \frac{l_i + F(k-1)}{u_i - l_i}, \frac{l_i + F(k)}{l_i - u_i}$$

Para transmitir uma mensagem $(m_1, m_2, ..., m_N)$ parte-se do intervalo $I_0 = [0, 1)$ e obtém-se intervalos sucessivamente menores usando a fórmula

$$I_n = P_{I_{n-1}}(m_n)$$

. Finalmente, escolhe-se um número contido neste último intervalo dessa sequência. Este número é a *tag* associada a mensagem.

É suficiente transmitir ao codificador a *tag*, a quantidade N de símbolos transmitidos, e a função de probabilidade acumulada. Para decodificar, parte-se do número recebido t_0 e procede-se da seguinte maneira; denominando $D_I(t)$ a função que associa t ao intervalo da partição P_I que o contém, basta calcular sucessivamente, para $< n \leq N$:

$$\hat{m}_n = D_{I_n}(t_n)$$

$$t_n = \frac{l + F(\hat{m}_n)}{u - l}$$

$$I_n = P_{I_{n-1}}(m_n)$$

.

2.5.2 Implementação com precisão finita

Muitas plataformas não podem trabalhar com números precisão arbitrária. Por isso, é interessante estudar a implementação com precisão finita, que foi empregada no presente trabalho. Ao invés de trabalhar com o subintervalos do intervalo unitário $[0, 1)$, são usados subintervalos do intervalo $[0, M)$ onde M é o máximo inteiro que pode ser representado com a dada precisão. Os cálculos para codificação e decodificação são adaptados à aritmética inteira. Uma desvantagem do uso de precisão finita é a limitação da resolução da função de probabilidade acumulada.

2.6 Estado da arte em compressão de nuvens de pontos

É usual na compressão de nuvens de pontos separar a informação correspondente à geometria do objeto (coordenadas dos pontos) daquela que representa o resto dos atributos

(por exemplo o valor de cor de cada ponto). Vários trabalhos já apresentaram esquemas de compressão do canal de cor de *point clouds* com base na geometria. De forma geral, estes esquemas partem do princípio que pontos espacialmente próximos tendem a ter cores parecidas. Assim, os algoritmos apresentados organizam os pontos de forma que pontos parecidos sejam transmitidos sequencialmente, gerando por assim dizer um sinal passa-baixa. É importante salientar que nestes esquemas a transmissão da geometria do objeto não pode ter perdas. Já existem algoritmos para comprimir a geometria de vídeos de *point clouds* [29].

Já o sinal de cor pode ser transmitido com ou sem perdas, dependendo do algoritmo. O estado da arte é o *codec* G-PCC (Geometry based Point Cloud Compression) desenvolvido pelo ISO/IEC MPEG.implementado no software TMC13 [30]. Ele é baseado no algoritmo da *Region-Adaptive Hierarchical Transform* (RAHT - Transformada Hierárquica Adaptada à Região), descrito em [31]. No artigo, os pontos são agrupados por proximidade em uma estrutura de árvore. Os trabalhos [13] e [32] dividem a imagem em sequências de pontos próximos denominadas filamentos. [32] utiliza a transformada *wavelet* para cada codificar cada filamento. [13] utiliza um esquema simples de codificação diferencial, e este esquema foi o ponto de partida para o presente trabalho. No capítulo de Resultados o algoritmo proposto no presente trabalho será comparado com os resultados obtidos com o algoritmo orignal em [13] e com o G-PCC/TMC13.

3 Metodologia

Este capítulo apresenta a metodologia aplicada para implementar um algortimo de compressão sem perdas do canal de cor de nuvens de pontos

3.1 Visão geral

O algoritmo proposto é composto pelos seguintes passos, como ilustrado na figura 3.1:

1. Processamento de geometria
   (a) Separação da nuvem em camadas (conjuntos com a mesma coordenada z).
   (b) Geração de um filamento para cada camada.
   (c) Corte dos filamentos de acordo com um limiar de distância.
2. Codificação
   (a) Escolha do método de codificação para cada filamento:
      - Para filamentos longos: codificação diferencial seguida de codificação aritmética.
      - Para filamentos curtos: transmissão pelo canal de informação lateral (sem compressão).

3.2 Processamento da geometria e codificação diferencial

O processamento da geometria consiste em obter uma lista de filamentos a partir da imagem fornecida ao codificador. Inicialmente a imagem é dividida em camadas de pontos que possuem a mesma coordenada z (a escolha da direção z é arbitrária). Alguns exemplos de camadas obtidas em diferentes alturas de uma mesma *point cloud* podem ser vistos

