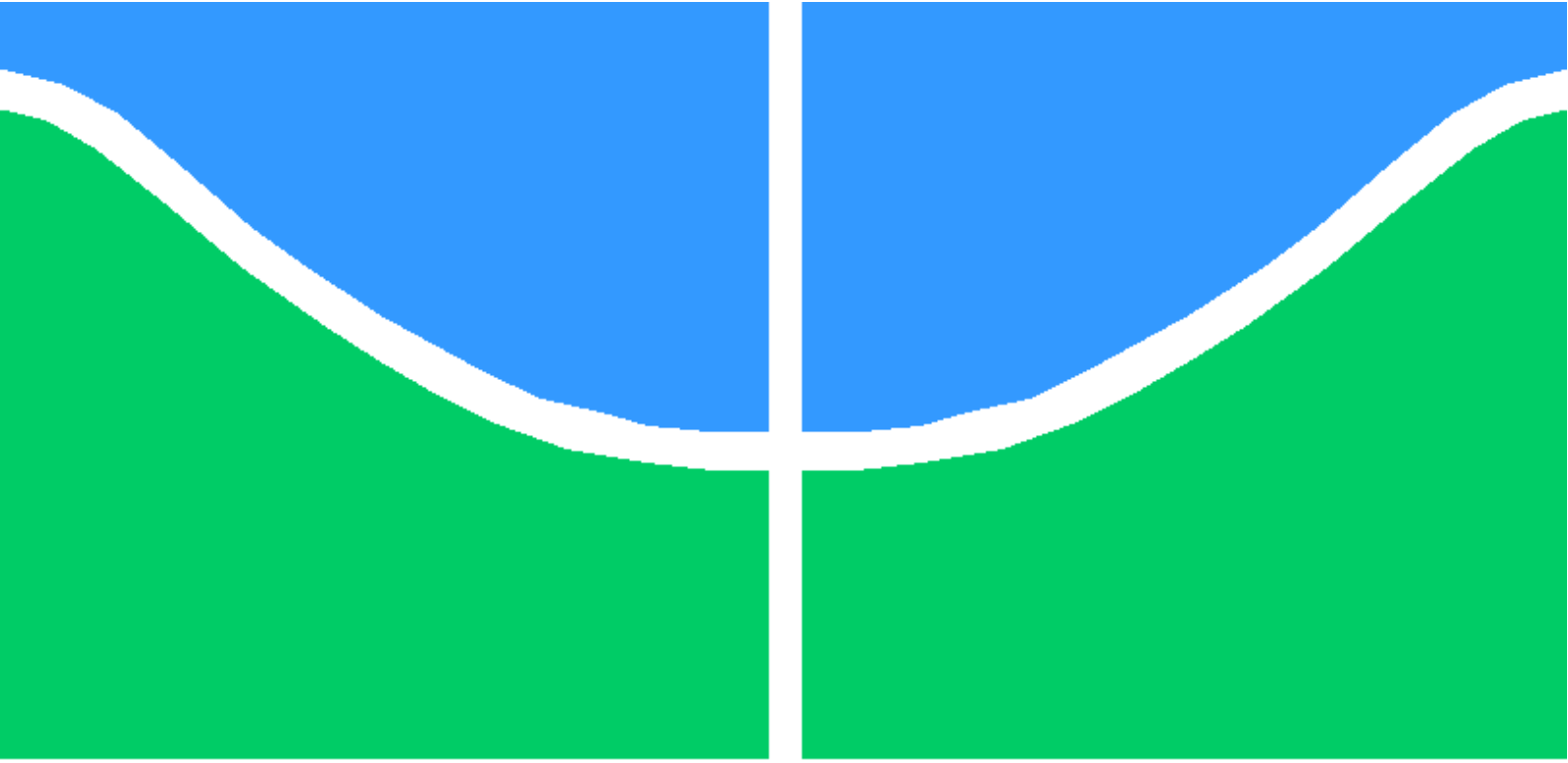

Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

Compressão de geometria de PointCloud por decomposição Booleana

Rodrigo Alves Rosário
Orientador: Prof. Eduardo Peixoto Fernandes da Silva, PhD.

Brasília
Dezembro de 2018

Rodrigo Alves Rosário

Compressão de geometria de PointCloud por decomposição Booleana

Trabalho submetido ao Departamento de Engenharia Elétrica da Universidade de Brasília como requisito parcial para obtenção do Título de Bacharel em Engenharia Elétrica.

Universidade de Brasília

Faculdade de Tecnologia

Orientador: Prof. Eduardo Peixoto Fernandes da Silva, PhD.

Brasília

Dezembro de 2018

Rodrigo Alves Rosário
Compressão de geometria de PointCloud por decomposição Booleana.
Brasília, Dezembro de 2018.
Orientador: Prof. Eduardo Peixoto Fernandes da Silva, PhD.

Projeto de Graduação – Universidade de Brasília
Faculdade de Tecnologia – Dezembro de 2018.

1. Compressão. 2. PointClouds. 3. Octrees.
Compressão de geometria de PointCloud por decomposição Booleana. ENE/FT – UnB/DF, Brasil.

Rodrigo Alves Rosário

Compressão de geometria de PointCloud por decomposição Booleana

Trabalho submetido ao Departamento de Engenharia Elétrica da Universidade de Brasília como requisito parcial para obtenção do Título de Bacharel em Engenharia Elétrica.

Banca Examinadora

Prof. Eduardo Peixoto Fernandes da Silva, PhD. – UnB/ENE
Orientador

Prof. Alexandre Zaghetto, Dr. – UnB/CIC
Examinador 1

Prof. Diogo Caetano Garcia, Dr. – UnB/FGA
Examinador 2

Brasília, 06 de Dezembro de 2018.

Dedico este trabalho aos meus pais, Ana e Pedro, à minha irmã Luciana e aos meus amigos, sem os quais não seria possível completar esta etapa de minha vida.

Agradecimentos

Agradeço primeiramente aos meus pais, Ana e Pedro, por todo o suporte e incentivos dados durante os longos anos de minha graduação. Acredito que sem os pequenos empurrões eu não teria conquistado metade do que conquistei em minha vida. Espero que esta conquista simbolize uma pequena parte de tantos esforços que vocês fizeram por mim.

Ao professor Eduardo, que aceitou ser meu orientador e garantiu todo o caminho a ser trilhado durante o projeto e que tornou essa etapa muito mais tranquila para mim.

Obrigado aos meus amigos e companheiros de curso por dividirem os melhores, e os piores, momentos dessa graduação. Vocês foram uma grande fonte de forças ao longo desses anos que dividimos histórias inesquecíveis e devo a vitória nessa jornada a vocês que me trouxeram até aqui, muitas vezes carregando nos ombros. Sempre estive apoiado nos ombros de gigantes, e assim pude ver mais longe.

Agradeço também aos excelentes professores que fazem parte do corpo doscente do Departamento de Engenharia Elétrica, que sempre se disponibilizaram, mesmo em horários alternativos, para tirar todas minhas dúvidas e me ajudaram em vários projetos durante a graduação.

Aos demais professores, colegas e funcionários da Universidade de Brasília, principalmente da Faculdade de Tecnologia, sem os quais este trabalho não poderia ser realizado.

Resumo

PointClouds, ou nuvem de pontos, são um tipo de representação de uma cena tridimensional na qual cada ponto no espaço é definido por uma representação matricial de sua coordenada no eixo espacial xyz e sua cor por outra matriz RGB. Este tipo de arquitetura é extremamente interessante pela fácil manipulação. Porém, inevitavelmente, quanto mais detalhado o modelo tridimensional, maior o número de elementos no espaço e, consequentemente, maior o tamanho da informação necessária para descrever a PointCloud. Por este fato, se faz necessária a utilização de algoritmos de compressão para que seja possível utilizar menor espaço para o armazenamento ou transferência da informação. Este trabalho tem como objetivo, então, propor uma maneira de comprimir a geometria de PointClouds de forma eficaz e sem perdas a partir da decomposição do modelo tridimensional em cortes transversais e posterior compressão utilizando algoritmos já consolidados, como o JBIG, e manipulações booleanas. Em seguida, é realizada uma comparação entre o método proposto e métodos utilizados atualmente no mercado e os caracterizados como estado da arte. Os resultados são apresentados através de análises estatísticas da aplicação do algoritmo desenvolvido em uma grande base de dados de PointClouds, podendo representar de forma concisa os pontos positivos e negativos do projeto. O conteúdo pós análise consiste de reflexões a respeito dos prós e contras do método desenvolvido e propostas de futuro desenvolvimento.

Palavras-chaves: Compressão; PointClouds; Octrees;

Abstract

PointClouds are a type of three-dimensional representation of a scenery in which each volume element is described in the form of a three column matrix filled by each coordinate (x,y and z) value and its color by another matrix composed by its RGB values. This type of architecture is extremely interesting because of the easy manipulation it provides. However, inevitably, the more detail the PointCloud provides, the bigger the size of the information required to describe the PointCLoud. Due to this fact, it is necessary to use compressing algorithms to make these geometries easier to transfer or to store. This paper's objective is to provide a reliable way to compress PointClouds' geometries effectively and without losses by decomposing the model in its cross-sections and using existing binary image compression algorithms, such as JBIG, and Boolean manipulations to compress them. In addition, a comparison between algorithms currently used in the market and the state of art and the project is then made. The results are presented through a statistical analysis of the proposed method being applied in a large PointCloud data base, providing information about the positive and negative sides of the developed algorithm. The post analysis content is made of reflections about the pros and cons of the method and a series of ideas for future development.

Key-words: Compression; PointClouds; Octrees;

Sumário

Lista de ilustrações

1 Introdução

Este capítulo introduz o contexto histórico que trouxe o tópico deste trabalho à tona além dos principais motivadores, objetivos e expectativas a respeito do projeto desenvolvido.

1.1 Contextualização

A tecnologia vem se aprimorando de forma exponencial ao longo das últimas décadas, a cada novo dispositivo desenvolvido mais fácil se torna o projeto de um mais avançado. Com isso, recursos primários de tecnologia como espaço de armazenamento e processamento são cada vez mais requisitados. Um exemplo simples dessa progressão é a criação de melhores elementos de armazenamento. Por exemplo, em 1956 o primeiro disco rígido foi criado pela IBM, o chamado RAMAC (*Random Access Method for Accounting and Control*) era do tamanho de duas geladeiras e tinha capacidade de armazenamento de 5MB [10]. Nos dias atuais, um smartphone possui pelo menos 8GB de armazenamento, mais de 1000 vezes a memória do RAMAC. Porém, isso não significa que hoje conseguimos armazenar 1000 vezes mais arquivos, pois os arquivos utilizados atualmente também ocupam mais espaço, mais do que isso, o espaço ocupado por nossos vídeos, imagens ou programas cresce mais rápido do que o avanço da tecnologia de armazenamento. Por esse motivo, dentre outros, novas formas de aumentar a capacidade de armazenamento foram sendo desenvolvidas ao longo dos anos, dando origem à área de compressão de sinais.

Comprimir um sinal sem perda de informação garante um maior espaço efetivo de armazenamento sem que seja necessário maior espaço físico para armazenamento, imaginemos um arquivo que ocupe 2MB, se conseguirmos comprimí-lo para apenas 1MB conseguiremos armazenar dois arquivos em um disco de 2MB, dobrando então sua capacidade efetiva de armazenamento.

Além disso, com a grande expansão dos meios de comunicação, o envio de informação também se tornou muito mais sofisticado e elevou a necessidade do desenvolvimento de formas de compressão da mensagem para que essa pudesse ser transmitida no menor tempo possível e ocupando a menor banda possível.

