Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

Compressão de nuvens de pontos

Miguel de Carvalho Pachá

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Engenharia Elétrica

Orientador
Prof. Dr. Eduardo Peixoto Fernandes da Silva

Brasília
2019

Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

Compressão de nuvens de pontos

Miguel de Carvalho Pachá

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Engenharia Elétrica

Prof. Dr. Eduardo Peixoto Fernandes da Silva (Orientador)
niversidade de Brasília

Prof. Dr. Prof. Dr.
University Institute

Prof.a Dr.a
Coordenadora do Bacharelado em Engenharia Elétrica

Brasília, 16 de março de 2019

Dedicatória

Dedicatória

Agradecimentos

Obrigado!

Resumo

O *resumo* é um texto inaugural para quem quer conhecer o trabalho, deve conter uma breve descrição de todo o trabalho (apenas um parágrafo). Portanto, só deve ser escrito após o texto estar pronto. Não é uma coletânea de frases recortadas do trabalho, mas uma apresentação concisa dos pontos relevantes, de modo que o leitor tenha uma ideia completa do que lhe espera. Uma sugestão é que seja composto por quatro pontos: 1) o que está sendo proposto, 2) qual o mérito da proposta, 3) como a proposta foi avaliada/validada, 4) quais as possibilidades para trabalhos futuros. É seguido de (geralmente) três palavras-chave que devem indicar claramente a que se refere o seu trabalho. Por exemplo: *Este trabalho apresenta informações úteis a produção de trabalhos científicos para descrever e exemplificar como utililzar a classe LaTeX do Departamento de Ciência da Computação da Universidade de Brasília para gerar documentos. A classe* `UnB-EnE` *define um padrão de formato para textos do EnE/FT-UnB, facilitando a geração de textos e permitindo que os autores foquem apenas no conteúdo. O formato foi aprovado pelos professores do Departamento e utilizado para gerar este documento. Melhorias futuras incluem manutenção contínua da classe e aprimoramento do texto explicativo.*

Palavras-chave: LaTeX, metodologia científica, trabalho de conclusão de curso

Abstract

O *abstract* é o resumo feito na língua Inglesa. Embora o conteúdo apresentado deva ser o mesmo, este texto não deve ser a tradução literal de cada palavra ou frase do resumo, muito menos feito em um tradutor automático. É uma língua diferente e o texto deveria ser escrito de acordo com suas nuances (aproveite para ler `http://dx.doi.org/10.6061%2Fclinics%2F2014(03)01`).

Por exemplo: *This work presents useful information on how to create a scientific text to describe and provide examples of how to use the Computer Science Department's LaTeX class. The* ***UnB-EnE*** *class defines a standard format for texts, simplifying the process of generating CIC documents and enabling authors to focus only on content. The standard was approved by the Department's professors and used to create this document. Future work includes continued support for the class and improvements on the explanatory text.*

Keywords: LaTeX, scientific method, thesis

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

BCE Biblioteca Central.

CAPES Coordenação de Aperfeiçoamento de Pessoal de Nível Superior.

CEP Comissão de Ética Pública.

CTAN Comprehensive TEX Archive Network.

EnE Departamento de Engenharia Elétrica.

UnB Universidade de Brasília.

1 Introdução

Este capítulo introduz o tema.

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

2.1.1 Imagens em duas e três dimensões

O processamento digital de imagens trata primeiramente de imagens bidimensionais. Em [2] uma imagem é definada 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.

2.1.2 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). Isso 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 tridimensionais, onde cada ponto possui atributo de cor. Estes pontos com atributo de cor são chamados *voxels*, do inglês *volume pixel*.
A representação em nuvens de pontos é uma representação esparsa, diferente da representação de uma imagem bidimensional como uma foto, por exemplo. Enquanto uma imagem bidimensional geralmente é entendida como uma função que associa a todo e qualquer pixel um dado de cor, as representação de uma *point cloud* não é assim, porque nem todos os *voxels* são ocupados. Para quantificar a taxa de dados de uma representação de uma nuvem de pontos, mede-se quantos bits são gastos, em média, para representar um voxel ocupado. A unidade é abreviada *bpov* (*bits per occupied voxel*).