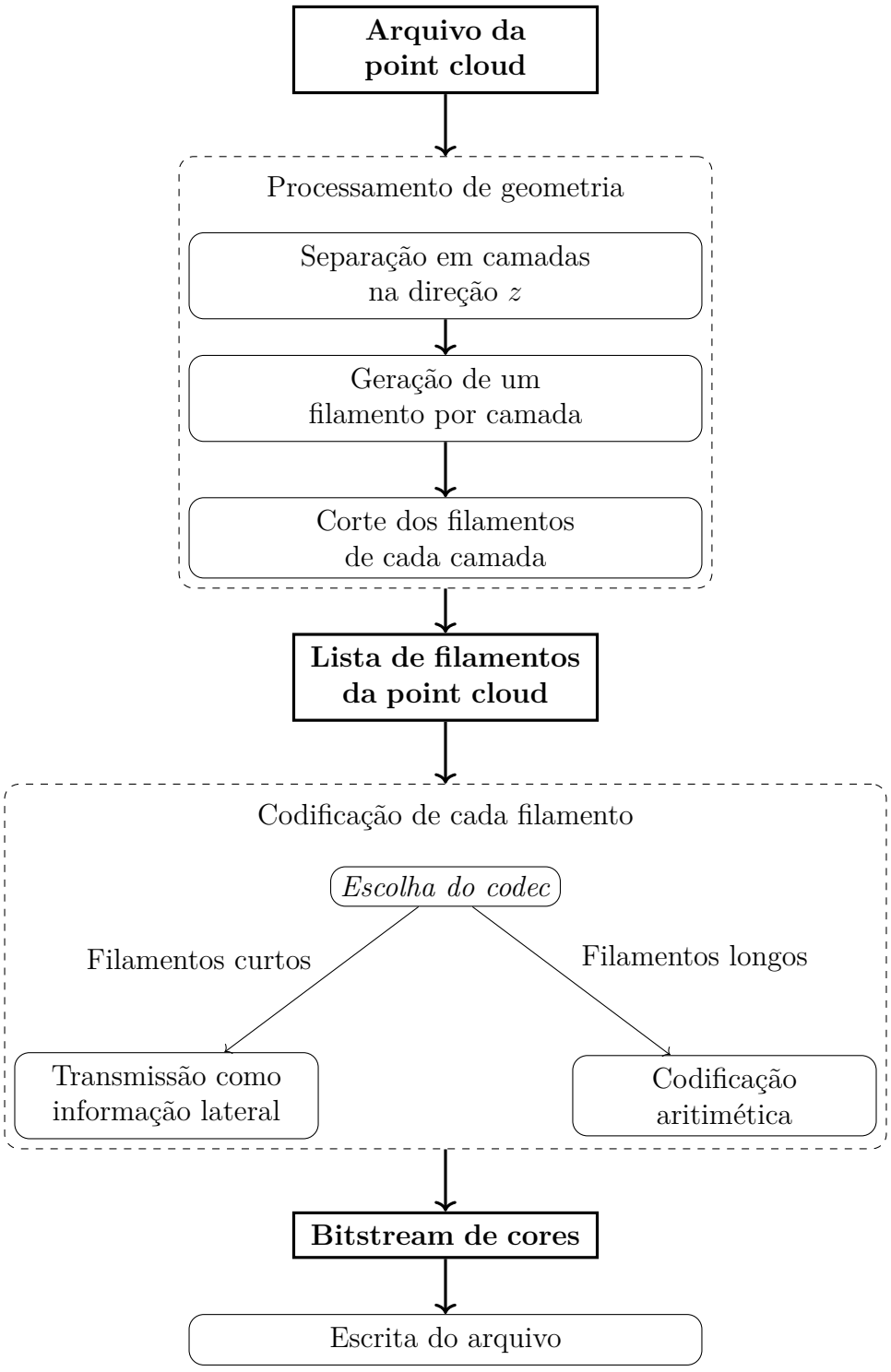


Figura 3.1: Organização do algoritmo proposto

na figura 3.2. Em cada camada, escolhe-se um ponto arbitrariamente para ser o primeiro ponto do filamento. A cada iteração, a partir do último ponto incluído no filamento escolhe-se um ponto próximo que ainda não foi visitado, usando um dos três algoritmos descritos na próxima seção. O ponto escolhido é acrescentado ao filamento. Repete-se esse processo até não sobrarem pontos na camada, sempre guardando a distância em um vetor auxiliar. Assim é obtida uma lista de filamentos, sendo um filamento por camada. Em seguida, este filamento é cortado quando as distâncias estão acima de um certo limiar. Este limar é um parâmetro de entrada do codificador a ser otimizado. Assim é obtida uma lista de filamentos para cada camada.

Para obter a estimativa da codificação diferencial, foi utilizado apenas o último *voxel* a ser transmitido.

3.2.1 Algoritmo de escolha de vizinho

O passo crucial da geração de filamentos é a escolha do próximo ponto dentre todos os vizinhos que não foram ainda visitados. Foram testados três algoritmos para escolha do próximo ponto a ser visitado:

- Algoritmo "exato": Calcula as distâncias até todos os pontos não visitados, e escolhe o mais próximo.
- Algoritmo "FLANN": utiliza uma função heurística da biblioteca Open3D para escolher um ponto suficientemente próximo.
- Algoritmo "Hilbert": percorre os *voxels* ocupados na ordem de travessia da curva de Hilbert.

Algoritmo "Exato"

A cada iteração, este algoritmo calcula exatamente as distâncias até todos os pontos que ainda não foram visitados e simplesmente escolhe o mais próximo. Para uma camada com n pontos, cada iteração tem complexidade $O(n)$, e são necessárias $O(n)$ iterações para terminar o processo. Portanto, o algoritmo tem complexidade $O(n^2)$. Espera-se que ele seja o mais lento de todos.

Algoritmo "FLANN"

Este algoritmo utiliza a classe *KDTreeFlann* da biblioteca *Open3D*. Esta classe implementa a representação de um conjunto de pontos como uma árvore $k-d$, e disponibiliza

algoritmos de pesquisa de pontos próximos usando o método FLANN, descrito originalmente em [24]. Inicialmente é inicializada uma instância da classe com todos os pontos da camada. Escolhe-se (arbitrariamente) um ponto para começar o filamento, e o método de pesquisa é usado a cada rodada para gerar uma lista de pontos próximos, e escolhe-se o menor dentre eles para visitar. Por ser um heurística, espera-se que este método seja mais rápido que o método "exato".

Algoritmo "Hilbert"

Este algoritmo visita os pontos da camada na ordem da curva de Hilbert. A curva de Hilbert é gerada previamente e guardada em uma tabela. É esperado que este algoritmo seja muito mais rápido que os dois outros, por empregar uma simples função de *look up*.

3.2.2 Limiar de corte e codificação diferencial

A codificação diferencial aproveita a redundância do sinal de cor de pontos próximos. A cor de pontos muito distantes tende a ter menos correlação; por isso, é interessante cortar os filamentos obtidos com os algoritmos descritos na seção anterior nos pontos em que a distância entre dois pontos consecutivos é muito grande. Para fazer isso, é necessário definir um limiar de corte, ou seja, uma distância máxima entre dois pontos consecutivos de um filamento. Este limiar é um parâmetro de entrada do codificador desenvolvido.

Um exemplo ilustrando os algoritmos de geometria

Nas figuras 3.3 e 3.4 podemos ver partes de uma camada de uma nuvem de pontos em diferentes estágios de processamento. Em ambas figuras, as subfiguras (b), (c) e (d) mostram o filamento gerado pelos diferentes algoritmos de pesquisa de vizinhos, e as subfiguras (e), (f) e (g) mostram os filamentos após o corte com limiar de distância de corte igual a 8. Podemos ver que, pelo menos no caso das regiões mostradas na figura, o método exato tende a gerar filamentos melhores, isto é, compridos e com menos saltos grandes. Podemos ver na subfigura 3.3(c) que com método "FLANN" foi gerado um filamento com apenas dois pontos, e outro com apenas um ponto. Este tipo de resultado é especialmente ruim para a codificação posterior porque não relaciona muitos pontos, limitando a quantidade de informação que pode ser aproveitada na compressão.

3.3 Escolha do método codificação

A codificação aritmética de sequências muito pequenas tende a ser, na prática, ineficiente porque a informação lateral que deve ser enviada é proporcionalmente grande. Então, para filamentos muito curtos, o codificador aritmético não é a melhor opção para codificação. A escolha feita foi não codificar os filamentos muito curtos, que têm os seus valores transmitidos diretamente pelo canal de informação lateral. O limiar de comprimento a partir do qual o filamento é codificado é um parâmetro de entrada a ser otimizado.

3.4 Modelagem da fonte e codificação de entropia

Como visto anteriormente, quanto melhor o modelo que se tem da fonte, melhores são os resultados do codificador aritmético. Vários esquemas de transmissão do contexto são possíveis. É possível enviar uma descrição completa da distribuição de probabilidade, mas isso ocupa bastante espaço e tende a ser ineficiente porque os filamentos são proporcionalmente curtos.