O ramo da teoria de informação, que engloba também a compressão de sinais, surgiu

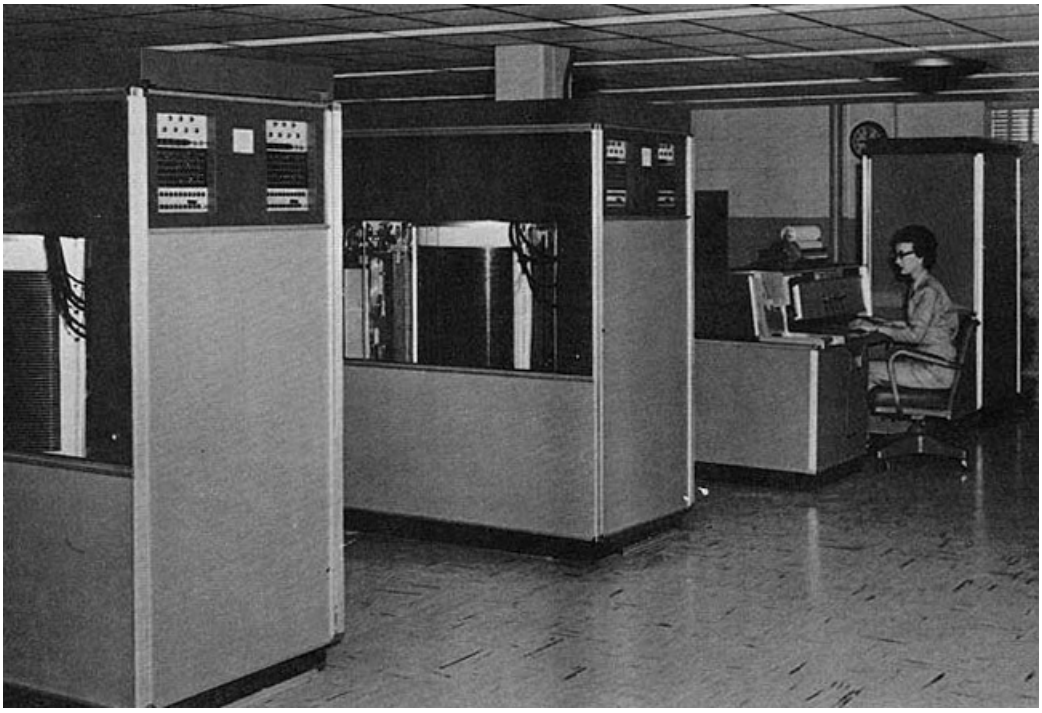

Figura 1 – IBM 305 RAMAC [4].

dos trabalhos pioneiros do engenheiro eletrônico, matématico e criptógrafo Claude Shannon, que contribuiu muito para a criação de formas de se guardar a mesma quantidade de informação em um número muito menor de bits. Shannon é conhecido como o pai da teoria da informação e seu trabalho *A Mathematical Theory of Communication*(Uma Teoria Matemática da Comunicação) [22] foi o deste ramo. Atualmente, pesquisadores que trabalham com compressão de sinais buscam formas de comprimir arquivos, com ou sem perdas, a depender da aplicação, da forma mais eficiente possível.

1.2 Motivação

As imagens 3D são um tipo de tecnologia extremamente interessante pois a visualização tridimensional de um objeto oferece uma gama de vantagens em diversos ramos, como por exemplo modelagem de produtos, visualização de imagens médicas, ou a própria criação de objetos para a impressão 3D. Para esses tipos de aplicação, as nuvens de pontos são uma opção bastante interessante pela facilidade de manipulação dos dados que elas fornecem, afinal, tratam-se de matrizes.

Mesmo com o desenvolvimento de diversas técnicas e o constante aprimoramento dos dispositivos atuais, a falta de espaço de armazenamento e a busca por maior eficiência na

transmissão de mensagens continuam sendo fatores críticos para o desenvolvimento de novas tecnologias. Na área médica, por exemplo, como detalhes são extremamente importantes nas imagens, as resoluções são muito altas e consequentemente o tamanho destes arquivos também são grandes.

A motivação deste trabalho surge da necessidade de se desenvolver métodos mais eficientes de compressão, aliado às vantagens da utilização de nuvens de pontos para representar modelos tridimensionais. O propósito principal deste projeto, então, é desenvolver um algoritmo com o objetivo de comprimir de forma mais eficaz um tipo específico de arquivo.

1.3 Definição do Problema

Neste trabalho, será exposto um método que busca comprimir geometrias tridimensionais chamadas *PointClouds*,ou, nuvens de pontos, sem perda de informação. Para tanto, serão utilizados conceitos novos, próprios deste trabalho, bem como algoritmos já desenvolvidos e utilizados no mercado. Tentar-se-á superar os programas atualmente comercializados ou equiparar-se aos modelos que compõem o atual estado da arte.

O método proposto será baseado na linguagem de programação própria da plataforma MATLAB. O MATLAB é um ambiente de computação numérica que permite a implementação de programas e possui como principal característica o cálculo matricial, o que se torna bastante conveniente para este trabalho pois as nuvens de pontos são compostas justamente de matrizes, como será definido posteriormente.

1.4 Expectativas

A principal expectativa à respeito deste trabalho é o sucesso no desenvolvimento de um método inovador e eficiente para a compressão sem perdas da geometria das nuvens de pontos igualando-se ou superando os métodos hoje comercializados e o estado da arte. Caso não seja possível atingir esse principal objetivo, espera-se que os resultados aqui demonstrados sirvam de base para estudos futuros que possam partir do princípio fundamental deste trabalho, ou de idéias comentadas no Capítulo 5, que tratará a respeito de futuros desenvolvimentos.

2 Revisão Bibliográfica

2.1 Nuvens de pontos

Nuvens de pontos são uma espécie de geometria tridimensional composta por pontos no espaço que são caracterizados através de sua coordenada no plano cartesiano na forma [x, y, z]. Uma nuvem de pontos pode definir o formato de um objeto no espaço através das coordenadas de cada ponto da superfície deste objeto. A unidade de volume adotada em nuvem de pontos é chamada de *voxel*, que vem de *volume element* assim como a unidade de área de imagens é chamada de *pixel*, de *picture element*. A obtenção desse tipo de composição tridimensional pode ser feita a partir de scanners que emitem raios de luz que são refletidos pelo objeto e recapturados pelo scanner, com isso, o dispositivo é capaz de medir o tempo que o raio levou para ser refletido e determinar a posição do objeto no espaço.

Os *voxels* de uma nuvem de pontos localizam-se, na maior parte das vezes, na parte externa do objeto, pois esses pontos são onde os raios são refletidos, dessa forma, a composição tridimensional sempre representa apenas a casca do objeto, ou seja, a parte externa apenas. Uma grande dificuldade presente ao se trabalhar com nuvem de pontos é justamente a sua obtenção, para boas resoluções os pontos são muito próximos uns dos outros e isso exige um scanner de alta qualidade.

O conceito de nuvem de pontos para representação de modelos tridimensionais ganhou muito espaço no mundo acadêmico principalmente após o desenvolvimento do famoso Kinect pela Microsoft, mas acabou perdendo espaço para outras tecnologias como por exemplo o 123D Catch, da Autodesk, como comenta Sam Billingsley Jr. em sua publicação [13]. Porém, a evolução de tecnologias de reconhecimento facial e de movimentos trouxe o tópico de nuvem de pontos de volta à tona, o método de escaneamento e produção de informação das nuvens de pontos se mostrou bastante conveniente para este tipo de implementação. Como conclui Lewis Boxer em [3], o futuro da nuvem de pontos não está exatamente na produção de dados, mas sim na sua documentação, interpretação e manipulação.

Em termos de computação as nuvens de pontos são uma forma muito interessante de trabalhar imagens tridimensionais pois os pontos são apenas a combinação de três valores, fáceis de serem manipulados pelo computador. É possível ainda combinar esses valores com

uma outra matriz RGB, dando então cor a cada *voxel* e dessa forma garantindo que a nuvem de pontos seja colorida. Em conclusão, Nuvens de pontos são uma forma simples de se trabalhar imagens tridimensionais, porém essa simplicidade vem ao custo de uma díficil produção, exigindo equipamentos de alto valor como um scanner de qualidade.

Figura 2 – Exemplo de nuvem de pontos [1].

2.2 Compressão de Sinais

Como explicitado no capítulo de introdução, o ramo de compressão de sinais se aprimorou significativamente devido à necessidade de se armazenar arquivos maiores sem que seja necessário um espaço físico maior para tal. Porém, os conceitos de compressão remetem a tempos anteriores à teoria da informação e aos computadores, como é o caso do conhecido código criado por Samuel F. B. Morse, o *código Morse* [12], largamente utilizado pelos meios de comunicação da época, os telégrafos. O código Morse também funcionara como um tipo de criptografia, afinal, apenas um receptor que conhecesse a tabela de conversão dos códigos seria capaz de decifrar a mensagem, tal aplicação foi muito explorada durante a primeira guerra mundial.

Para que seja possível o entendimento do trabalho aqui apresentando, alguns conceitos principais precisam ser explicitados. Primeiramente, deve-se haver uma clareza a respeito dos trabalhos pioneiros desenvolvidos por Claude E. Shannon [22], onde o autor demonstra os principais fundamentos por trás da teoria da informação e da compressão de sinais, e por David A. Huffmann, conhecida como codificação de Huffman [9], esta metodologia permite a redução do tamanho ocupado por cada símbolo a partir da frequência em que o mesmo aparece na mensagem a ser enviada, bem como os principais conceitos por trás da codificação aritmética, uma outra abordagem que pertence, assim como a codificação Huffman, ao grupo de codificações

de entropia, um termo genérico utilizado para classificar metodologias de compressão que não levam em consideração a natureza da informação a ser comprimida, ou seja, este tipo de método ignora a semântica da informação a ser comprimida. A fim de facilitar o entendimento do leitor a respeito das terminologias utilizadas neste trabalho, apresenta-se um pequeno glossário com palavras comumente utilizadas e seus respectivos significados na área.

Tabela 1 – Glossário

Fonte	Gerador ou origem dos dados
Símbolo	Unidade de informação ou dado
Alfabeto	Conjunto de símbolos que a Fonte pode gerar
Mensagem	Sequência de símbolos
Código	Sequência binária que representa um símbolo
Dicionário	Conjunto de códigos e seus respectivos símbolos
Redundância	Informação repetida
Irrelevância	Informação desnecessária cuja remoção não ocasiona nenhuma perda
Taxa	Relação entre número de bits utilizados e símbolos (bit/simb)
Contexto	Relação probabilística entre símbolos
Bitstream	Fluxo de bits

2.2.1 Entropia

Um dos conceitos primordiais na área de compressão de sinais é a entropia. Ela define uma medida de imprevisibilidade de uma fonte. Imaginemos uma fonte de informação que emite constantemente símbolos, formando uma mensagem. A entropia define quão prevísivel, ou aleatório, é o próximo símbolo a ser enviado pela fonte. Por exemplo, se 1000 bits representam 1000 lances de moeda, a entropia desta mensagem será de 1 bit por símbolo, pois não há como prever se o lance cairá em cara ou coroa, em termos de compressão de sinais, não há como comprimir a mensagem sendo gerada por lances de moeda, sua entropia será de 1 bit por símbolo (1 ou 0 para cara ou corora).

Então como é possível comprimir sinais? Basicamente o que é feito é retirar redundâncias na mensagem, a maioria dos sinais de informação possuem uma congruência, o que significa que os símbolos não são completamente aleatórios, eles possuem alguma correlação, e é a partir desta correlação que surgem as redundâncias. As próprias palavras são um simples exemplo de como existe esta redundância. Se as letras são os símbolos, após uma sequência do tipo a-b-a-c-a-x é extremamente provável que a próxima letra será a letra i. Portanto, para que gastar bits se já é sabido qual será a última letra? Este conceito é utilizado para se realizar compressão.

A definição de entropia é igual ao nível de desordem de uma mensagem, e pode ser elaborada matemáticamente por:

$$H = -\sum p_{\mathrm{i}} I(p_{\mathrm{i}}) \tag{2.1}$$

Em que,

H = entropia;

$\mathrm{p_i}$ = probabilidade de ocorrer o símbolo i;

I = autoinformação do evento (redundância);

Que nos leva a:

$$H = -\sum p_{\mathrm{i}} \log_{\mathrm{b}}(p_{\mathrm{i}}) \tag{2.2}$$

A base b do logaritmo define a unidade de informação, como neste caso trabalhamos com fontes binárias, a base do logaritmo será sempre 2, cuja unidade são bits. À medida em que a probabilidade de ocorrência de um símbolo aumenta, ou seja, à medida que sua previsibilidade aumenta, menor será a entropia desta fonte, pois maior será sua redundância. Em uma fonte de apenas 2 símbolos, a maior entropia, 1, será quando a probabilidade de ocorrência dos símbolos

for 0,5 pois é nesse valor que os símbolos são completamente imprevisíveis, como exemplificando em um lance de moedas. Se a probabilidade de ocorrência de um dos símbolos for maior que 0,5, então pode ser projetado um código que, na média, utiliza menos bits para representar a mensagem como um todo.

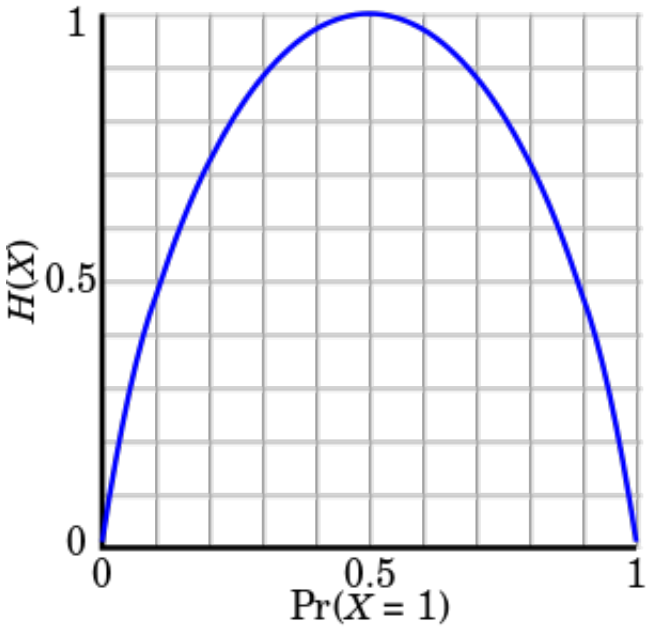

Figura 3 – Entropia de uma fonte de 2 símbolos.