2.1.3 Aplicações

Representações em nuvens de pontos podem ser usadas para criar ambientes imersivos, seja para entretenimento (como filmes e jogos) ou para comunicação pessoal (vídeo-chamadas tridimensionais). Uma outra aplicação pode ser encontrada no desenvolvimento de sistemas de visão computacional para a carros autônomos a partir de dados de *LiDAR* (4). Aliadas a sistemas de informação geográfica, nuvens de pontos também podem ser utilizadas também para representar relevo e cobertura do terreno, tendo assim uma possível aplicação na agrimensura e no manejo de recursos naturais. Também podem ser usadas para fins acadêmicos, por exemplo para a preservação e estudo de ambientes ou objetos de valor histórico.

2.1.4 Obtenção e transmissão de nuvens de pontos

Nuvens de pontos podem ser obtidas seja a partir de dados brutos gerados por tecnologias especializadas, como *LiDAR* e *scanners* tridimensionais, ou então a partir do cruzamento de imagens convencionais em duas dimensões. Este processo de obtenção de *point clouds* é denominado registro de pontos (em ingês *point set registration*). Um dos formatos de arquivo utilizado para salvar nuvens de pontos é o formato PLY (Polygon File Format - uma descrição está disponível em [3]). Apesar de ter menos informação do que representações poligonais, este tipo de modelagem gera arquivos muito grandes, o que dificulta sua transmissão. De fato, o tamanho de cada quadro dos vídeos tridimensionais estudados neste trabalho é da ordem de alguns *megabytes*. Portanto, é interessante comprimir este tipo de informação para transmissão ou arquivamento.

2.2 Compressão de dados

2.2.1 Informação e entropia

Como notam Thomas e Cover [4],"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. 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 (este pode ser finito ou infinito). 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 [5].

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 quer dizer que não é possível comprimir um sinal abaixo de sua entropia sem perder informação ou acrescentar informação oriunda de outra fonte.

2.2.2 Compressão

Sayood [6] 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. A compressão têm 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 métrica 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,por exemplo 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 e codificação de entropia

As técnicas de compresssão são tipicamente compostas por uma sequência de vários estágios. [7] A relação entre esses componentes pode ser vista na figura.
emph(incluir figura)
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. Por isso, ela deve ser adaptada à natureza do sinal, e caso ela induza perdas, ela deve ser adaptada 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. Além disso, eles também tendem a ser parecidos com os pixels da mesma região de quadros anteriores. Essa semelhança implica uma redundância que pode ser explorada na compressão. 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 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 que pode ser aproveitada pela codificação diferencial, que está descrita na próxima seção.

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)(n^2). No entanto, é possível realizar um pré-processamento dos pontos, gerando uma estrutura denominda árvore K-D, que particiona o plano em regiões contendo quantidades iguais de pontos. Esse processo inicial tem a complexidade $\mathcal{O}(??)$. Usando essa árvore a procura pelos KNN 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 FLANN é uma heurística que permite fazer isso de forma mais rápida, mas não garante uma seleção exata.

2.3.2 Curvas preennchedoras de espaço

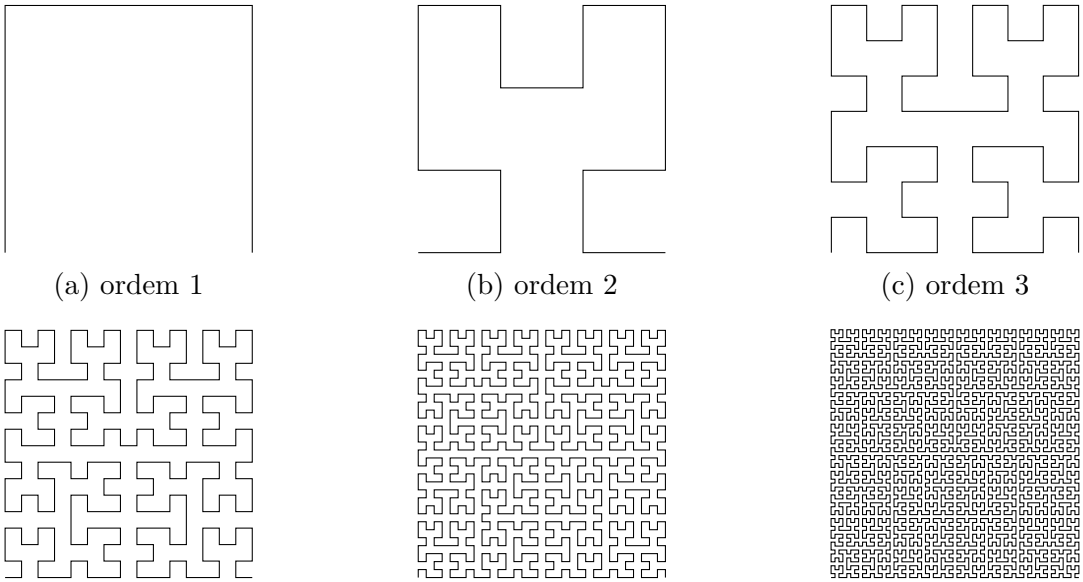
As curvas preenchedoras de espaço (*Space-filling curves*, ou *SPF*s) são curvas fractais que preenchem um hipercubo unitário[8]. Elas podem ser interpretadas como uma função sobrejetiva e contínua que leva do intervalo unitário $[0, 1]$ a um hipercubo unitário $[0, 1]^n$. A sobrejetividade de tais funções 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 converso não é sempre verdadeiro: dois pontos próximos no espaço $\mathbf{R}^n$ podem ter suas imagens afastadas no domínio $\mathbf{R}$. 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[9]. 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*).
Interessam a este trabalho em particular aproximações discretas de curvas preenchedoras de plano, ou seja, funções sequências H^j que associam um conjunto de inteiros $\{1, 2...N^2\}$ ao conjunto $\{1, 2...N\}^2$. Essas funções podem ser representadas como um percurso contínuo pela grade quadrada de pontos x, y onde $0 \leq x < N$ e $0 \leq y < N$.
Um exemplo de SFC é a curva de Hilbert, cujas primeiras iterações podem ser vistas na figura **??**. 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 geométrica:

- $F \rightarrow$ 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 2.1: As primeiras iterações da curva de Hilbert. (geradas a partir de [1])

2.4 Codificação diferencial

A codificação diferencial aproveita a correlação entre o 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 simplemente usar o último símbolo enviado como estimativa. É o que foi feito no trabalho anterior [11] 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 [6].

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*, dentrod 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 [6], 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

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

. Escolhe-se um número contido neste último intervalo dessa sequência. Este número é a *tag* associada a mensagem. Basta transmitir a *tag*, a quantidade N de símbolos transmitidos, e a função de probabilidade acumulada. Para decodificar, basta partir do número recebido t_0 e proceder da seguinte maneira. Seja $D_I(t)$ a função que associa t ao intervalo da partição P_I que o contém. Basta então 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 precisão arbitrária (esta depende de cálculo simbólico e é computacionalmente custosa, ou simplesmente não está implementada). 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 sem perdas. Já existem algoritmos para comprimir a geometria de vídeos de *point clouds* [12].

Já o sinal de cor pode ser transmitido com ou sem perdas, dependendo do algoritmo. O estado da arte é o algoritmo da *Region-Adaptive Hierarchical Transform* (RAHT - Transformada Hierárquica adaptada à região), descrito em [13]. No artigo, os pontos são agrupados por proximidade em uma estrutura de árvore. Os trabalhos [11] e [14] dividem a imagem em sequências de pontos próximos denominadas filamentos. [14] utiliza a transformada wavelet para cada codificar cada filamento. [11] 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 trabalhos anteriores.

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 passo:

1. Procassamento de geometria
   (a) Separação da nuvem em camadas na direção z.
   (b) Geração de um filamento para cada camada.
   (c) Corte dos filamentos de cada camada em subfilamentos de acordo com o limiar de distância de corte.
2. Codificação
   (a) Codficação diferencial.
   (b) Codificação com o codec escolhido de acordo com o limiar de comprimento para transmissão: transmissão pelo canal de inforamação lateral (sem compressão) ou codifcação aritmética.

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). Em cada camada, escolhe-se um ponto arbitrariamente para ser o primeiro ponto do filamento. A cada iteração, a partir do último ponto incluido no filamento escolhe-se um ponto próximo