A escolha feita foi utilizar um codificador aritmético adaptativo, com uma distribuição de probabilidade inicial uniforme. Mesmo com uma distribuição inicial errada, o codificador adaptativo aprende a seguir a variável com o tempo, aproximando-se da entropia. Além disso, o contexto do codificador aritmético não foi reiniciado entre os filamentos. Presumindo-se que o sinal do erro de estimativa deve ter distribuições de probabilidade parecidas para todos os filamentos, é razoável reaproveitar a função distribuição de probabilidades de um filamento para o próximo. Partindo do princípio que as distribuições dos símbolos oriundos da codificação diferencial são parecidas para filamentos diferentes, é possível usar um esquema que reaproveita o contexto para todos os filamentos.

3.5 Implementação e testes

Os algoritmos foram implementados usando a linguagem de programação Python (versão 3.7) [33]. Foram usadas as seguintes bibliotecas:

NumPy Para funções numéricas, especialmente para vetores e matrizes.[34]

Open3D Para processamento de nuvens de pontos (leitura de arquivos PLY, algoritmos de busca e renderização). [35]

bitstring Para a escrita e leitura *bit*-a-*bit* dos arquivos.[36]

Matplotlib para gerar os gráficos presentes deste relatório. [37]

Foram desenvolvidos módulos para processar a geometria, codificar diferencialmente e implementar o código aritmético. Todos os testes foram feitos em um notebook com processador Intel Core i5 2.3 GHz, 8 GB de RAM e placa de vídeo Intel Iris Plus Graphics 640 1536 MB, usando o sistema operacional *macOS* versão 10.13.6.

3.6 Procedimentos experimentais

Foram realizados vários tipos de experimentos para otimizar os parâmetros de compressão. Os testes foram feitas sobre alguns quadros da base de dados *Microsoft Voxelized Upper Bodies* [14]. A base de dados é constituída por vídeos tridimensionais representando humanos da cintura para cima. Cada quadro está salvo em um arquivo PLY separado. Em todas as nuvens estudadas, cada *voxel* tem 9 bits de para cada coordenada geométrica, ou seja, cada uma delas está contida em um cubo de dimensão 512. Para cada voxel, cada canal de cor comporta 8 bits de informação. Assim os arquivos possuem 27 bpov de informação de geometria e 24 bpov de informação de cor. Os quadros escolhidos estão descritos na tabela 3.1. A figura 1.2 mostra vistas frontais dos quadros.

As imagens selecionadas estão entre as que foram escolhidas por [13]. Elas apresentam propriedades que são úteis para testar comparar os algoritmos de compressão. A *point cloud* 1 não tem artefatos devidos ao movimento e sua cor é relativamente uniforme. Por isso, espera-se que ela tenha os melhores resultados de codificação. As *point clouds* 2, 3 e 4 apresentam pessoas com os braços estendidos, fazendo com que os cortes em camadas tenham aglomerações desconexas, o que pode ser um desafio para o algoritmo de geometria. Além disso, elas apresentam sombras compostas por pontos registrados erroneamente devido ao rápido movimento das pessoas no vídeo. Estes artefatos, localizados principalmente em volta das mãos, também podem causar problemas. Finalmente, as *point clouds* 3 e 4 têm componentes de alta frequência no sinal de cor na região da camisa das pessoas representadas. A camisa da *point cloud* 2 tem listras verticais, e a camisa da *point cloud* 3 é quadriculada. Espera-se que isso dificulte a estimação feita pelo codificador diferencial, gerando um sinal de erro de estimativa com maior variância, piorando portanto as taxas de codificação.

3.6.1 Parâmetros variados

Foi estudada a variação dos comprimentos dos filamentos em função do limiar de distância, para diferentes métodos de geração de filamentos.
Foi variada a quantidade de pontos utilizada na estimativa.
Para determinar o limiar de codificação ideal, foi medida a taxa de compressão em função

Nome	**Vídeo**	**Número do quadro**	**Voxels ocupados**	**Propriedades interessantes**
Pointcloud 1	Ricardo9	0000	214656	Uniforme e sem artefatos
Pointcloud 2	Ricardo9	0091	337803	Cor uniforme, muitos artefatos
Pointcloud 3	Phil9	0116	385467	Cor variada, muitos artefatos
Pointcloud 4	Andrew9	0263	277583	Cor muito variada
Pointcloud 5	Sarah9	0146	304597	Cor uniforme, artefatos

Tabela 3.1: *Point clouds* escolhidas para o trabalho

do comprimento dos filamentos, comparando com a transmissão sem codificação. Foram comparados os resultados do codificador proposto com os resultados do codificador proposto por [13].

4 Resultados

Este capítulo compara os tempos de execução dos algoritmos de busca de vizinho, analisa o efeito da variação do limiar de distância sobre o tamanho dos filamentos gerados, e apresenta as taxas obtidas para vários limiares de transmissão.

4.1 Escolha do algoritmo de segmentação e do limiar de distância de corte

Nesta seção são comparados os tempos de execução dos diferentes algoritmos de processamento de geometria, e é analisada a performance deles na geração de filamentos em função da distância de corte escolhida.

4.1.1 Análise do tempo de execução dos algoritmos de segmentação

Podemos ver na tabela 4.1 o tempo de execução dos três algoritmos de segmentação para as cinco nuvens de pontos estudadas. Como esperado, nota-se que para todos as *point clouds*, o algoritmo "Exato" é o mais lento, o algoritmo "FLANN" é bem mais rápido e o algoritmo "Hilbert" é quase três ordens de grandeza mais rápido comparado aos anteriores. Além disso, nota-se que, para os algoritmos "exato" e "FLANN" o arquivo com o menor número de pontos (*point cloud* 1) foi o mais rápido a ser processado. No entanto, a nuvem com a maior quantidade de pontos (*point cloud*) foi processada mais rapidamente comparado às *point clouds* 2 e 5, apesar destas terem menos pontos. Isso leva a crer que não só o número total de pontos, mas a quantidade de pontos em cada camada (em outras palavras, a distribuição deles em relação ao eixo z) influencia o tempo de execução do algoritmo.

	Exato	**FLANN**	**Hilbert**
Point cloud 1	47.34	32.37	0.13
Point cloud 2	106.51	80.29	0.21
Point cloud 3	72.00	48.66	0.20
Point cloud 4	63.61	43.28	0.18
Point cloud 5	112.85	78.88	0.24

Tabela 4.1: Tempo de execução dos algoritmos em segundos

4.1.2 Parâmetro: limiar de distância de corte

Nos gráficos da figura 4.1 podemos ver a variação do comprimento médio dos filamentos gerados por diferentes métodos segmentação em função do limiar de distância de corte. Podemos ver que, como esperado, para todos os arquivos, o comprimento médio tende a aumentar com o limiar do corte. Todos os arquivos estudados apresentam comportamento muito similar. Comparando os algoritmos entre si, o "exato" gerou os filamentos mais longos, especialmente para limiares de corte elevados. Surpreendentemente, o algoritmo "Hilbert" teve resultados parecidos com os do algoritmo "FLANN". Para limiares de corte maiores (> 10), o algoritmo "Hilbert" gerou filamentos mais compridos em comparação com os do algoritmo "FLANN". Isso pode ser explicado pela tendência deste último de "esquecer" alguns pontos em áreas densas. Estes pontos "esquecidos" depois são processados e integram filamentos curtos, inclusive filamentos composto por apenas um ponto, como pode ser observado na figura 4.2. O algoritmo "exato" também apresenta este problema, mas em escala bem menor. Um grande problema do algoritmo "Hilbert" são as regiões de "fratura", que são contínuas no espaço bidimensional, mas discontínuas na curva. Assim, se uma região de fratura é muito densa em pontos, estes provavelmente não serão percorridos de maneira ótima por este algoritmo. Alguns exemplos de fraturas podem ser vistos na figura 4.3.