2.2.2 Codificação Huffman

Durante seu período na faculdade, Huffman cursava a disciplina de teoria da informação quando recebeu de seu professor uma tarefa de encontrar a codificação binária mais eficiente para determinada sequência de símbolos. Com a idéia de utilizar uma árvore binária com a frequência de cada símbolo, Huffman conseguiu produzir um código sem perdas mais eficiente que seu professor, que trabalhava com o próprio pioneiro da área Claude Shannon.

Imaginemos a seguinte frase: "bombom e mesmo bom", temos exatamente 6 diferentes símbolos, portanto, para representarmos todos os símbolos em forma binária sem que haja erro de interpretação, precisaríamos de no mínimo 3 bits, podendo ser distribuídos da forma apresentada na Tabela 2.

Dessa forma, a mensagem poderia ser codificada como:

000 011 010 000 011 010 101 001 101 010 001 100 010 011 101 000 011 010

Tabela 2 – Símbolos Huffman

Símbolo	Código
"b"	000
"e"	001
"m"	010
"o"	011
"s"	100
" "	101

utilizando um total de 54 bits para ser enviada. Porém, segundo a teoria de Huffman [9], há uma maneira de representar a mesma mensagem sem que haja erro de interpretação em um número menor de bits se utilizarmos menos bits para os símbolos que mais aparecem e mais bits para símbolos que aparecem menos. Pode-se então contar o número de vezes que cada símbolo aparece na mensagem e calcular a probabilidade de cada símbolo, a partir da divisão do número de ocorrências pelo número total de símbolos enviados. Estes valores estão presentes na Tabela 3.

Tabela 3 – Símbolos Huffman

Símbolo	Aparições	Probabilidade(%)
"b"	3	17
"e"	2	11
"m"	5	27
"o"	4	22
"s"	1	6
" "	3	17

A árvore binária começa a ser construída pegando os símbolos que menos aparecem e unindo-os em um só nó, o qual terá número de aparição equivalente à soma dos dois símbolos, e atribui-se o bit 0 para o símbolo com maior aparição e o bit 1 para o com menor aparição. Em seguida, encontra-se os dois símbolos ou nós de menor aparição e repete-se o processo de junção, novamente atribuindo os bits 0 e 1 até que reste apenas um símbolo ou nó. Após a construção da árvore, pode-se realizar a conversão da mensagem para o código resultante. A Figura 4 a seguir demonstra o processo de construção da árvore binária dos símbolos utilizados na mensagem.

Com a Tabela 4 podemos codificar a mensagem sem perda de informações como sendo

11 10 00 11 10 00 010 0110 00 0110 0111 00 10 010 11 10 00

Tabela 4 – Códigos em Huffman

Símbolo	Código
"b"	11
"e"	0110
"m"	00
"o"	10
"s"	0111
" "	010

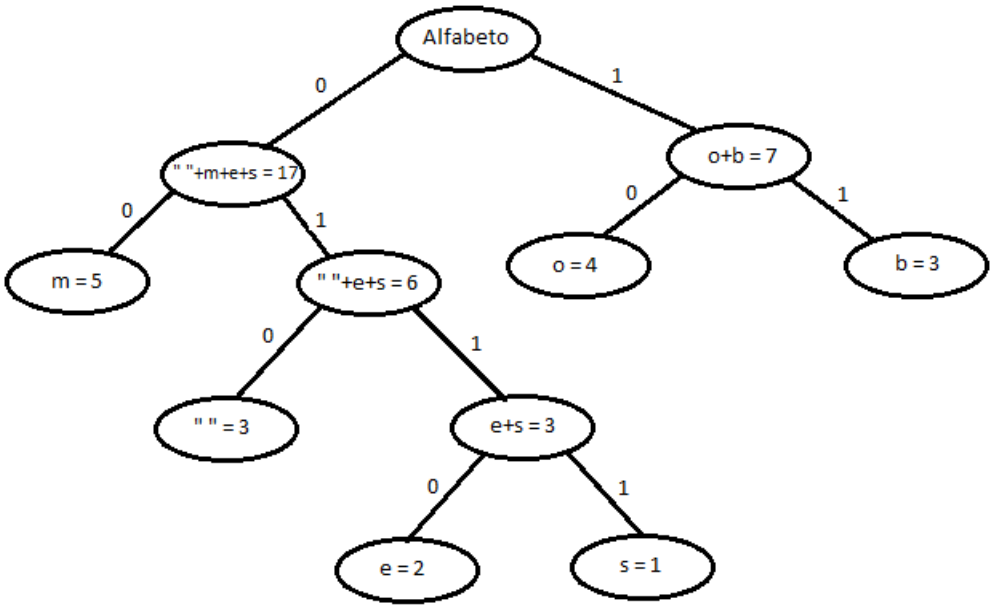

Figura 4 – Árvore binária de Huffman.

totalizando então 43 bits, valor aproximadamente 20,3 % menor que a codificação sem utilização de nenhum método de compressão. Essa diminuição do tamanho total ocupado em cerca de um quinto se deve ao simples fatos de utilizarmos menos bits para símbolos que aparecem recorrentemente e mais bits para símbolos que ocorrem menos. Um ponto negativo desta metodologia, entretanto, é a necessidade de se conhecer todas as probabilidades de cada símbolo possível na mensagem antes de se realizar a compressão, afinal, a eficiência da compressão se dá a partir da noção prévia estatística de aparição de cada símbolo.

Pode ser feita uma comparação entre a eficiência de compressão utilizando o método proposto por [9] e a entropia da fonte, o valor mínimo em bits por símbolo que esta mensagem é capaz de ser comprimida, calculada a partir das probabilidades de cada símbolo, apresentada

na Tabela 3. O calculo deste valor é apresentado em 2.3.

$$\begin{aligned} H &= -\sum p_{\mathrm{i}} log_{\mathrm{b}}(p_{\mathrm{i}}) \\ H &= -3/18 * \log_2(3/18) \\ &- 2/18 * \log_2(2/18) \\ &- 5/18 * \log_2(5/18) \\ &- 4/18 * \log_2(4/18) \\ &- 1/18 * \log_2(1/18) \\ &- 3/18 * \log_2(3/18) \\ &= 2,37 \end{aligned} \tag{2.3}$$

A entropia da mensagem emitida é de 2,37 bits por símbolo, enquanto que o código de Huffmann gerado possui uma taxa de $43/18 = 2,38$ bits por símbolo, resultado muito próximo da entropia. O exemplo citado é de uma mensagem extremamente pequena com um número limitado de símbolos e uma probabilidade bem definida, na maioria dos casos reais isto não acontece, portanto, nem sempre as taxas serão tão próximas da entropia. Vale ressaltar, ainda, que o código formado é apenas de primeiro nível, ou seja, ele não considera possíveis relações entre símbolos, o mesmo pode ser dito a respeito da entropia calculada. A real entropia de um código deve ser calculada para toda e qualquer combinação de símbolos, desde todos os símbolos independentes, como de todos os símbolos tomados 2 a 2 até n a n, sendo n o número total de símbolos na mensagem.

Com os resultados apresentados, pode-se notar, que comprimir sinais é um método extremamente valioso para que seja possível armazenar uma quantidade maior de informação. Para que o decodificador seja capaz de recriar a mensagem original, basta que o mesmo tenha em mãos a tabela de conversão representada na Tabela 4, que pode ser tanto transmitida antes da mensagem, na forma de *header* quanto definida previamente.

2.2.3 Codificação Aritmética

Após os trabalhos pioneiros de Shannon e Huffman, diversas outras técnicas foram desenvolvidas para otimizar a compressão de sinais. Uma metodologia especificamente importante para este trabalho é a codificação aritmética.

Imaginemos uma fonte de informação que emite constantemente um fluxo de símbolos, na codificação Aritmética, parte-se de um intervalo ([0,1[) e subdivide-se este intervalo em N partições proporcionais a sua probabilidade, sendo N o número de símbolos presentes no alfabeto. Ao receber o primeiro símbolo, o codificador altera seu intervalo de ([0,1[) para o intervalo que representa este símbolo, em seguida, o novo intervalo é novamente fracionado em N partes, da mesma forma como foi feito na primeira vez. O segundo símbolo recebido pelo codificador alterará novamente o intervalo e suas subdivisões, formando um processo iterativo em que, a cada iteração, menor é o intervalo que representa cada símbolo, portanto, quanto mais símbolos necessita-se codificar, maior o número de bits necessário para representar todas as casas decimais necessárias para dar precisão ao intervalo. Para a decodificação, com o intervalo final em mãos, basta encontrar quais os subintervalos anteriores que foram utilizados para que se pudesse chegar ao número final. Como as subdivisões não se intersectam, existe apenas uma possível combinação de subintervalos capaz de contemplar o valor binário final [20].

Um dos principais pontos negativos da codificação aritmética é a elevada precisão matemática necessária para transmitir mensagem maiores. Por este fato, existem diversas propostas de metodologias derivadas da codificação aritmética que buscam resolver este problema.

Tomemos a pequena mensagem "ARBER"como exemplo, temos um total de 4 possíveis símbolos em que suas probabilidades são dadas na Tabela 5.

Tabela 5 – Símbolos Huffman

Símbolo	Probabilidade(%)
"A"	20
"B"	20
"E"	20
"R"	40

Os intervalos iniciais para cada símbolo seria a divisão do intervalo (]0,1[) em 4 partes iguais, ou seja, A[0 – 0.2), B[0.2 – 0.4), E[0.4 – 0.6), R[0.6 – 1). À medida que o codificador intepreta qual o símbolo gerado pela fonte, ele adotará como novo intervalo total o intervalo correspondente ao símbolo, e dividirá novamente este intervalo em 4 partes iguais. Como na mensagem de exemplo o primeiro símbolo é A, teremos que subdividir o intervalo [0 – 0.2) em 4 partes iguais, sendo então: A[0 – 0.05), B[0.05 – 0.1), E[0.1 – 0.15), R[0.15 – 0.2). O processo completo de codificação é demonstrado na Figura 5, gerando um valor final para toda a mensagem dentro do intervalo 0,14432 a 0,1456. Ou seja, basta enviar qualquer número dentro deste intervalo que o decodificador conseguirá recuperar a mensagem original, basta

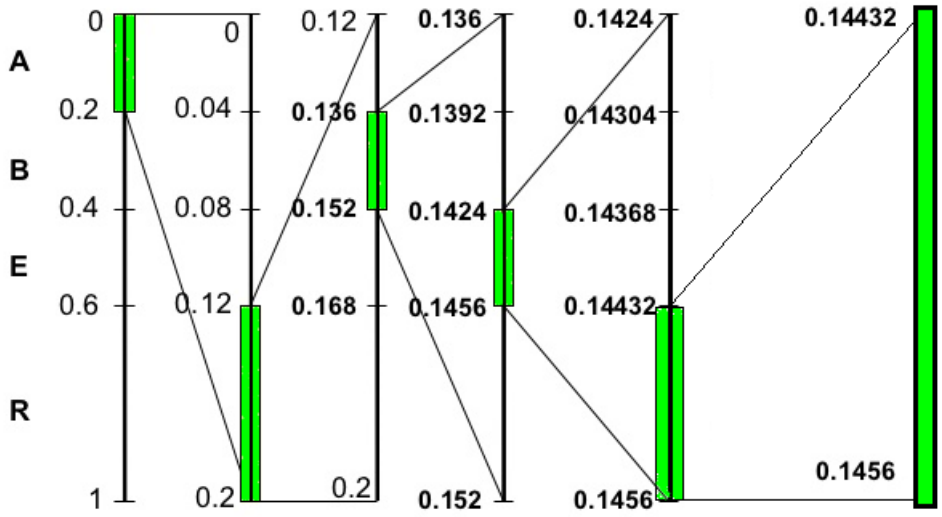

Figura 5 – Exemplo de processo de codificação aritmética [2].

este saber de antemão todos os símbolos e a estatística da mensagem, que podem ser enviados anteriormente na forma de *header*.

2.2.4 Compressão de Imagens

As codificações Aritmética e de Huffman são tipos de codificação de entropia, ou seja, não existe discernimento do tipo de mensagem sendo comprimida. Porém, à medida que criamos métodos de compressão mais específicos, somos capazes de obter taxas melhores por conseguirmos aproveitar características específicas do tipo de mensagem sendo comprimida. Este é o caso de métodos de compressão de imagem, que utilizam de diversos artifícios para obter uma melhor compressão. Alguns métodos de compressão de imagem se tornaram extremamente comuns no uso cotidiano de usuários de computadores, sendo alguns deles: JPEG, PGF e JPEG2000. O projeto realizado neste trabalho utilizará de métodos de compressão de imagem já bem consolidados na comunidade acadêmica para buscar uma taxa de compressão mais interessante. Os métodos utilizados serão abordados brevemente nesta introdução.