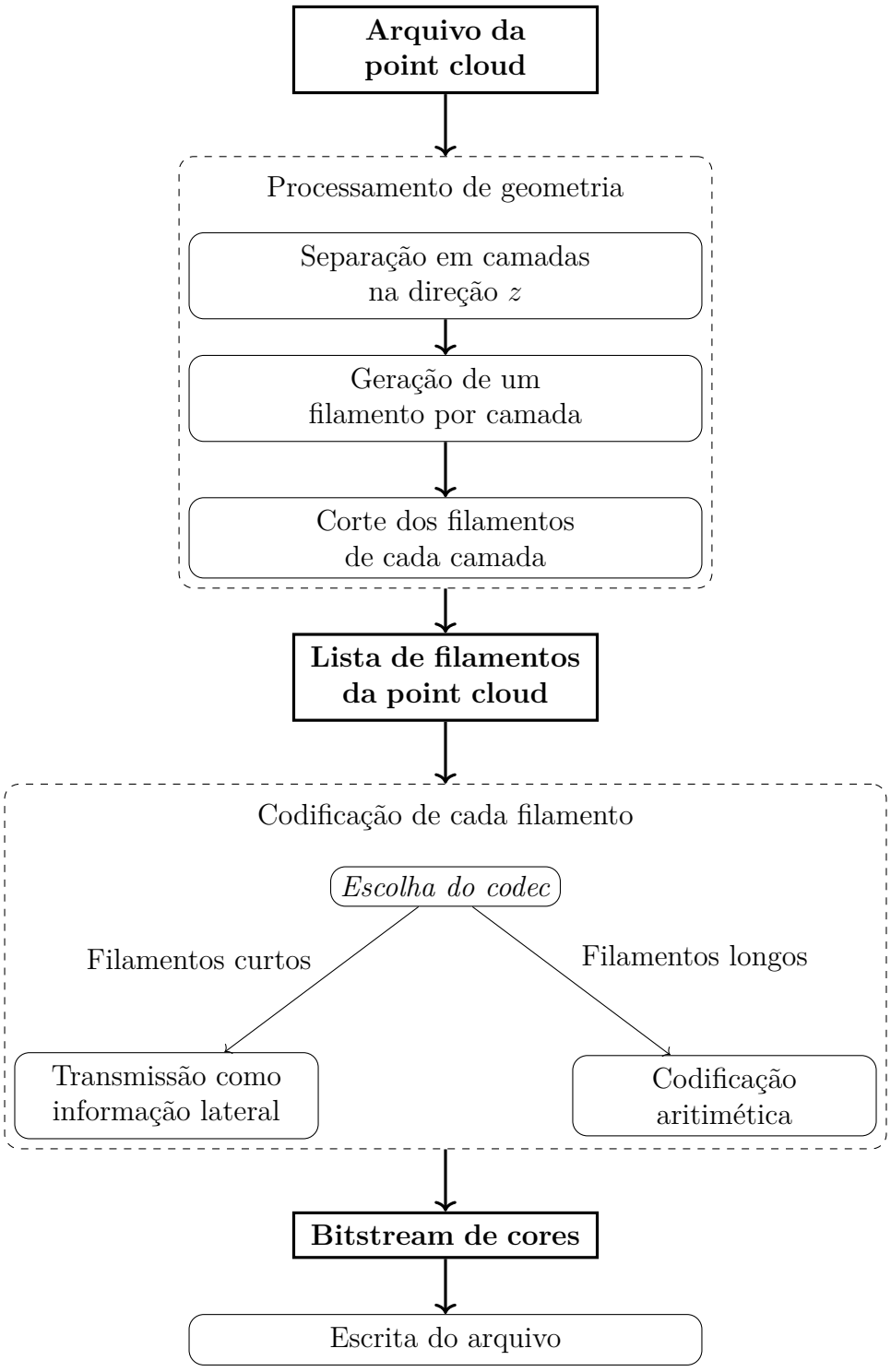


Figura 3.1: Organização do algoritmo proposto

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 $O(n^2)$. Trata-se de um algoritmo de força bruta e espera-se que ele seja o mais lento de todos.

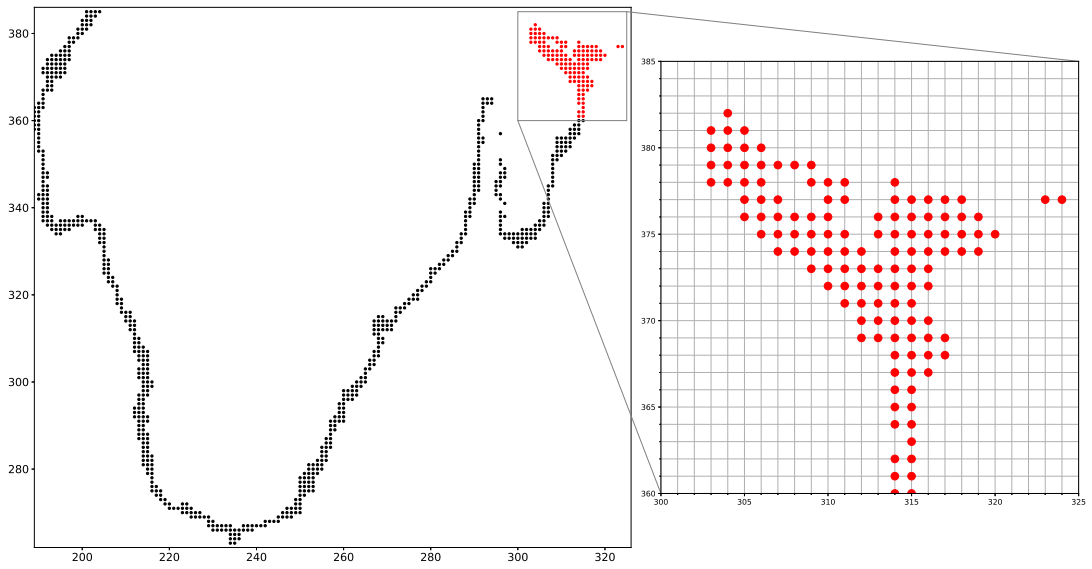
Algoritmo "FLANN"