4.2 Análise do algoritmo de codificação

O objetivo desta seção é mostrar os resultados da codificação para cada uma das nuvens de pontos. Foram variados três parâmetros:

- algoritmo de geração de filamento
- limiar de corte de filamento (distância máxima entre dois pontos consecutivos do mesmo filamento)
- limiar de codificação (tamanho mínimo do filamento para que ele passe pelo codificador aritmético, ao invés de ser enviado pelo canal de informação lateral)

4.2.1 Otimizando a taxa em função do limiar de transmissão, para limiar de corte fixo

A figura 4.4 mostra um comportamento típico da taxa em função do limiar de transmissão para um limiar de corte fixo. Na subfigura (a), cada ponto do gráfico corresponde a um filamento da *point cloud*. A linha vermelha indica a taxa de 24 *bpov*, que é a taxa dos dados brutos. Podemos ver que a maior parte dos filamentos muito pequenos está acima da linha, o que significa que é melhor transmiti-los sem compressão. Na subfigura (b), podemos ver a taxa de toda a nuvem de pontos em função do limiar de transmissão escolhido. Para limiares de transmissão muito elevados, a taxa se aproxima assintoticamente da taxa dos dados brutos, já que a maioria dos filamentos passa a ser transmitida sem compressão. Para otimizar o limiar de transmissão, precisamos encontrar o comprimento de filamento a partir do qual a maioria dos filamentos tem uma taxa melhor que a dos dados brutos. Neste caso específico, podemos ver (no destaque da subfigura (b)) que existe um mínimo para a taxa quando o limiar de transmissão é 6. De forma geral, para cada uma das imagens, dados um método de geração de filamentos e um limiar de corte, existe um limiar de transmissão ótimo.

Os gráficos das figuras 4.5 a 4.9 mostram os limiares de transmissão ótimos para todas as *point clouds*, segmentadas com os três algoritmos, para limiar de corte variando entre 1 e 30. (estes resultados também estão listados no anexo, nas tabelas A.1 - A.13). Também são apresentados os resultados com limiar de corte igual a infinito, ou seja, sem cortar dos filamentos de cada camada.

4.2.2 Comparação dos algoritmos de geração de filamentos

Não foi encontrado um ótimo para o limiar de distância de corte. Limiares de distância de corte maiores produziram taxas melhores, mas foram testados apenas limiares menores ou iguais a 25. Como os melhores resultados foram para os filamentos não cortados, resta saber se há um limiar ótimo entre 25 e infinito.

Foi justificada a introdução de um limiar de comprimento de filamento para codificação abaixo do qual os filamentos são enviados como informação lateral. Como foi demonstrado, filamentos muito pequenos comprimidos pelo algoritmo proposto têm uma taxa pior do que a taxa dos dados brutos. O limiar de codificação ótimo ficou entre 6 e 12 para todos os métodos com todos os limiares de corte, exceto algumas anomalias. As anomalias foram para limiar de corte infinito (sem corte). Nas anomalias em que o valor ótimo de limiar de codificação é zero ou um, é porque não há muitas camadas com número pequeno de pontos, e portanto não foram gerados filamentos cuja a taxa após codificação é pior que a taxa bruta. A outra anomalia é a *point cloud* 2, que retornou um ótimo do limiar de transmissão em 72. Uma hipótese possível seria a geração de várias camadas com menos de 72 pontos que tiveram taxas muito ruins, provavelmente por serem compostas de vários *clusters* pequenos e desconexos e sem relação em seu sinal de cor.

As taxas obtidas foram muito similares para os métodos "exato" e "FLANN", com este último sendo na maioria das vezes um pouco melhor que o anterior. O algoritmo "Hilbert" teve uma performance satisfatória, bastante próxima dos dois outros, especialmente para limiares de corte maiores.

4.3 Comparação com resultados de trabalhos anteriores

Na tabela 4.2 está a comparação dos métodos aqui propostos com o melhor dos dois métodos propostos por [13].

Tabela 4.2: Comparação com os resultados de trabalhos anteriores

Arquivo	GPCC [30]	MTDC [13]	"exato"	"FLANN"	"Hilbert"
ricardo9/frame0000	5.79	13.76	9.94	9.68	10.32
ricardo9/frame0091	6.07	14.56	10.70	10.43	11.12
phil9/frame0116	9.67	22.16	15.30	14.95	15.77
andrew9/frame0263	10.53	24.56	16.67	16.50	17.03
sarah9/frame0146	5.67	16.48	10.83	10.61	11.32

5 Conclusão

Este capítulo resume e comenta os os resultados obtidos e aponta para trabalhos futuros.

5.1 Conclusão

Foi concebido e implementado com sucesso um esquema de compressão do sinal de cor de nuvens de pontos com a linguagem de programação Python. O algoritmo de processamento de geometria foi adaptado e melhorado a partir de [13], e consiste em dividir a nuvem em camadas para em seguida segmentar a nuvem de pontos em filamentos. Um parâmetro de entrada deste algoritmo é a distância máxima entre pontos consecutivos, que foi denominada "limar de corte". Foram usados três métodos de geração de filamentos: "exato", "FLANN" e "Hilbert". Os dois primeiros tiveram os melhores resultados, mas o algoritmo "Hilbert" é muito mais rápido. Para a compressão foi utilizada a codificação diferencial seguida de um codificador aritmético adaptativo iniciado com um um contexto genérico. O contexto do codificador aritmético adaptativo foi mantido de um filamento para o outro de forma que o algoritmo "aprende" as propriedades da nuvem. Como filamentos curtos não apresentam uma taxa boa quando comprimidos com o codificador aritmético, o esquema proposto envia os filamentos mais curtos que um certo limiar pelo canal de informação lateral. Este limiar de decisão do *codec* foi denominado limiar de transmissão.

Procurou-se otimizar os limiares de corte e de transmissão. Limiares de corte maiores produziram filamentos mais longos e levaram a taxas de compressão melhores: de fato, os melhores resultados foram para limiar de corte infinito, ou seja, com apenas um filamento por camada. Os limiares de transmissão ótimos encontrados ficaram entre 5 e 20 em na maior parte dos casos testados.