As imagens a serem trabalhadas neste projeto são imagens binárias, o que significa que seus *pixels* podem assumir apenas dois valores, 0 ou 1. Imagens binárias são, por convenção, imagens preto e brancas, e sua taxa sem compressão é igual a 1 bit por pixel. Para seu processamento digital, as imagens são interpretadas na forma de matriz, em que os elementos α_{ij} correspondem aos pixels da imagem e seu valor corresponde à cor. Em imagens coloridas, são

necessárias três matrizes para compor a cor desejada, as cores mais comumente utilizadas são variações de vermelho, verde e azul, daí o termo RGB (Red Green Blue).

Já foi discutido neste capítulo o conceito de contexto e como métodos de compressão específicos para um tipo de arquivo podem aproveitar de maneira muito eficiente esta relação entre símbolos. O contexto em imagens é muito aproveitado por diversas metodologias. Como as imagens retratam alguma gravura, a probabilidade de um pixel ser de mesma coloração que seu vizinho é relativamente alta. A Figura 6 torna fácil a visualização do contexto em uma imagem, os *pixels* pretos e brancos ocupam espaços onde a maioria de seus vizinhos são de mesmo valor.

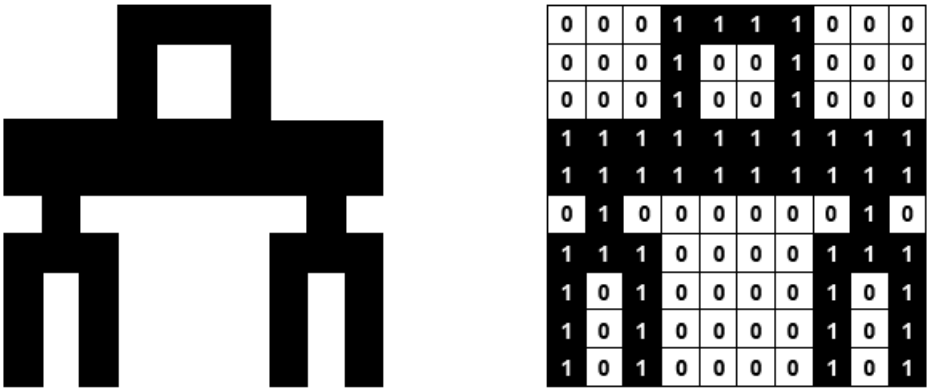

Figura 6 – Exemplo de imagem binária.

2.2.4.1 JBIG

O algoritmo de compressão de imagens JBIG foi primeiramente publicado como uma alternativa ao método padronizado G4 para imagens binárias [11], em seguida o método JBIG2 foi publicado, porém, o núcleo de ambos padrões permaneceu o mesmo. O JBIG utiliza uma modelagem estatística e codificação aritmética como orginalmente proposto em [14]. A eficiência de compressão de imagens binárias do JBIG é muito superior aos algoritmos de compressão genéricos como JPEG e PNG, por exemplo, como pode ser observado nos trabalhos realizados em [7]. Partindo de uma varredura de todos os pixels chamada de *raster scan*, da esquerda para direta e de cima para baixo, o JBIG estima uma probabilidade do próximo pixel baseada nos pixels já lidos anteriormente, o contexto é definido como uma combinação dos pixels já processados, além disso, o contexto é reavaliado a cada leitura de pixel, ou seja, o algoritmo é adaptativo ao contexto.

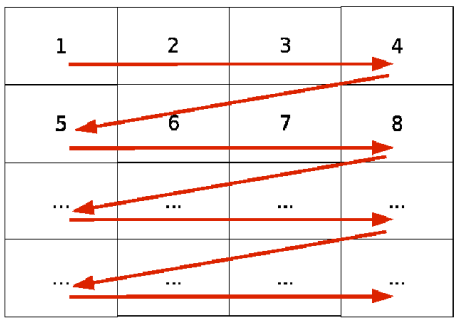

Figura 7 – Raster Scan.

No algoritmo original do JBIG, a probabilidade de um pixel ser 1 ou 0 é calculada a partir de 10 bits de contexto, a partir da frequência de bits 1's e 0's na vizinhança. Essa probabilidade é estimada adaptativamente, ou seja, a cada novo pixel codificado a tabela de frequência e probabilidade é atualizada. O pseudo-código para o algoritmo do JBIG pode ser escrito da seguinte forma:

Algorithm 1 Pseudocódigo JBIG

```
for Cada pixel na matriz Imagem do
    Encontra o valor de cada bit do contexto,
    Calcula a probabilidade do bit ser 1 ou 0, dado a vizinhança,
    Codifica o bit atual usando codificação aritmética,
    Atualiza a tabela de frequências,
end for
```

	1	2	3	
4	5	6	7	8
9	10	**X**		

Figura 8 – Contexto utilizado no JBIG [6].

2.2.5 Octree

Octrees são formas de representação de imagens tridimensionais em termos de *bytes* na qual o espaço, comumente um cubo, é subdividido em 8 menores cubos em que, caso o cubo esteja ocupado, possui valor binário 1 e caso esteja vazio possui o valor binário 0. Uma forma mais simples de se compreender os conceitos por trás de Octrees é iniciar pelo modelo bidimensional, as chamadas Quadtrees são o equivalente em 2D. Dessa forma, parte-se de um

grande quadrado que possui o tamanho da imagem completa, divide-o em quatro quadrados de igual tamanho e atribui-se o bit 1 para quadrados ocupados e o bit 0 para quadrados vazios, pode-se então realizar essas subdivisões recursivamente até que se alcance a dimensão unitária do pixel. Todas as subidivisões formarão uma sequência de bits que pode, sem perda de informação, reconstruir a imagem original.

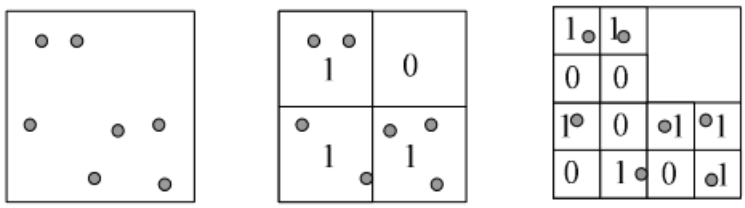

Figura 9 – Exemplo de codificação do tipo quadtree [18].

A grande vantagem deste tipo de arquitetura é que grandes espaços vazios são codificados com apenas a utilização de um bit, como pode ser visto no quadrante direto superior da segunda Quadtree na Imagem 9. As octrees possuem a mesma vantagem, portanto, dados esparsos são muito bem comprimidos. A Imagem 11 demonstra um claro exemplo de como as octrees são codificadas e como o *bitstream* gerado é suficiente para recuperar toda a informação.

Octrees são um método muito utilizado para codificação de nuvens de pontos [21] porque além de garantir uma taxa de compressão interessante quando comparada a outros métodos de compressão de nuvens de pontos [16], são também capazes de modificar a resolução da imagem 3D a depender da taxa desejada como pode ser observado na imagem 10 de [18].

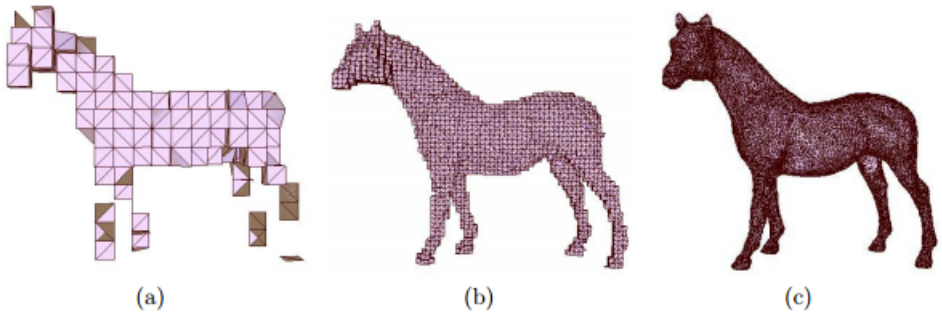

Figura 10 – Restauração da imagem a taxa de: (a) 0.05 bpv, (b) 0.62 bpv, (c) 5.91 bpv[18].

Como a cada subdivisão dos cubos em 8 cubos menores um novo byte é necessário, quanto maior a resolução, ou seja, menores os cubos, maior o número de bytes necessários para enviar a imagem. O modelo de octree permite que seja preestabelecido uma dimensão mínima dos cubos, diminuindo a resolução porém diminuindo a taxa de transmissão também. Essa

escalabilidade é um grande diferencial da metodologia de octrees principalmente no que diz respeito à renderização de espaços tridimensionais, pode ser feita uma renderização dinâmica na qual à medida que se aproxima de um objeto, mais bem definido ele se torna.

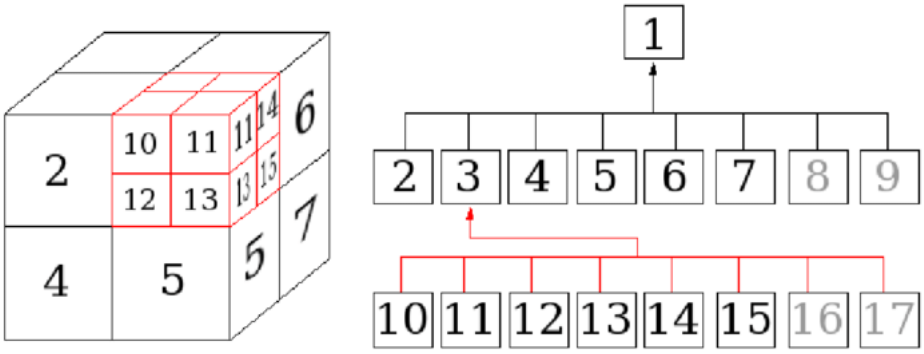

Figura 11 – Metodologia octree [5].

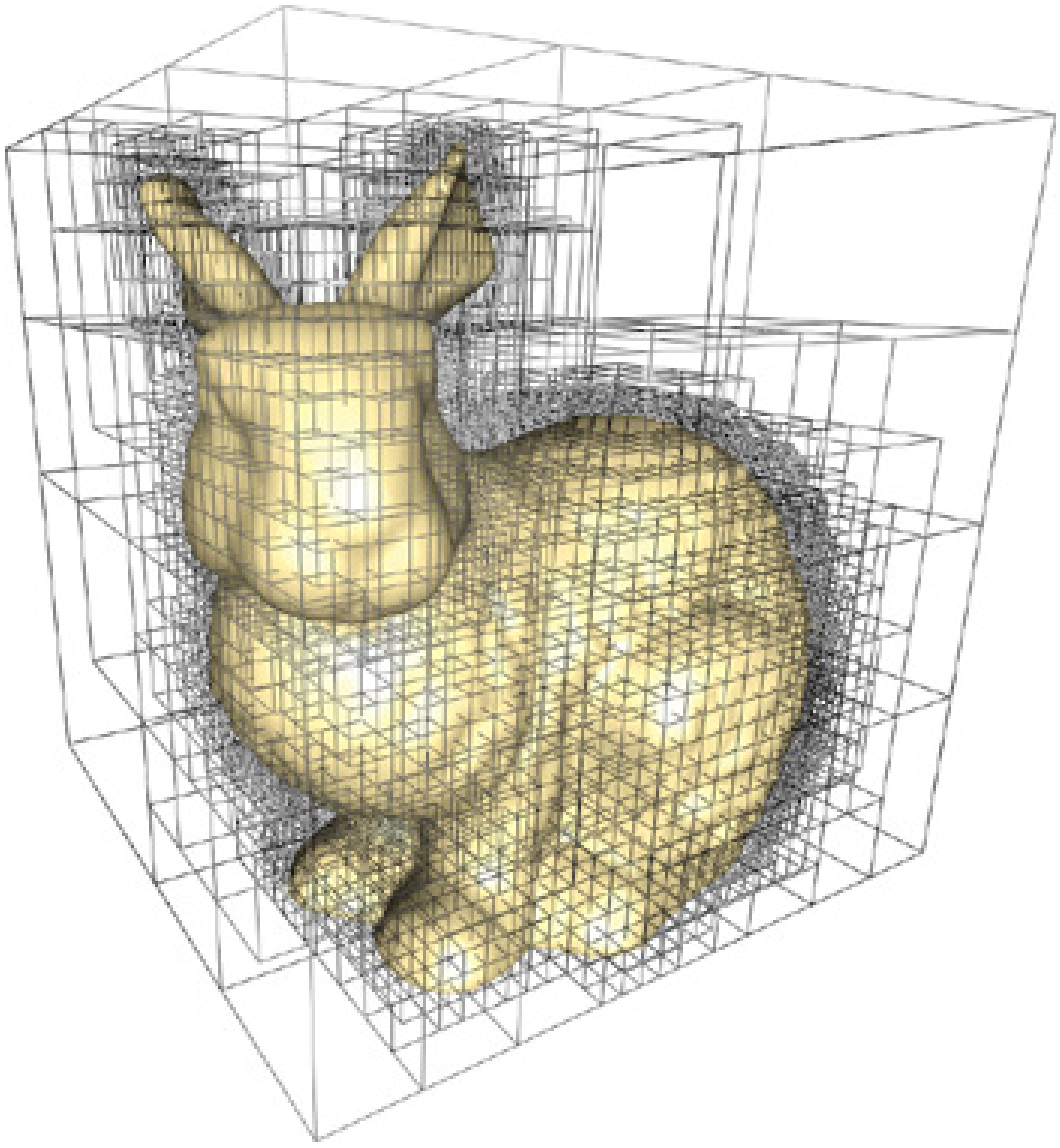

Figura 12 – Exemplo de codificação Octree [15].