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 implementa algoritmos de pesquisa de pontos próximos usando o método FLANN, descrito originalmente em [15]. Inicalmente é inicializada uma instância da classe com todos os pontos da camada. Escolhe-se (arbitrariamente) um ponto para começar o filamento, e o método

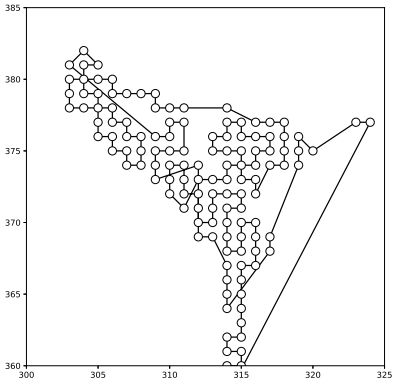
de pesquisa é usado a cada rodada para gerar uma lista de pontos próximos, e escolhe-se o menor dentre eles para visitar. Por ser um heurística, espera-se que este método seja mais rápido que o método "exato".

Algoritmo "Hilbert"

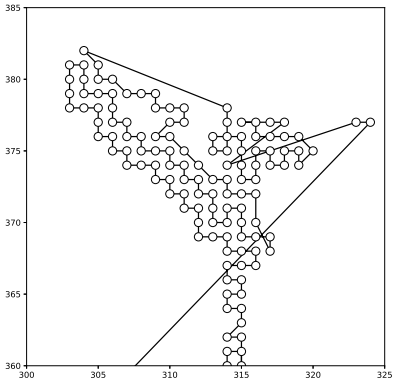
Este algoritmo visita os pontos da camada na ordem da curva de Hilbert. A curva de Hilbert é gerada previamente e guardada em uma tabela. É esperado que este algoritmo seja muito mais rápido que os dois outros, por empregar uma simples função de *look up*.



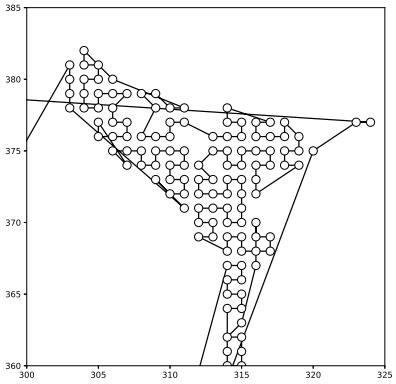
(a) Visão geral da camada, com destaque para o detalho mostrado nas subfiguras abaixo



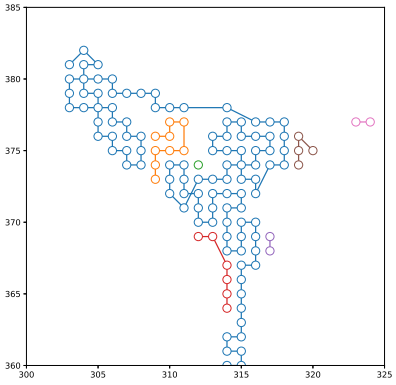
(b) Filamento gerado (método “exato”)



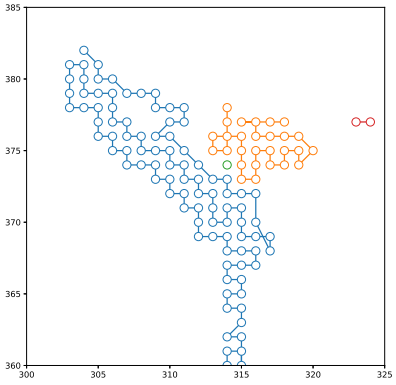
(c) Filamento gerado (método “FLANN”)