Em conclusão, o algoritmo proposto mostra uma melhora nas taxas de compressão em comparação com o trabalho que o inspirou [13]. No entanto, as taxas obtidas são muito piores que as atingidas pelo *codec* em desenvolvimento pelo MPEG [30]. Restam melhorias a serem implementadas, que são discutidas na próxima seção.

5.2 Trabalhos futuros

Nesta seção são apresentadas possibilidades de trabalhos futuros baseados no algoritmo apresentado.

5.2.1 Otimização do processamento de geometria

Sem mudar nada no algoritmo proposto, seria interessante determinar se existe um ótimo para o limiar de distância de corte para valores finitos maiores 25. Os trabalhos futuros podem tentar melhorar a geração de filamentos, procurando algoritmos menos custosos computacionalmente, mas que produzam filamentos de melhor qualidade. Por exemplo, neste trabalho a métrica usada para medir a distância entre *voxels* foi a distância euclidiana (norma L^2)). Uma opção é avaliar se usar outras métricas (por exemplo, normas L^1, L^∞) podem ter resultados satisfatórios. A vantagem de usar outras normas é que estas podem ter um custo computacional menor. A motivação principal por trás da divisão preliminar em camadas de espessura unitária é diminuir o espaço de busca dos algoritmos de pesquisa de vizinhos, fazendo com que os estes retornem resultados mais rapidamente. Poderia-se avaliar outros esquemas de divisão que gerem regiões de busca diferentes, como camadas mais grossas, ou regiões cúbicas.

Trabalhos futuros com curvas preenchedoras de espaço.

O algoritmo "Hilbert" mostrou-se quase três ordens de grandeza mais rápido em relação aos outros algoritmos, sem perder muito nas taxas de compressão. Portanto, a exploração de curvas preenchedoras de espaço parece um caminho promissor para compressão de *point clouds*.

No processamento de geometria, o presente trabalho separou as nuvens de pontos em camadas, e para cada camada utilizou uma curva preenchedora do plano para percorrer os *voxels* ocupados. Como existem curvas que preenchem o espaço tridimensional [26], talvez talvez seja possível obter resultados melhores implementando um codificador que percorre todos os pontos do espaço 3D, ao invés de passar pela etapa de corte de camadas.

Mesmo que se utilizem algoritmos com curvas em três dimensões, os problemas de "fratura", expostos anteriormente, não são resolvidos. Uma possibilidade de melhora seria a implementação de um passo de "correção" dos filamentos cortados, em que filamentos próximos mas separados por uma fratura poderiam ser conectados. Recentemente, foram desenvolvidos algoritmos eficientes de busca de vizinhos que utilizam as propriedades das curvas preenchedoras do espaço aliadas à representação com *octrees*[38]. Estes algoritmos poderiam ser utilizados no desenvolvimento de algoritmos de geração de filamento mais rápidos.

5.2.2 Melhorias no algoritmo de compressão

Uso de outros esquemas de codificação

Neste trabalho a geração de filamentos foi otimizada. Assim, seria interessante estudar como a geração de filamentos melhores afetaria outros esquemas de compressão diferentes dos aqui propostos. Por exemplo, um trabalho futuro seria aplicar o algoritmo de codifcação com transformada *wavelet* proposto em [32] aos resultados do processamento de geometria aqui apresentados.

Melhorias na codificação diferencial

Na parte do codificador diferencial, o algoritmo proposto estima os valores de cor a partir de um único vizinho. É interessante estudar se levando em conta mais pontos próximos o sinal de erro teria uma variância menor, e portanto resultaria em taxas melhores. Para diminuir ainda mais a taxa, seria interessante estudar o resultado de induzir perdas no estágio da codificação diferencial, fazendo o sinal de erro passar por um quantizador.

Escolha adaptativa de parâmetros ótimos

Este trabalho mostrou que é possível otimizar um dos dois parâmetros de entrada: o limiar de codificação. Uma melhoria ao algoritmo seria implementar um mecanismo automático de escolha destes parâmetros sem intervenção do usuário, visando otimizar a taxa de compressão.

Referências

[1] Zhang, C., Q. Cai, P. A. Chou, Z. Zhang e R. Martin-Brualla: *Viewport: A distributed, immersive teleconferencing system with infrared dot pattern.* IEEE MultiMedia, 20(1):17–27, Jan 2013. 1, 5

[2] Bruder, G., F. Steinicke e A. Nüchter: *Poster: Immersive point cloud virtual environments.* Em *2014 IEEE Symposium on 3D User Interfaces (3DUI)*, páginas 161–162, March 2014. 1, 5

[3] Sugiura, N. e T. Komuro: *Dynamic 3d interaction using an optical see-through hmd.* Em *2015 IEEE Virtual Reality (VR)*, páginas 359–360, March 2015. 1, 5

[4] Zhu, Q., L. Chen, Q. Li, M. Li, A. Nüchter e J. Wang: *3d lidar point cloud based intersection recognition for autonomous driving.* Em *2012 IEEE Intelligent Vehicles Symposium*, páginas 456–461, June 2012. 1, 5

[5] Zermas, D., I. Izzat e N. Papanikolopoulos: *Fast segmentation of 3d point clouds: A paradigm on lidar data for autonomous vehicle applications.* Em *2017 IEEE International Conference on Robotics and Automation (ICRA)*, páginas 5067–5073, May 2017. 1, 5

[6] Chen, Chao I e Roger Stettner: *Drogue tracking using 3D flash lidar for autonomous aerial refueling.* Em Turner, Monte D. e Gary W. Kamerman (editores): *Laser Radar Technology and Applications XVI*, volume 8037, páginas 216 – 226. International Society for Optics and Photonics, SPIE, 2011. `https://doi.org/10.1117/12.886572`. 1, 5

[7] Yao, AWL: *Applications of 3d scanning and reverse engineering techniques for quality control of quick response products.* The International Journal of Advanced Manufacturing Technology, 26(11-12):1284–1288, 2005. 1

[8] Höfle, Bernhard, Thomas Geist, Martin Rutzinger e Norbert Pfeifer: *Glacier surface segmentation using airborne laser scanning point cloud and intensity data.* International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, 36(Part 3):W52, 2007. 1, 5

[9] Abellán, A., J.M. Vilaplana e J. Martínez: *Application of a long-range terrestrial laser scanner to a detailed rockfall study at vall de núria (eastern pyrenees, spain).* Engineering Geology, 88(3):136 – 148, 2006, ISSN 0013-7952. `http://www.sciencedirect.com/science/article/pii/S001379520600247X`. 1, 5

[10] Naether, Silvio, Ursula Buck, Lorenzo Campana, Robert Breitbeck e Michael Thali: *The examination and identification of bite marks in foods using 3d scanning and 3d comparison methods*. International journal of legal medicine, 126(1):89–95, 2012. 1

[11] Dore, C. e M. Murphy: *Integration of historic building information modeling (hbim) and 3d gis for recording and managing cultural heritage sites*. Em *2012 18th International Conference on Virtual Systems and Multimedia*, páginas 369–376, Sep. 2012. 1, 5