3 Método Proposto

Esta seção abordará o desenvolvimento do método de compressão de nuvens de pontos proposto e seus principais fundamentos. Em seguida serão apresentados resultados e comparações com outros métodos presentes na literatura.

O método proposto tem como objetivo a compressão apenas da geometria da nuvem de pontos, em outras palavras, o foco deste trabalho é a compressão sem perda de informação do modelo tridimensional em que os voxels assumem apenas dois valores, 1 - ocupado e 0 - vazio. Portanto, qualquer informação a respeito de cores deve ser comprimida de maneira independente ou será perdida. O método consiste na subdivisão da nuvem de pontos em cortes transversais ou longitudinais, seguidos de manipulações booleanas em cada par de cortes (1 e 2, 3 e 4...) para que seja preciso enviar apenas a informação essencial para reconstrução de ambos os cortes. Tal informação é então codificada com um codificador de entropia. Cada etapa do processo será abordada de forma detalhada neste capítulo.

3.1 Segregação em cortes

Como explicitado no Capítulo 2, métodos de compressão que fazem uso do contexto são capazes de obter taxas menores que codificadores de entropia puros, e, como o método proposto neste trabalho é de compressão especificamente de nuvens de pontos, este artifício é amplamente utilizado. O algoritmo JBIG garante uma menor taxa ao utilizar os pixels próximos para estimar a probabilidade do pixel a ser decodificado ser 1 ou 0. No nosso caso, tentaremos utilizar os voxels ao redor do voxel em questão para também estimar o seu valor.

Tomemos como exemplo uma nuvem de pontos de 512x512x512 voxels podendo assumir os valores binários de 1 ou 0, sendo então, 0 espaços desocupados e 1 espaços ocupados. Esta nuvem de pontos forma a imagem de uma pessoa, e foi produzida a partir dos trabalhos de [23]. Ela servirá de modelo para uma melhor compreensão da metodologia elaborada.

A partir desta modelo tridimensional, serão realizados diversos cortes longitudinais ou transversais em qualquer eixo. O eixo escolhido será sempre o que resulte em um menor número de cortes, pois este é o eixo que garante mais informações por corte. No caso apresentado, o corte foi realizado no eixo Y. Tais cortes definirão imagens bidimensionais que representam a

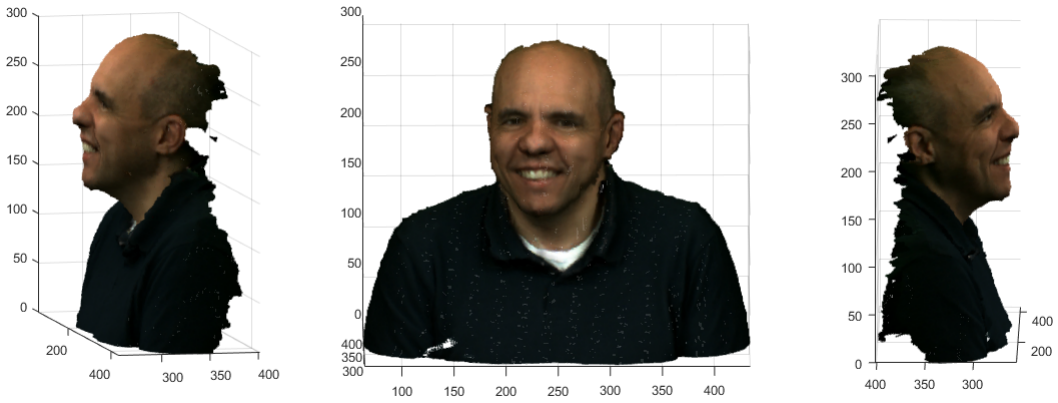

Figura 13 – Exemplo de nuvem de pontos (test sample: Ricardo9).

casca da figura nesta determinada coordenada, como apresentado no exemplo da Figura 14, que demonstra alguns dos cortes realizados na geometria da Figura 13.

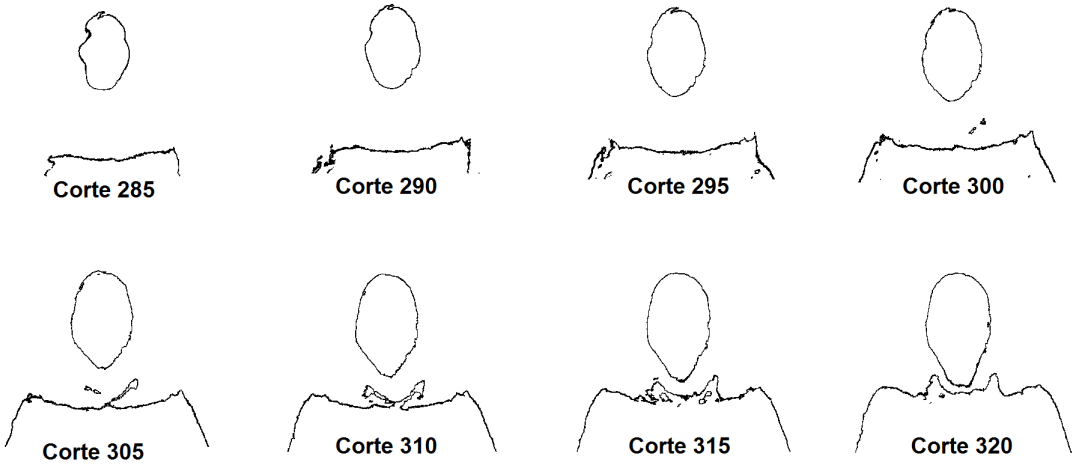

Figura 14 – Exemplo de cortes longitudinais.

No exemplo da Figura 14 utilizou-se de um passo de 5 cortes para que não fossem todos muito parecidos. Ainda assim, como pode ser observado, a semelhança entre os cortes é muito grande, os pixels que compõem cada corte estão quase em posições iguais, separados por uma diferença de uma ou duas unidades de área. Portanto, ao se codificar um corte, não é necessário que o próximo corte seja codificado por inteiro, algumas manipulações podem ser realizadas para que seja necessário codificar apenas a informação essencial para a reconstrução do corte, tendo em vista que o decodificador já terá a sua disposição a informação de todos os cortes anteriores.

3.2 Utilizando o contexto 3D

Tomemos então dois cortes consecutivos A e B. A depender do objeto escaneado, tais cortes apresentarão pixels quase idênticos como exemplificado na Figura 14. Pode ser criado então uma nova imagem Y tal que $Y = A + B$, onde + deve ser compreendido como o operador booleano OU. Esta imagem conterá as informações de A e B, porém, deve-se estabelecer uma maneira de voltar aos cortes iniciais sem perda de informações. Estabelece-se, então, dois novos cortes Y_1 e Y_2, sendo estes compostos pelo operador OU exclusivo, teremos então as seguintes definições:

$$Y = A + B; \tag{3.1}$$

$$Y_1 = A \bigoplus Y; \tag{3.2}$$

$$Y_2 = B \bigoplus Y; \tag{3.3}$$

Estes três novos cortes são o suficiente para que seja possível decodificar a informação e montar A e B novamente, como explicitado na álgebra Booleana em 3.4 Além disso, os cortes Y, Y_1 e Y_2 possuem maior probabilidade de os pixels vizinhos possuírem mesmo valor que o pixel analisado, em outras palavras, possuem um contexto que pode ser melhor aproveitado, resultando em uma menor taxa de compressão se utilizado o JBIG.

$$\begin{aligned}
A &= A \\
&= A(\overline{Y} + Y) \\
&= A\overline{Y} + AY \\
&= A\overline{Y} + Y(\overline{Y}A + YA) \\
&= A\overline{Y} + Y(\overline{A} + Y)(A + \overline{Y}) \\
&= \overline{Y}(A\overline{Y} + \overline{A}Y) + Y(\overline{A} + Y)(A + \overline{Y}) \\
&= \overline{Y}(A\overline{Y} + \overline{A}Y) + Y\overline{[(A\overline{Y}) + (\overline{A}Y)]} \\
&\text{Se } Y_1 = A\overline{Y} + \overline{A}Y = A \oplus Y \\
&\text{Então, } A = \overline{Y}Y_1 + Y\overline{Y_1} = Y \oplus Y_1
\end{aligned} \tag{3.4}$$

A demonstração em 3.4 serve, sem perda de generalidade, para o corte B, Y, e seu derivado Y_2. Portanto, é provado que a partir de Y, Y_1 e Y_2 podemos reconstruir A e B. O exemplo apresentado nas Figuras 15, 16 e 17 demonstra visualmente como os novos cortes possuem pixels mais parecidos com sua vizinhança, ou seja, um contexto mais bem definido.

Imaginemos dois cortes consecutivos de uma nuvem de pontos de 8x8x8 voxels, os cortes foram realizados no eixo y e formam as imagens A e B, de 8x8 pixels cada. Os cortes gerados podem ser manipulados a partir das equações 3.1, 3.2 e 3.3 para então formarem as imagens Y, Y_1 e Y_2.

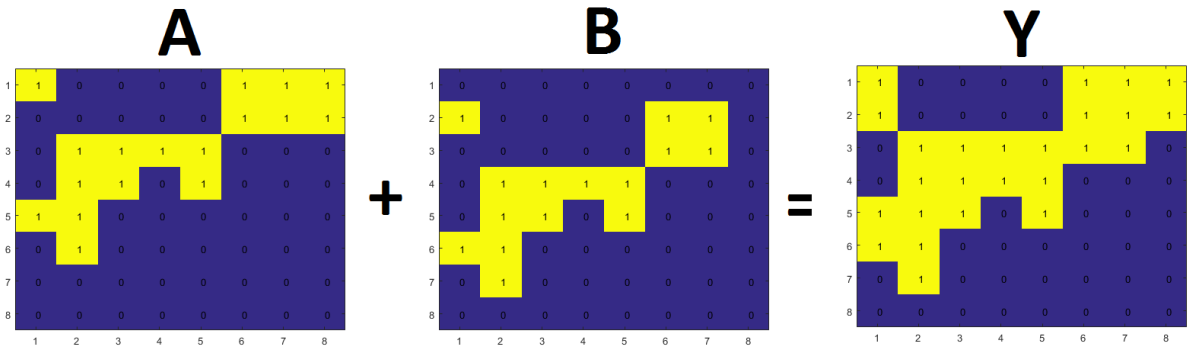

Figura 15 – Primeira operação Booleana.

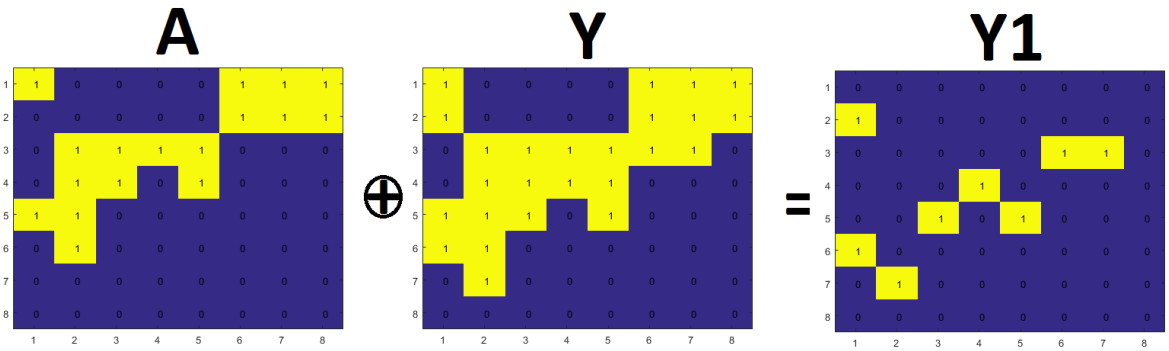

Figura 16 – Segunda operação Booleana.