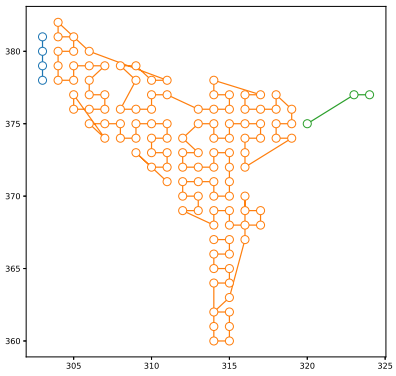
(d) Filamento gerado (método “Hilbert”)



(e) Filamentos cortados (limiar=8, “exato”)



(f) Filamentos cortados (limiar=8, “flann”)



(g) Filamentos cortados (limiar=8, “Hilbert”)

Figura 3.2: Ilustração da camada 150 da *point cloud* 1 em diferentes estágios do processamento, para os métodos “Exato”, “FLANN” e “Hilbert”

3.3 Escolha do método codificação

A codficação 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 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, faz sentido reaproveitar o contexto de codificação 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) [16]. Foram usadas as seguintes bibliotecas:

NumPy Para funções numéricas, especialmente para vetores e matrizes.[17]

Open3D Para processamento de nuvens de pontos (leitura de arquivos PLY, algoritmos de busca e renderização). [18]

bitstring Para a escrita e leitura *bit*-a-*bit* dos arquivos.[19]

Matplotlib para gerar os gráficos presentes deste relatório. [20]

Tabela 3.1: *Point clouds* escolhidas para o trabalho.

Nome	**Vídeo**	**Número de quadro**	**Voxels ocupados**
Pointcloud 1	Ricardo9	0001	214656
Pointcloud 2	Ricardo9	0091	337803
Pointcloud 3	Phil9	0116	385467
Pointcloud 4	Andrew9	0263	277583
Pointcloud 5	Sarah9	0146	304597

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 diferentes para otimizar os parâmetros de compressão. Os testes foram feitas sobre alguns quadros da base de dados *Microsoft Voxelized Upper Bodies* [21]. 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 (incluir uma figura com renderizações das pointclouds)

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 do comprimento dos filamentos, comparando com a transmissão sem codificação. Foram comparados os resultados do codificador proposto com os resultados do codificador proposto por [11] e com os resultados do algoritmo *RAHT*, também citados em [11].

4 Resultados

Este capítulo apresenta os resultados obtidos.

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

4.1.1 Análise do tempo de execução dos algoritmos de segmentação

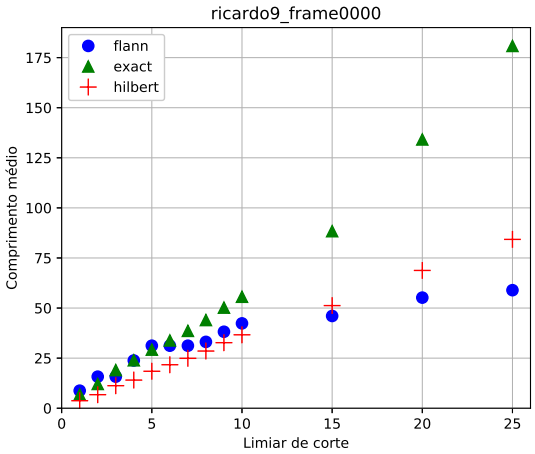
Podemos ver na tabela Tabela 4.1 o tempo de execução dos três algoritmos de segmentação para as cinco nuvens de pontos estudadas. Como esperado, nota-se que para todos as *point clouds*, o algoritmo "Exato" é o mais lento, o algoritmo "FLANN" é bem mais rápido e o algoritmo "Hilbert" é quase três ordens de grandeza mais rápido comparado aos anteriores.

Tabela 4.1: Tempo de execução dos algoritmos em segundos.

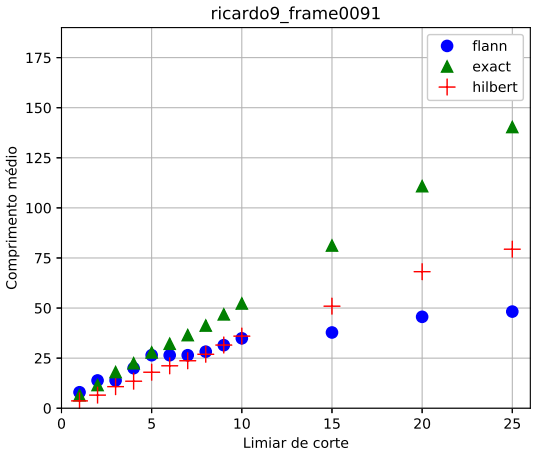
	Exato	**FLANN**	**Hilbert**
Point cloud 1	47.34	32.37	0.13
Point cloud 2	106.51	80.29	0.21
Point cloud 3	63.61	43.28	0.18
Point cloud 4	72.00	48.66	0.20
Point cloud 5	112.85	78.88	0.24