[12] Zuffo, Marcelo: *University of são paulo point cloud dataset*. `http://uspaulopc.di.ubi.pt/`, acesso em 2019-11-18. 1

[13] Reis, Bruno José Bergamaschi Kumer: *Codificação das cores de uma point cloud através da sua divisão em filamentos*, 2018. 2, 13, 15, 24, 25, 38, 39

[14] Microsoft (Loop, Charles; Cai, Qin; Escolano, Sergio Orts; e Chou, Phillip A.): *Jpeg pleno database: Microsoft voxelized upper bodies - a voxelized point cloud dataset*. `https://jpeg.org/plenodb/pc/microsoft/`, acesso em 2019-10-31. 2, 24

[15] Gonzalez, Rafael C. e Richard E. Woods: *Digital Image Processing*. Pearson, 2018. 4, 9

[16] ScanLAB Projects: *Scanlab projects point cloud data sets: Biplane*. `http://grebjpeg.epfl.ch/jpeg_pc/index_Bi-plane.html`, acesso em 2019-11-18. 5

[17] Lubos, P., R. Beimler, M. Lammers e F. Steinicke: *Touching the cloud: Bimanual annotation of immersive point clouds*. Em *2014 IEEE Symposium on 3D User Interfaces (3DUI)*, páginas 191–192, March 2014. 5

[18] Sansoni, Giovanna, Marco Trebeschi e Franco Docchio: *State-of-the-art and applications of 3d imaging sensors in industry, cultural heritage, medicine, and criminal investigation*. Sensors, 9(1):568–601, 2009, ISSN 1424-8220. `https://www.mdpi.com/1424-8220/9/1/568`. 5

[19] Bourke, Paul: *Ply - polygon file format*. `http://paulbourke.net/dataformats/ply/`, acesso em 2019-10-31. 6

[20] Cover, Thomas M. e Joy A. Thomas: *Elements of Information Theory, Second Edition*. Wiley-Interscience, 2006. 8

[21] Sayood, Khalid: *Introduction to Data Compression, Fifth Edition*. Morgan Kaufmann, 2018. 8, 13

[22] Shannon, C. E.: *A mathematical theory of communication*. The Bell System Technical Journal, 27(3):379–423, July 1948. 8

[23] Sayood, Khalid: *Lossless Data Compression Handbook*. Academic Press, 2003. 9

[24] Marius Muja, David G. Lowe: *Fast approximate nearest neighbors with automatic algorithm configuration*. 2009. `https://www.cs.ubc.ca/research/flann/uploads/FLANN/flann_visapp09.pdf`. 10, 20

[25] Muja, Marius e David G Lowe: *Fast approximate nearest neighbors with automatic algorithm configuration.* VISAPP (1), 2(331-340):2, 2009. 10

[26] Bader, Michael: *Space-filling Curves.* Springer, 2013. 10, 40

[27] Przemyslaw Prusinkiewicz, Aristid Lindenmayer: *The Algorithmic Beauty of Plants.* Springer Verlag, 1990. `http://algorithmicbotany.org/papers/`. 11

[28] Kottwitz, Stefan: *Example: Lindenmayer systems.* `http://www.texample.net/tikz/examples/lindenmayer-systems/`, acesso em 2019-11-16. 12

[29] De Queiroz, Ricardo, Diogo Garcia, Philip Chou e Dinei Florencio: *Distance-based probability model for octree coding.* IEEE Signal Processing Letters, 25:1–1, abril 2018. 15

[30] ISO/IEC MPEG 3-Dimensional Graphics Team: *G-pcc codec description v4*, 2019. 15, 38, 39

[31] de Queiroz, Ricardo L. e Philip A. Chou: *Compression of 3d point clouds using a region-adaptive hierarchical transform.* IEEE Transactions on Image Processing, 25(8):3947–3956, 2016. 15

[32] de Macedo, Lucas Rezende: *Compressão de cor de point clouds utilizando transformada wavelets*, 2018. 15, 41

[33] Python Software Foundation: *Python language reference, version 3.7.0.* `http://www.python.org`, acesso em 2019-10-31. 23

[34] van der Walt, S., S. C. Colbert e G. Varoquaux: *The numpy array: A structure for efficient numerical computation.* Computing in Science Engineering, 13(2):22–30, March 2011. 23

[35] Qian-Yi Zhou, Jaesik Park, Vladlen Koltun: *Open3D: A modern library for 3D data processing.* arXiv:1801.09847, 2018. 23

[36] Griffiths, Scott: *Bitstring, a python module to help you manage your bits.* `https://scott-griffiths.github.io/bitstring/`, acesso em 2019-10-31. 23

[37] Hunter, J. D.: *Matplotlib: A 2d graphics environment.* Computing in Science Engineering, 9(3):90–95, May 2007. 23

[38] Holzmüller, David: *Efficient neighbor-finding on space-filling curves (bachelor thesis)*, 2017. `https://arxiv.org/abs/1710.06384`. 40

A Resultados da codificação para todas as *point clouds*

Tabela A.1: Resultados para a *point cloud* ricardo9, frame0000 com método "exato"

Limiar de corte	**Melhor limiar de transmissão**	**Taxa**	**Quantidade de filamentos não codificados**	**Parte da taxa correspondente aos filamentos não codificados**	**Parte da taxa correspondente aos filamentos codificados**
1.0	6	19.19	89%	13.43	5.76
2.0	6	13.79	77%	3.98	9.82
3.0	6	12.17	76%	2.38	9.79
4.0	7	11.63	76%	1.96	9.68
5.0	7	11.33	76%	1.71	9.61
6.0	7	11.17	75%	1.55	9.62
7.0	7	11.05	74%	1.42	9.63
8.0	7	10.93	71%	1.22	9.71
9.0	7	10.84	68%	1.08	9.76
10.0	7	10.78	65%	0.96	9.82
15.0	8	10.52	51%	0.54	9.98
20.0	8	10.34	36%	0.27	10.07
25.0	8	10.23	27%	0.15	10.08
∞	6	9.94	0%	0.00	9.94

Tabela A.2: Resultados para a *point cloud* ricardo9, frame0000 com método "FLANN"

Limiar de corte	**Melhor limiar de transmissão**	**Taxa**	**Quantidade de filamentos não codificados**	**Parte da taxa correspondente aos filamentos não codificados**	**Parte da taxa correspondente aos filamentos codificados**
1.0	6	18.46	85%	12.16	6.30
2.0	6	12.18	61%	1.94	10.24
3.0	6	11.00	62%	1.05	9.95
4.0	7	10.65	63%	0.87	9.78
5.0	7	10.45	63%	0.76	9.70
6.0	7	10.35	63%	0.69	9.67
7.0	7	10.28	62%	0.62	9.66
8.0	7	10.24	60%	0.55	9.69
9.0	6	10.20	58%	0.50	9.70
10.0	6	10.16	57%	0.47	9.70
15.0	7	10.07	50%	0.35	9.72
20.0	7	10.02	45%	0.29	9.73
25.0	7	9.99	39%	0.23	9.76
∞	6	9.68	0%	0.00	9.68