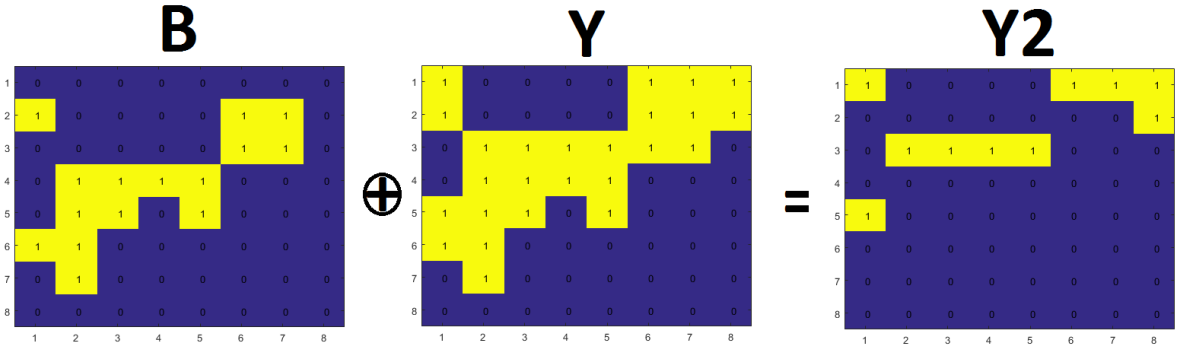

Figura 17 – Terceira operação Booleana.

É interessante notar a predoominância de pixels 1 em Y e de pixels 0 em Y_1 e Y_2, isto se deve pelas características tanto dos operadores OU e XOR, quanto pelas características dos cortes A e B que, por terem sido retirados de uma casca tridimensional, possuem pixels iguais ou muito próximos. Voltando ao exemplo real com a nuvem de pontos "Ricardo9", utilizaremos os cortes de número 295 e 296 para exemplificar a diferença do contexto no momento da compressão via JBIG.

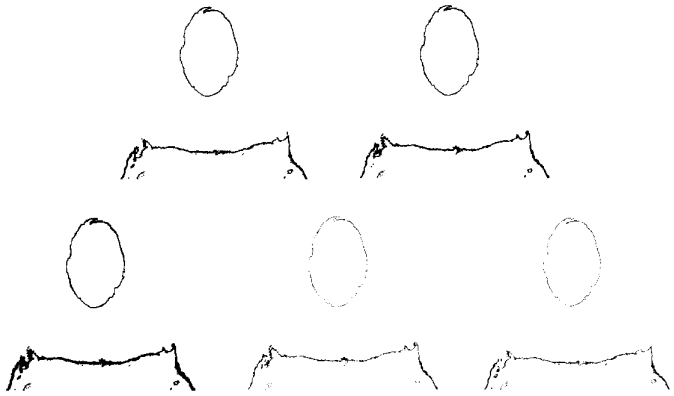

Figura 18 – Cortes 295 e 296 na linha superior e seus derivados Y, Y_1 e Y_2 na linha inferior.

Analisemos então qual o tamanho ocupado pós JBIG de cada um dos cortes, lembrando que o espaço ocupado para enviar cada imagem sem compressão alguma é de 512*512 = 262144 bits, ou seja, uma taxa equivalente a 1 bit/pixel.

Tabela 6 – Bits ocupados para compressão de cada corte (JBIG)

Imagem	Tamanho (bits)	Taxa (bits/pixel)
A	4664	0,0178
B	4624	0,0176
Y	4800	0,0183
Y_1	4464	0,017
Y_2	4384	0,0167

Os cortes provenientes das manipulações Y_1 e Y_2 de fato ocupam menos espaço. Porém, o fato de ser necessário enviar os três cortes torna o tamanho da informação muito maior que apenas comprimir os cortes A e B utilizando o algoritmo JBIG. De fato, existe muita informação repetida nos cortes manipulados, por exemplo, as células vazias são enviadas três vezes, afinal, se o pixel em Y é 0 sabemos de antemão que não há informação naquela célula, nem do corte A nem do corte B, então não há a necessidade de enviar este pixel, como foi feito. A próxima seção introduzirá, então, uma forma mais eficiente de de envio da informação a partir da eliminação deste tipo de redundância.

3.3 Eliminação de Informação Redundante

Este tópico tem como objetivo demonstrar como pode ser realizada a remoção da informação redundante explicitada na seção anterior. Sabemos que se um determinado pixel (x,y) em Y é 0, ele será 0 em ambos os cortes A e B. Portanto, não há a necessidade de enviar este pixel em Y_1 nem em Y_2. A saída escolhida para o projeto consiste em varrer toda a imagem Y no estilo *raster scan*. Se um determinado pixel (x_1,y_1) for 1, incluir em um *bitstream*, chamado de U_1, o bit correspondente a (x_1,y_1) da imagem Y_1. Dessa maneira, enviaremos apenas o *bitstream* que contém os pixels relevantes, como Y será enviado inteiramente, codificado por JBIG, o decodificador será capaz de identificar a localização de todos os pixels relevantes, dessa forma, a leitura e identificação dos pixels no *bitstream* não são uma tarefa difícil.

Para criar o *bitstream* de Y_2, chamado de chamado de U_2, podemos nos desfazer de ainda mais informação redundante. Sabe-se de antemão que, como $Y_1 = Y \oplus A$, todo pixel no qual $Y_1(x,y) = 1$, nos informa que $Y(x,y) = 1$ e $A(x,y) = 0$, e, consequentemente, que B(x,y)

= 1. Então o *bitstream* de Y_2 deverá conter apenas os pixels em que A(x,y) = 1 e B(x,y) = 1, que são os pixels que Y_1(x,y) = 0. O processo de obtenção dos *bitstreams* de Y_1 e Y_2 poderá ser melhor compreendido a partir do exemplo utilizando as já criadas Figuras 15, 16 e 17.

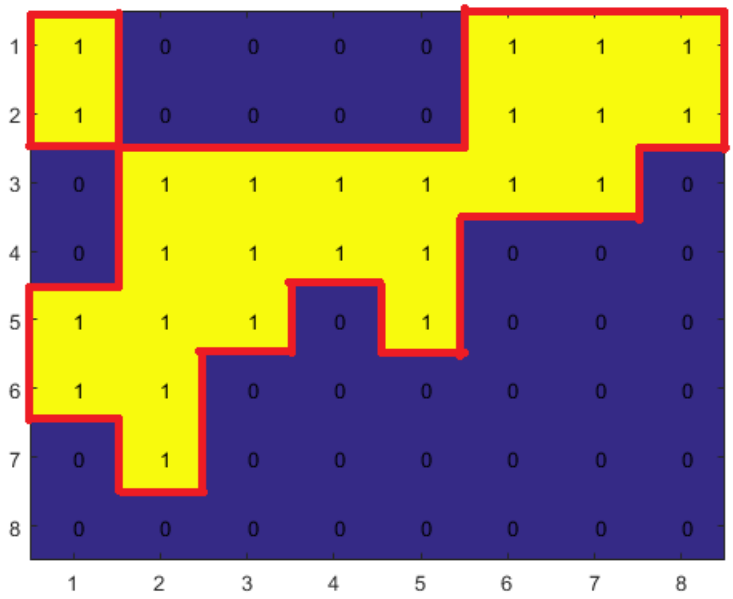

Figura 19 – Corte Y deve ser enviado inteiro comprimido em JBIG.

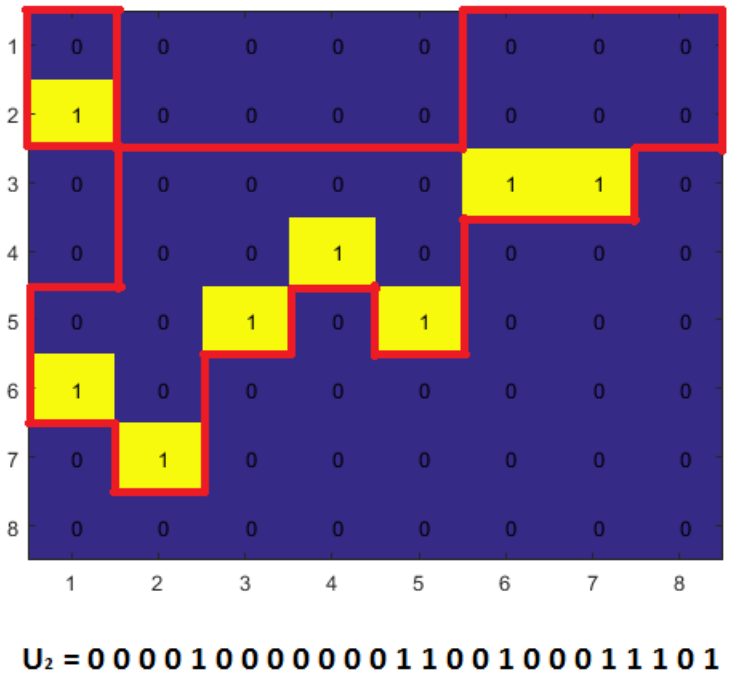

Figura 20 – Y_1 e seu bitstream a ser enviado.

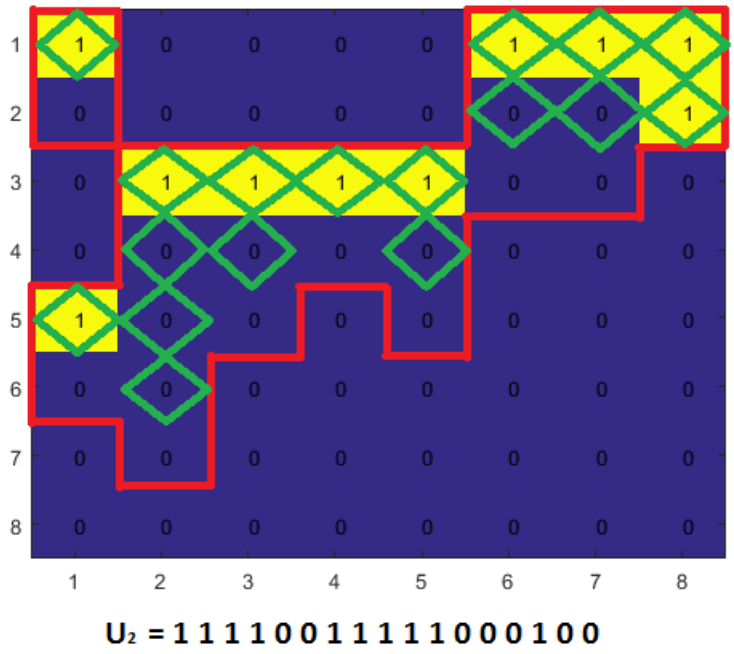

Figura 21 – Y_2 e seu bitstream a ser enviado.

A parte destacada em vermelho representa o espaço onde Y(x,y) = 1, ou seja, toda a informação relevante dos cortes. Esta área delimita quais serão os pixels enviados em U_1, em seguida, na Figura 21, os pixels destacados em verde representam os que, dentro da informação relevante em vermelho, $Y_1(x,y) = 0$, apenas estes são necessários para criação de U_2.

Observando as três figuras pode-se notar quanta informação redundante foi eliminada no processo. Sem compressão alguma, cada imagem ocuparia um espaço total de $16 * 16 = 256$ bits, totalizando 768 bits para as três imagens. Com a metodologia de *raster scan* e eliminação de informação adotada, o espaço total ocupado reduz-se a $256 + 25 + 17 = 298$, sendo 256 o número de bits em Y, 25 o número de bits em U_1 e 17 o número de bits em U_2. Uma compressão de 61,2%, de uma taxa de 1 bpp para 0,388 bpp. O envio base, ou seja, dos cortes A e B, geraria um total de $256 + 256 = 512$ bits a serem enviados, portanto, se comparado à base, o método desenvolvido reduziu o número de bits de 512 para 298, houve uma compressão de 41,7%.

Em resumo, o algoritmo proposto pode ser interpretado da seguinte forma: Primeiro define-se os *headers* iniciais como sendo as dimensões da matriz da nuvem de pontos, o número dos cortes inicial e final, e uma *flag* para dizer se o número de cortes é par ou ímpar. Em seguida, é feita a decomposição booleana em todos os cortes, 2 a 2. Verifica-se então qual a forma mais efetiva de compressão, utilizar o algoritmo JBIG nos dois cortes ou realizas as manipulações booleanas do método proposto. Agrega-se então ao *bitstream* os valores corres-

pondentes à codificação dos dois cortes em questão, caso houver um número ímpar de cortes, este será codificado em JBIG e anexado ao fim do bitstream. Por fim, transforma-se o *bitstream* em *bytestream*, colocando os zeros necessários no final para tranformar o *bitstream* em múltiplo de 8.

Retornando ao exemplo da Figura 18, onde são utilizados os cortes 295 e 296 de *Ricardo9*, podemos realizar as mesmas manipulações e chegar nos *bitstreams* U_1, U_2 e na imagem Y codificada em JBIG, com resultados apresentados de compressão apresentados na Tabela 7. Os dados demonstram que mesmo utilizando o algoritmo JBIG para comprimir os cortes A e B, o aproveitamento do contexto entre eles garantiu um número total de bits para envio ainda menor, com todos os *headers* necessários. Somando então o número total de bits necessários para o envio dos cortes tem-se $3904 + 3888 = 7792$ bits para enviar A e B, e $4064 + 1909 + 1584 = 7557$ bits para enviar Y, Y_1 e Y_2.