4.1.2 Parâmetro: limiar de distância de corte

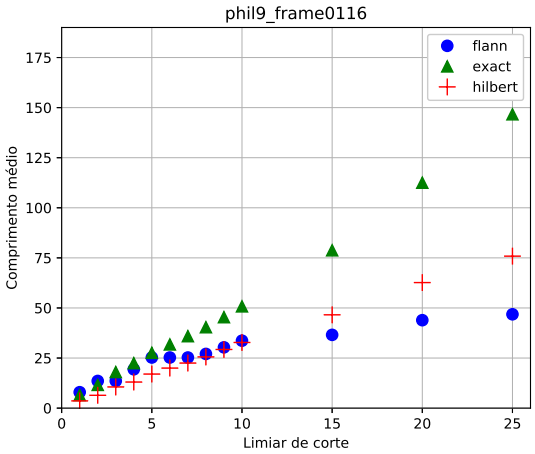
Nos gráficos da figura 4.1 podemos ver a variação do comprimento médio dos filamentos gerados por diferentes métodos segmentação em função do limiar de distância de corte. Podemos ver que, como esperado, para todos os arquivos, o comprimento médio tende a aumentar com o limiar do corte. Além disso, o método "exato" tende a gerar filamentos mais compridos, especialmente para limiares maiores. De maneira surpreendente, o algoritmo "Hilbert" gerou, em média, filamentos mais compridos do que o algoritmo "FLANN".



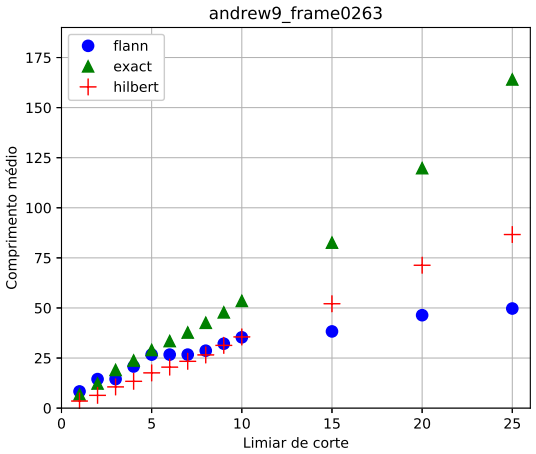
(a) Point cloud 1



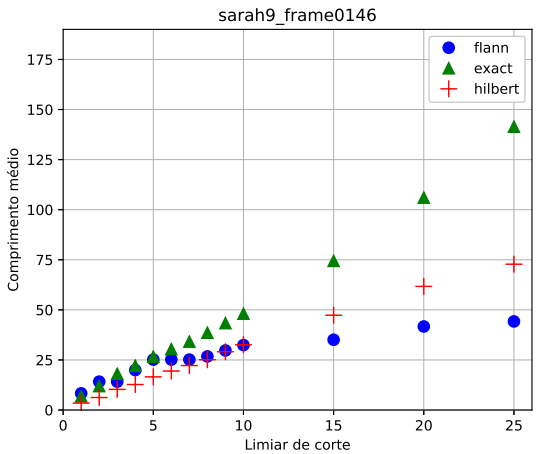
(b) Point cloud 2



(c) Point cloud 3



(d) Point cloud 4



(e) Point cloud 5

Figura 4.1: Comprimento médio dos filamentos em função do limiar de distância de corte, para diferentes métodos de geração de filamento