Tabela A.3: Resultados para a *point cloud* ricardo9, frame0000 com método "Hilbert"

Limiar de corte	**Melhor limiar de transmissão**	**Taxa**	**Quantidade de filamentos não codificados**	**Parte da taxa correspondente aos filamentos não codificados**	**Parte da taxa correspondente aos filamentos codificados**
1.0	6	23.83	100%	23.62	0.21
2.0	6	22.93	98%	21.61	1.32
3.0	6	19.61	82%	13.90	5.71
4.0	6	17.82	69%	9.82	8.01
5.0	6	15.65	47%	4.87	10.78
6.0	6	14.82	40%	3.54	11.28
7.0	6	14.05	29%	2.11	11.94
8.0	6	13.52	25%	1.56	11.96
9.0	6	13.01	18%	0.86	12.15
10.0	6	12.71	15%	0.64	12.07
15.0	6	11.97	9%	0.21	11.75
20.0	6	11.51	7%	0.14	11.37
25.0	5	11.27	7%	0.11	11.16
∞	6	10.32	0%	0.00	10.32

Tabela A.4: Resultados para a *point cloud* ricardo9, frame0091 com método "exato"

Limiar de corte	**Melhor limiar de transmissão**	**Taxa**	**Quantidade de filamentos não codificados**	**Parte da taxa correspondente aos filamentos não codificados**	**Parte da taxa correspondente aos filamentos codificados**
1.0	6	20.05	91%	14.87	5.18
2.0	6	14.67	78%	4.58	10.09
3.0	6	13.06	75%	2.58	10.48
4.0	6	12.57	74%	2.14	10.43
5.0	7	12.25	72%	1.84	10.41
6.0	7	12.08	71%	1.65	10.43
7.0	7	11.94	68%	1.45	10.49
8.0	7	11.83	66%	1.30	10.53
9.0	8	11.73	63%	1.16	10.57
10.0	8	11.66	60%	1.03	10.63
15.0	8	11.40	45%	0.59	10.81
20.0	9	11.25	34%	0.34	10.91
25.0	8	11.15	25%	0.21	10.94
∞	72	10.70	0%	0.00	10.70

Tabela A.5: Resultados para a *point cloud* ricardo9, frame0091 com método "FLANN"

Limiar de corte	**Melhor limiar de transmissão**	**Taxa**	**Quantidade de filamentos não codificados**	**Parte da taxa correspondente aos filamentos não codificados**	**Parte da taxa correspondente aos filamentos codificados**
1.0	6	18.89	89%	13.09	5.80
2.0	6	13.19	68%	2.67	10.51
3.0	7	12.06	66%	1.58	10.48
4.0	7	11.69	63%	1.19	10.50
5.0	7	11.48	60%	0.97	10.51
6.0	7	11.37	57%	0.86	10.52
7.0	7	11.29	55%	0.76	10.53
8.0	7	11.23	53%	0.68	10.55
9.0	7	11.17	51%	0.60	10.57
10.0	7	11.12	50%	0.56	10.57
15.0	7	10.98	43%	0.38	10.60
20.0	7	10.92	38%	0.30	10.62
25.0	7	10.87	35%	0.25	10.62
∞	72	10.43	0%	0.00	10.43

Tabela A.6: Resultados para a *point cloud* ricardo9, frame0091 com método "Hilbert"

Limiar de corte	**Melhor limiar de transmissão**	**Taxa**	**Quantidade de filamentos não codificados**	**Parte da taxa correspondente aos filamentos não codificados**	**Parte da taxa correspondente aos filamentos codificados**
1.0	6	23.81	100%	23.63	0.17
2.0	6	22.99	98%	21.83	1.17
3.0	6	20.28	85%	15.18	5.11
4.0	7	18.33	71%	10.17	8.16
5.0	7	16.41	53%	5.70	10.70
6.0	7	15.42	41%	3.52	11.89
7.0	7	14.95	37%	2.87	12.09
8.0	7	14.47	33%	2.26	12.22
9.0	7	13.85	25%	1.27	12.58
10.0	7	13.49	22%	0.95	12.54
15.0	7	12.72	16%	0.44	12.28
20.0	6	12.29	13%	0.27	12.02
25.0	7	12.11	12%	0.21	11.90
∞	72	11.12	0%	0.00	11.12

Tabela A.7: Resultados para a *point cloud* phil9, frame0116 com método "exato"

Limiar de corte	**Melhor limiar de transmissão**	**Taxa**	**Quantidade de filamentos não codificados**	**Parte da taxa correspondente aos filamentos não codificados**	**Parte da taxa correspondente aos filamentos codificados**
1.0	9	21.96	93%	16.17	5.80
2.0	9	18.47	82%	5.72	12.75
3.0	9	17.19	78%	3.24	13.95
4.0	9	16.75	77%	2.51	14.23
5.0	9	16.48	76%	2.15	14.34
6.0	10	16.35	75%	1.96	14.39
7.0	10	16.24	74%	1.79	14.45
8.0	11	16.16	73%	1.65	14.51
9.0	12	16.10	71%	1.51	14.58
10.0	12	16.04	68%	1.38	14.66
15.0	14	15.85	56%	0.87	14.98
20.0	14	15.74	44%	0.54	15.20
25.0	13	15.67	34%	0.36	15.31
∞	0	15.30	0%	0.00	15.30

Tabela A.8: Resultados para a *point cloud* phil9, frame0116 com método "FLANN"

Limiar de corte	**Melhor limiar de transmissão**	**Taxa**	**Quantidade de filamentos não codificados**	**Parte da taxa correspondente aos filamentos não codificados**	**Parte da taxa correspondente aos filamentos codificados**
1.0	9	21.28	91%	14.46	6.82
2.0	9	17.41	73%	3.67	13.74
3.0	9	16.39	70%	2.11	14.29
4.0	9	16.06	68%	1.63	14.43
5.0	10	15.86	65%	1.26	14.60
6.0	10	15.76	63%	1.10	14.66
7.0	10	15.69	61%	0.98	14.71
8.0	9	15.63	59%	0.88	14.75
9.0	10	15.58	57%	0.79	14.79
10.0	10	15.55	56%	0.74	14.80
15.0	10	15.42	50%	0.54	14.88
20.0	10	15.34	46%	0.43	14.91
25.0	10	15.30	42%	0.36	14.94
∞	0	14.95	0%	0.00	14.95

Tabela A.9: Resultados para a *point cloud* phil9, frame0116 com método "Hilbert"