Tabela 7 – Número de bits utilizados

A(JBIG)	B(JBIG)	Y(JBIG)	$Y_1(U_1)$	$Y_2(U_2)$
3904	3888	4064	1909	1584

3.3.1 Escolhendo qual método utilizar

Todo o trabalho realizado até este ponto converge para dois principais métodos de compressão dos cortes, pode-se apenas comprimí-los utilizando o algoritmo JBIG, ou pode-se manipulá-los de forma a obter as imagens Y, Y_1 e Y_2. Para garantir a menor taxa possível, o projeto realizado utiliza os dois métodos, verificando, a cada iteração, qual o método que garante menor número total de bits. Acrescenta-se então um *header* de um bit para explicitar ao decodificador qual o método utilizado.

Vejamos por exemplo qual a taxa resultante dos dois métodos para alguns cortes retirados da nuvem de pontos "Ricardo9", sendo o método 1 a compressão apenas por JBIG e o método 2 a compressão através do aproveitamento do contexto tridimensional, os resultados são exibidos na Tablea 8.

Neste pequeno exemplo é possível perceber que existem momentos em que o método 1 comprime mais os cortes e há momentos em que o método proposto é melhor, a razão por trás desta diferença está no aproveitamento do contexto entre os dois cortes. Se existem mudanças bruscas entre os pixels de um corte e outro, o contexto 3D não será tão bem aproveitado, e,

Tabela 8 – Tamanho em bits dos cortes para método 1 (JBIG) e método 2 (Proposto)

Cortes	Método 1	Método 2
258 e 259	2048	1294
280 e 281	7104	8346
302 e 303	9648	11331
322 e 323	10264	11344
338 e 339	8264	8175
356 e 357	6968	6581

por isso, comprimir cada imagem individualmente ocasionará em uma taxa menor. Em outras palavras, quanto maior o número de pontas na imagem, como os dedos de uma pessoa por exemplo, menor será o contexto utilizável, quanto mais homogênea for a imagem, melhor será a compressão através do método proposto. É por esse motivo que é interessante a implementação de um decisor que faz a melhor escolha sempre. Este decisor é feito de forma simples, basicamente se aplica os dois métodos e verifica qual foi mais eficiente. O tema a respeito da dependência da homogeneidade do modelo tridimensional será melhor abordado no capítulo de análise de resultados, onde ambos métodos serão aplicados a uma vasta gama de modelos.

O leitor pode estar se perguntando neste momento o que ocorre se o número de cortes não for par, afinal, as compressões são sempre de 2 em 2. O codificador deve sempre sinalizar por *header* o número de cortes codificados, dessa forma o decodificador sabe quantos cortes são enviados. Com isso, se o número de cortes for ímpar, o último corte será sempre codificado individualmente pelo JBIG.

4 Análise de resultados

4.1 Resultados Obtidos

Nesta seção, serão utilizados dois vídeos em nuvem de pontos, ou seja, uma sequência nuvens de pontos que foram capturadas uma atrás da outra, sendo assim, elas possuem muita semelhança entre si e nos permitem análisar quais mudanças no modelo geraram diferenças no momento de compressão. Dessa forma, primeiramente será feita uma análise individual de uma nuvem de pontos e demonstrada a eficiência de compressão do algoritmo. Em seguida, serão expostos gráficos que relacionam todos os modelos disponíveis. Com isso teremos dados o suficiente para tirar conclusões baseadas em estatística e não apenas na análise de resultados individuais. Os vídeos de nuvem de pontos utilizados compõem uma base de dados apresentada em [23]. Os dados a serem avaliados serão os frames de número 50 de quatro vídeos de nuvens de pontos diferentes, como apresentado nas Figuras 22, 23, 24 e 25.

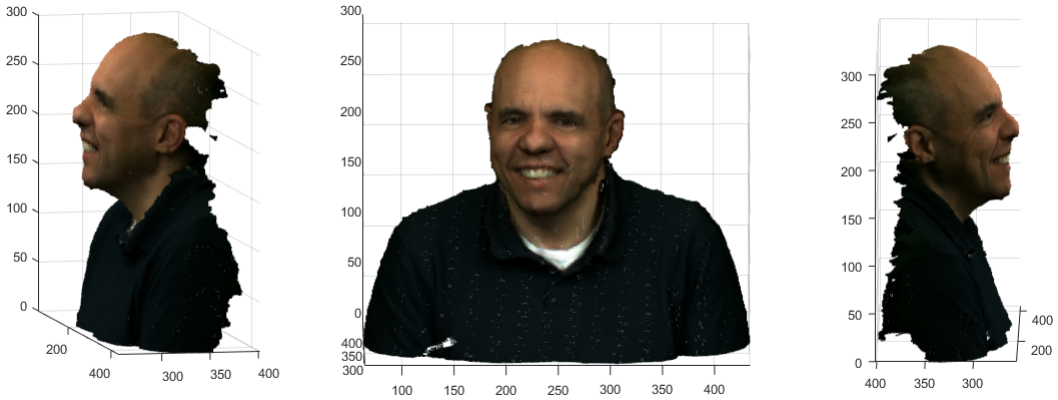

Figura 22 – a) Nuvem de pontos "*Ricardo*9".

Tabela 9 – Valores de compressão de *test samples*

	Original(bytes)	Comprimido(bytes)	Voxels	Taxa (*bpov*)	Compressão (%)
Ricardo9	6.607.037	59.026	300.693	2,4158	99,11
Sarah9	7.915.166	64.580	242.794	2,127894	99,18
Phil9	7.483.392	103.065	337.283	2,4446	98,62
David9	6.483.968	87.804	303.971	2,022	98,65

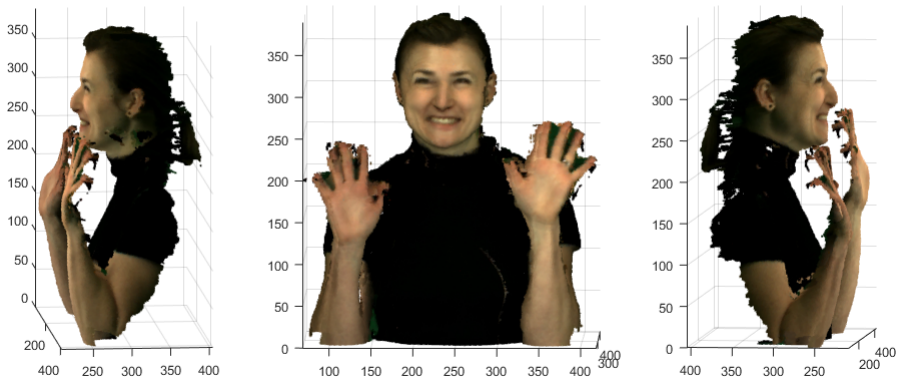

Figura 23 – b) Nuvem de pontos "*Sarah*9".

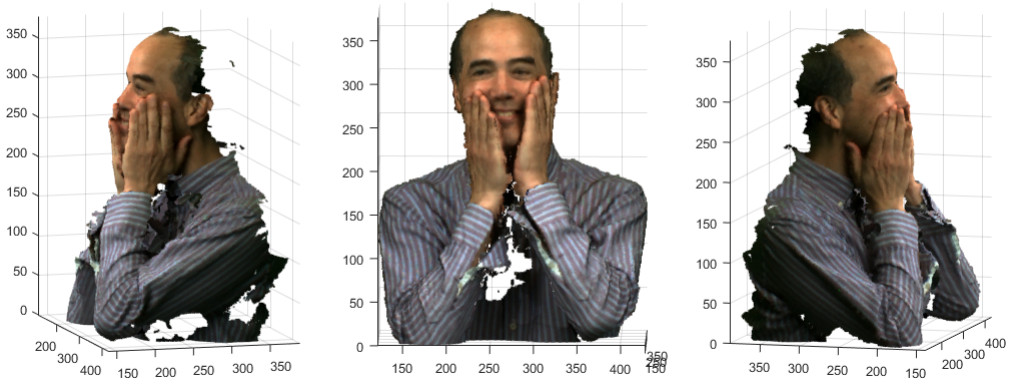

Figura 24 – c) Nuvem de pontos "*Phil*9".

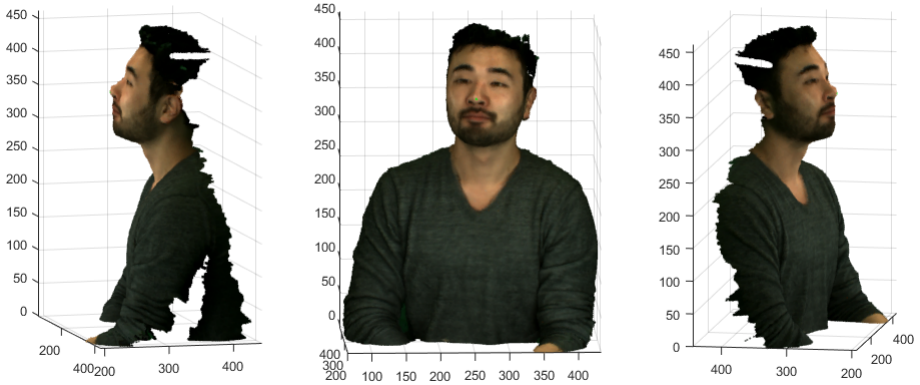

Figura 25 – d) Nuvem de pontos "*David*9".

Os valores de tamanho dos arquivos sem compressão alguma não são muito relevantes pois sempre serão exageradamente grandes, todo arquivo manipulado por computadores atualmente sofrem algum tipo de compressão. Por essa razão os percentuais de compressão são tão altos, na casa de 99%. O que é mais interessante de ser notado nos testes é a taxa final de compressão em bits por voxel ocupado, pois é essa taxa que relaciona o espaço de memória

ocupado por cada elemento de volume da nuvem de pontos. Os testes realizados demonstraram uma taxa de 2,415 bits por voxel ocupado para a imagem *Ricardo9* e 2,127894 para a imagem *Sarah9*, valores extremamente interessantes quando comparados ao atual estado da arte. As taxas para as imagens sem compressão são de 32,48 bits por voxel ocupado para *Ricardo9* e 32,6 para *Sarah9*.

Após apresentados os valores para duas imagens específicas, pode-se realizar o mesmo procedimento para todos os frames do vídeo e então analisar todas as taxas resultantes. Vale ressaltar que nos vídeos cada frame possui uma forma diferente, e isso é muito interessante porque é possível determinar relações de causa e efeito entre a geometria da nuvem de pontos e a taxa resultante.

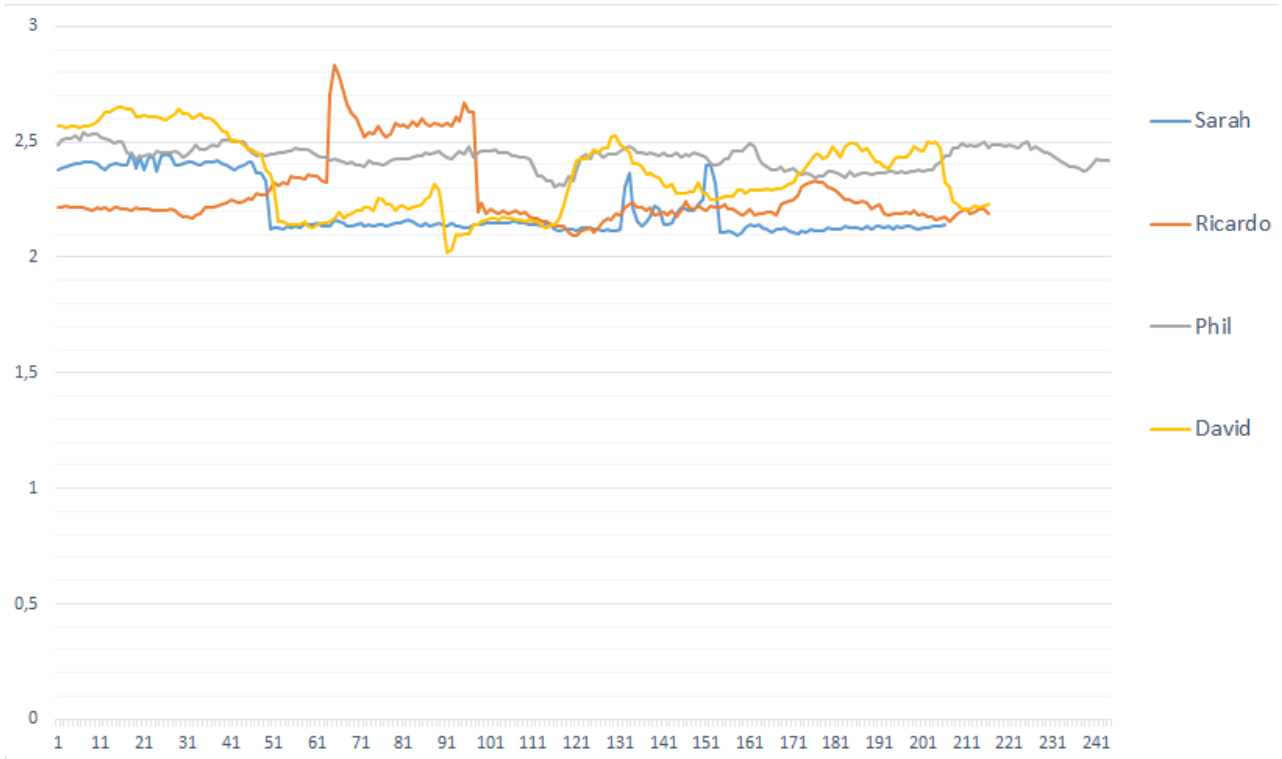

Figura 26 – Taxa por frame, em bpov, para os *test samples* "*Ricardo*9", "*Sarah*9, "*David*9", "*Phil*9".