4.2 Análise do algoritmo de codificação

intro

4.2.1 Point cloud 1

pass

4.2.2 Point cloud 2

pass

4.2.3 Point cloud 3

pass

4.2.4 Point cloud 4

pass

4.2.5 Point cloud 5

pass

5 Conclusão

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

5.1 Análise dos resultados obtidos

intro

5.1.1 Comparação dos algoritmos de geração de filamentos

5.1.2 Custo computacional e qualidade dos filamentos

Tempo levado, histogramas, etc

5.1.3 Otimização de um parâmetro: limiar de distância de corte

bla

5.2 Análise do algoritmo de codificação

intro

5.2.1 Parâmetro: limiar de codificação

bla

5.3 Trabalhos futuros

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

5.3.1 Otimização do processamento de geometria

Os trabalhos futuros podem tentar melhorar a geraçãod e filamentos, procurando algoritmos menos custosos computacionalmente, mas que produzam filamentos de melhor qualidade. 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.
No processamento de geometria com curvas preenchedoras de espaço, 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 os espaço tridimensional [8], seria interessente implementar um codificador que percorre todos os pontos do espaço 3D, ao invés de passar pela etapa de corte de camadas. Recentemente, foram desenvolvidos algoritmos que utilizam as propriedades das curvas preenchedoras do espaço para encontrar vizinhos de pontos em conjuntos esparsos, com auxílio da representação com *octrees*[10].

5.3.2 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, quantificando o sinal de erro.

5.3.3 Escolha adaptativa de parâmetros ótimos

Este trabalho mostrou que é possível otimizar dois parâmetros de entrada: o limiar de distância de corte de filamentos, e o limiar de codificação que define se cada filamento é codificado com o codificador aritimético ou se é transmitido como informação lateral. Neste trabalho estes parâmetros são pré-definidos pelo usuário do codificador antes da codificação. Uma melhoria ao algoritmo seria implementar um mecanismo automático de escolha destes parâmetros sem intervenção do usuário, visando otimizar a taxa de compressão.

5.3.4 Codificação *inter-frame*

Finalmente, o escopo deste trabalho limitou-se à codificação inidividual de nuvens de ponto, ignorando a informação que poderia ser obtida dos quadros anteriores (codificação *intra-frame*). Um trabalho futuro possível é estudar a possibilidade de adaptar as técnicas aqui apresentadas à codificação *inter-frame*, que utiliza dados de quadros já transmitidos.

Referências

[1] Kottwitz, Stefan: *Example: Lindenmayer systems.* `http://www.texample.net/tikz/examples/lindenmayer-systems/`, acesso em 2019-11-16. ix, 8

[2] Rafael C. Gonzalez, Richard E. Woods: *Digital Image Processing.* Pearson, 2018. 2

[3] Bourke, Paul: *Ply - polygon file format.* `http://paulbourke.net/dataformats/ply/`, acesso em 2019-10-31. 3

[4] Thomas M. Cover, Joy A. Thomas: *Elements of Information Theory, Second Edition.* Wiley-Interscience, 2006. 4

[5] Shannon, C. E.: *A mathematical theory of communication.* The Bell System Technical Journal, 27(3):379–423, July 1948. 4

[6] Sayood, Khalid: *Introduction to Data Compression, Fifth Edition.* Morgan Kaufmann, 2018. 4, 8, 9

[7] Sayood, Khalid: *Lossless Data Compression Handbook.* Academic Press, 2003. 5

[8] Bader, Michael: *Space-filling Curves.* Springer, 2013. 6, 23

[9] Przemyslaw Prusinkiewicz, Aristid Lindenmayer: *The Algorithmic Beauty of Plants.* Springer Verlag, 1990. `http://algorithmicbotany.org/papers/`. 7

[10] Holzmüller, David: *Efficient neighbor-finding on space-filling curves (bachelor thesis)*, 2017. `https://arxiv.org/abs/1710.06384`. 23

[11] Reis, Bruno José Bergamaschi Kumer: *Codificação das cores de uma point cloud através da sua divisão em filamentos*, 2018. 8, 10, 17

[12] 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. 10

[13] 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. 10

[14] de Macedo, Lucas Rezende: *Compressão de cor de point clouds utilizando transformada wavelets*, 2018. 10

[15] 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. 13

[16] Foundation, Python Software: *Python language reference, version 3.7.0.* http://www.python.org, acesso em 2019-10-31. 16

[17] 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. 16

[18] Qian-Yi Zhou, Jaesik Park, Vladlen Koltun: *Open3D: A modern library for 3D data processing.* arXiv:1801.09847, 2018. 16

[19] Griffiths, Scott: *Bitstring, a python module to help you manage your bits.* https://scott-griffiths.github.io/bitstring/, acesso em 2019-10-31. 16

[20] Hunter, J. D.: *Matplotlib: A 2d graphics environment.* Computing in Science Engineering, 9(3):90–95, May 2007. 16

[21] Microsoft (Charles Loop, Qin Cai, Sergio Orts Escolano e Philip A. Chou): *Jpeg pleno database: Microsoft voxelized upper bodies - a voxelized point cloud dataset.* https://jpeg.org/plenodb/pc/microsoft/, acesso em 2019-10-31. 17