Limiar de corte	**Melhor limiar de transmissão**	**Taxa**	**Quantidade de filamentos não codificados**	**Parte da taxa correspondente aos filamentos não codificados**	**Parte da taxa correspondente aos filamentos codificados**
1.0	9	23.93	100%	23.76	0.17
2.0	9	23.62	99%	22.48	1.13
3.0	10	22.51	91%	17.76	4.76
4.0	10	21.67	82%	14.09	7.58
5.0	10	20.55	68%	9.45	11.10
6.0	10	19.93	59%	7.21	12.72
7.0	11	19.49	53%	5.77	13.72
8.0	11	18.98	44%	4.04	14.94
9.0	10	18.57	37%	2.91	15.65
10.0	10	18.24	33%	2.24	16.01
15.0	9	17.46	21%	0.85	16.61
20.0	9	17.00	16%	0.45	16.55
25.0	9	16.77	14%	0.33	16.44
∞	0	15.77	0%	0.00	15.77

Tabela A.10: Resultados para a *point cloud* andrew9, frame0263 com método "exato"

Limiar de corte	**Melhor limiar de transmissão**	**Taxa**	**Quantidade de filamentos não codificados**	**Parte da taxa correspondente aos filamentos não codificados**	**Parte da taxa correspondente aos filamentos codificados**
1.0	11	22.33	93%	15.44	6.89
2.0	11	19.46	82%	5.44	14.02
3.0	11	18.28	79%	2.94	15.35
4.0	12	17.90	79%	2.35	15.55
5.0	11	17.66	79%	2.01	15.65
6.0	12	17.54	78%	1.83	15.70
7.0	12	17.45	77%	1.69	15.76
8.0	12	17.38	76%	1.55	15.83
9.0	12	17.32	74%	1.44	15.88
10.0	14	17.27	72%	1.32	15.95
15.0	14	17.13	61%	0.88	16.24
20.0	18	17.02	48%	0.56	16.46
25.0	18	16.95	36%	0.32	16.63
∞	1	16.67	0%	0.00	16.67

Tabela A.11: Resultados para a *point cloud* andrew9, frame0263 com método "FLANN"

Limiar de corte	**Melhor limiar de transmissão**	**Taxa**	**Quantidade de filamentos não codificados**	**Parte da taxa correspondente aos filamentos não codificados**	**Parte da taxa correspondente aos filamentos codificados**
1.0	11	22.02	91%	14.66	7.35
2.0	11	18.63	72%	3.35	15.28
3.0	11	17.72	71%	1.90	15.82
4.0	12	17.40	72%	1.55	15.85
5.0	12	17.23	70%	1.30	15.92
6.0	12	17.14	69%	1.15	15.99
7.0	12	17.08	68%	1.03	16.04
8.0	11	17.02	66%	0.93	16.09
9.0	12	16.98	64%	0.82	16.16
10.0	12	16.95	62%	0.75	16.20
15.0	12	16.84	56%	0.55	16.29
20.0	12	16.78	51%	0.42	16.36
25.0	12	16.75	46%	0.34	16.40
∞	1	16.50	0%	0.00	16.50

Tabela A.12: Resultados para a *point cloud* andrew9, frame0263 com método "Hilbert"

Limiar de corte	**Melhor limiar de transmissão**	**Taxa**	**Quantidade de filamentos não codificados**	**Parte da taxa correspondente aos filamentos não codificados**	**Parte da taxa correspondente aos filamentos codificados**
1.0	11	23.94	100%	23.73	0.21
2.0	12	23.69	99%	22.52	1.18
3.0	12	22.86	92%	17.92	4.94
4.0	12	22.22	83%	14.40	7.82
5.0	12	21.45	72%	10.63	10.81
6.0	12	20.84	62%	7.80	13.04
7.0	13	20.43	54%	6.13	14.30
8.0	12	20.00	46%	4.37	15.64
9.0	12	19.56	37%	2.85	16.70
10.0	11	19.26	31%	2.09	17.17
15.0	11	18.51	17%	0.68	17.84
20.0	10	18.08	13%	0.36	17.71
25.0	10	17.86	10%	0.22	17.64
∞	1	17.03	0%	0.00	17.03

Tabela A.13: Resultados para a *point cloud* sarah9, frame0146 com método "exato"

Limiar de corte	**Melhor limiar de transmissão**	**Taxa**	**Quantidade de filamentos não codificados**	**Parte da taxa correspondente aos filamentos não codificados**	**Parte da taxa correspondente aos filamentos codificados**
1.0	6	19.34	89%	13.25	6.09
2.0	7	14.43	78%	4.17	10.26
3.0	7	12.95	77%	2.47	10.48
4.0	7	12.52	77%	2.12	10.40
5.0	7	12.26	76%	1.88	10.38
6.0	8	12.11	75%	1.73	10.37
7.0	7	11.99	73%	1.56	10.44
8.0	8	11.90	71%	1.42	10.48
9.0	8	11.82	69%	1.29	10.54
10.0	8	11.75	66%	1.17	10.59
15.0	8	11.52	53%	0.72	10.80
20.0	9	11.35	40%	0.41	10.93
25.0	8	11.23	30%	0.24	10.99
∞	0	10.83	0%	0.00	10.83

Tabela A.14: Resultados para a *point cloud* sarah9, frame0146 com método "FLANN"

Limiar de corte	**Melhor limiar de transmissão**	**Taxa**	**Quantidade de filamentos não codificados**	**Parte da taxa correspondente aos filamentos não codificados**	**Parte da taxa correspondente aos filamentos codificados**
1.0	6	18.69	87%	12.25	6.44
2.0	6	13.09	69%	2.53	10.56
3.0	7	12.09	69%	1.61	10.48
4.0	7	11.79	67%	1.28	10.50
5.0	7	11.59	64%	1.08	10.51
6.0	7	11.49	63%	0.97	10.52
7.0	7	11.41	61%	0.88	10.52
8.0	7	11.34	60%	0.81	10.53
9.0	7	11.29	58%	0.74	10.55
10.0	7	11.24	56%	0.68	10.57
15.0	9	11.12	49%	0.49	10.63
20.0	9	11.06	43%	0.39	10.67
25.0	9	11.01	39%	0.32	10.69
∞	0	10.61	0%	0.00	10.61

Tabela A.15: Resultados para a *point cloud* sarah9, frame0146 com método "Hilbert"

Limiar de corte	**Melhor limiar de transmissão**	**Taxa**	**Quantidade de filamentos não codificados**	**Parte da taxa correspondente aos filamentos não codificados**	**Parte da taxa correspondente aos filamentos codificados**
1.0	6	23.88	100%	23.76	0.12
2.0	6	23.29	98%	22.41	0.89
3.0	6	20.78	88%	16.18	4.60
4.0	6	19.29	77%	12.32	6.97
5.0	6	17.51	61%	7.86	9.65
6.0	6	16.30	49%	5.17	11.13
7.0	6	15.61	43%	4.02	11.59
8.0	6	14.91	35%	2.67	12.25
9.0	6	14.32	29%	1.83	12.49
10.0	8	13.92	23%	1.14	12.77
15.0	7	13.02	15%	0.46	12.57
20.0	8	12.60	11%	0.24	12.36
25.0	7	12.39	10%	0.17	12.22
∞	0	11.32	0%	0.00	11.32