É evidente que as taxas sofrem variações ao longo de cada frame e que essas variações possuem ora tendência de alta ora tendência de baixa. Esta característica pode ser atribuída ao já comentado aspecto de que o algoritmo desenvolvido sofre bastante em sua tentativa de aproveitar o contexto quando a imagem possui muitas pontas, em outras palavras, quando não há a mudança contínua entre um corte e outro, como é esperado. Nos valores de taxa apresentados no gráfico da Figura 26, a curva de "*Ricardo*9" demonstra isso com bastante

clareza durante os frames 63 a 96, quando a taxa sofre um repentino pico. Exatamente o período em que o homem da imagem decide levantar os braços, gerando assim um maior número de pontas na imagem do que antes, quando ele estava com os braços colados ao corpo.

Outro aspecto importante a ser analisado nos resultados é quantas vezes cada método é utilizado, ou seja, se o algoritmo desenvolvido neste projeto consegue consistentemente ganhar do JBIG na compressão de cada corte. Uma alta utilização do algoritmo proposto demonstra que, além de ser superior a um algoritmo já consolidado no âmbito acadêmico, se ainda houver formas de comprimir um pouco mais os bitstreams U_1 e U_2, com a utilização de um codificador de entropia, por exemplo, pode ser possível reduzir ainda mais a taxa obtida. Caso o JBIG esteja sendo utilizado na maioria dos casos, como este algoritmo já é muito otimizado, pode ser que a metodologia proposta não seja tão competitiva em comparação com o estado da arte.

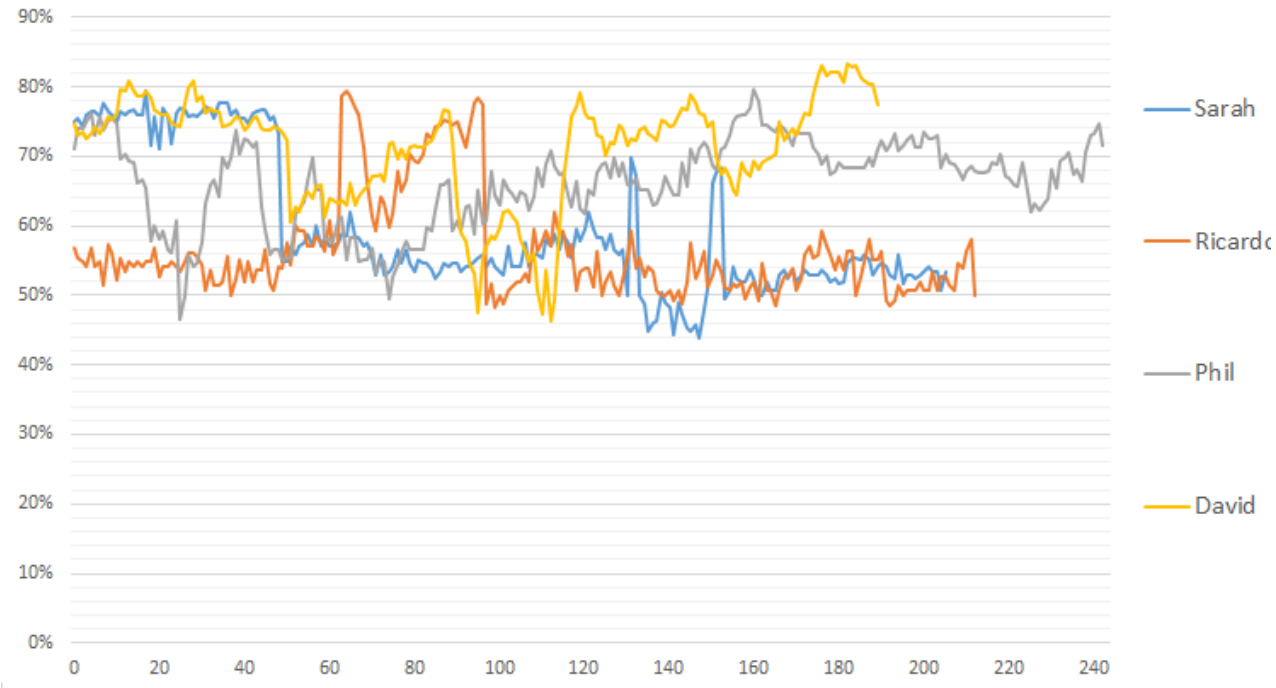

Figura 27 – Percentual de utilização do algoritmo proposto para os *test samples* "*Ricardo*9", "*Sarah*9, "*David*9", "*Phil*9".

Como exposto no gráfico da Figura 27, o percentual de utilização do algoritmo proposto varia muito para cada frame, porém, esse valor está quase que inteiramente acima de 50%, e em média na casa dos 70%. Isso demonstra que o aproveitamento do contexto é uma estratégia muito eficiente capaz de comprimir mais que um dos mais conhecidos algoritmos de compressão de imagens binárias.

A utilização de um codificador de entropia para os resultados finais de U_1 e U_2 foi

testada e demonstrou uma compressão da imagem final em adicionais 5 a 6%. O codificador utilizado para tal foi um codificador Huffmann, de desenvolvimento próprio, utilizando os conceitos expostos em [9]. O desenvolvimento de um codificador específico para U_1 e U_2 pode levar essa taxa a ser ainda menor. Esta idéia será melhor exposta no capítulo que trata de futuros desenvolvimentos.

É interessante ainda a comparação do modelo proposto com a compressão dos cortes puramente com o JBIG. Como pôde ser visto na seção 3.3.1, o algoritmo desenvolvido escolhe entre o método proposto e o JBIG o que resultar em menor número de bits, portanto é esperado que ao ser comparado com apenas a codificação dos cortes em JBIG o método proposto seja mais eficiente. A tabela 10 apresenta uma comparação entre a compressão dos frames 50 de cada nuvem de pontos da base de dados tanto pelo método proposto quanto por apenas JBIG.

Tabela 10 – Valores de compressão de *test samples*

	Método proposto(bytes)	JBIG(bytes)	Diferença(%)
Ricardo9	59.026	62.170	5,06
Sarah9	64.580	87.809	26,45
Phil9	103.065	107.166	3,83
David9	87.804	99.243	11,53

Os resultados obtidos apresentam uma grande diversidade com relação à capacidade de compressão do método desenvolvido. Os valores de diferença percentual variam desde 3,83% em Phil9 a 26,45% em Sarah9, ou seja, pode-se inferir que a capacidade de compressão dos cortes através de manipulação booleana está muito vinculado às características da nuvem de pontos em questão, por isso faz bastante sentido aplicar o decisor que sempre opta pelo método mais eficaz. Embora o decisor faça necessária a execução dos dois métodos, o tempo gasto para tal parece ser, em primeira análise, um fator pouco significante tendo em vista o ganho em compressão fornecido.

4.2 Estado da Arte

Existem inúmeros trabalhos em compressão de sinais que abordam especificamente o conceito de compressão de nuvens de pontos, alguns tentam manipular as já conhecidas *octrees*,como em [16], outros propoem métodos de predição, como em [17]. Esta seção busca expor alguns dos trabalhos que obtiveram resultados efetivos, ou seja, os que podem ser considerados estado da arte em compressão de nuvens de pontos, além de métodos já utilizados

comercialmente, e compará-los com o método proposto. Para esta finalidade, serão utilizados os resultados apresentados em [8] e em [19], que utilizam a mesma base de *test samples* utilizada neste trabalho, facilitando, dessa forma, a comparação e análise.

Em primeira análise, serão comparados algoritmos já produzidos e utilizados comercialmente. Desde algoritmos de compressão genérica, ou seja, que não são voltados a nuvens de pontos especificamente, até algoritmos específicos, como as *octrees*.

Tabela 11 – Cenários de teste propostos.

MPEG	Código âncora MPEG
DRACO	Draco open-source
OR	Octree
AC	Representação Octree em código aritmético
LZW	Octree codificada em LZW
$[P1]$	Código 1 proposto em [8] como P(AC)
$[P2]$	Código 2 proposto em [8] como P(LZW)
$[M1]$	Código 1 proposto em [19] como método 1
$[M2]$	Código 2 proposto em [19] como método 2
$[P]$	Código Proposto neste trabalho

A tabela 11 identifica todos os métodos propostos que foram submetidos a testes com a mesma base de dados, vale ressaltar que os resultados foram obtidos diretamente das referências, não foram realizados testes reais para cada um deles.

Tabela 12 – Taxa média em bit por voxel ocupado para cada *test sample*

	David9	Phil9	Sarah9	Ricardo9
MPEG	2,71	2,89	3,10	3,53
DRACO	4,35	4,54	4,86	4,82
OR	2,62	2,64	2,61	2,92
AC	2,05	2,10	2,04	2,43
LZW	1,89	2,00	1,89	2,37
$[P1]$	1,97	2,03	1,97	2,37
$[P2]$	1,73	1,84	1,75	2,27
$[M1]$	2,41	2,31	2,05	2,6
$[M2]$	1,87	1,88	1,72	2,25
$[P]$	2,35	2,43	2,20	2,27

As taxas obtidas com o método proposto se aproximam do estado da arte, sendo superior a alguns métodos em determinadas imagens. Tais resultados são muito interessantes pois demonstram que o trabalho aqui realizado pode resultar em um método bem definido que corresponda ao estado da arte em compressão de nuvens de pontos. É fácil notar que os códigos propostos em [19] e [8] superam com folga os métodos já comercializados, isso também

ocorre na maioria dos casos do método proposto neste trabalho. Para discutir estado da arte, então, é mais interessante comparar apenas os artigos. Para isso, podemos criar uma tabela que representa apenas a variação percentual de cada método com relação ao método aqui proposto.

Tabela 13 – Variação percentual com relação ao método proposto $[P]$

	David9	Phil9	Sarah9	Ricardo9
$[P1]$	-16,17%	-16,46%	-10,45%	+4,41%
$[P2]$	-26,38%	-24,28%	-20,45%	+0,00%
$[M1]$	+2,55%	-4,94%	-6,82%	+14,54%
$[M2]$	-20,43%	-22,63%	-21,82%	-0,88%

Para o vídeo Ricardo9, o método proposto $[P]$ obteve a melhor taxa média quando comparado aos outros métodos, basicamente pelo fato de este ser o vídeo em que há menos movimentação da pessoa com braços e outros movimentos que ocasionem um grande número de pontas. Em valores absolutos, os métodos que garantiram menor taxa foram $[P2]$ e $[M2]$, podendo ser caracterizados como estado da arte, assim como seus próprios autores os definem. As taxas para estes métodos são realmente extremamente eficientes, sendo até 26% menores que o método aqui proposto. Porém, elas utilizam mecanismos muito mais sofisticados, ou seja, o fato de o método $[P]$ chegar tão próximo do estado da arte com baixa utilização de algoritmos mais sofisticados demonstra que um grande potencial para a técnica desenvolvida. Além disso, vale salientar que os métodos utilizados em [19] são inter-frame, enquanto o método aqui desenvolvido é intra-frame, fato pelo qual é esperado que os métodos utilizados em [19] sejam realmente mais eficientes.

5 Conclusões e trabalhos futuros

O principal propósito deste trabalho foi apresentar uma nova metodologia para compressão de nuvens de pontos, separando a arquitetura em diversos cortes bidimensionais e tratando estes cortes com operações booleanas pixel a pixel. Apresentou-se então toda a lógica por traz de cada manipulação e quais são as vantagens e desvantagens de adotá-las. Desenvolveu-se também alternativas e idéias futuras que possam vir a ser utilizadas em futuros trabalhos. Como apresentado no primeiro capítulo, a utilização de nuvem de pontos vem sendo bastante explorada principalmente nos ramos de análise e manipulação de dados, como por exemplo reconhecimento facial, tornando o tópico abordado neste trabalho relevante para o ramo.

Os resultados apresentados foram comparados com os principais modelos de compressão utilizados atualmente bem como com modelos propostos por outros autores que definem o atual estado da arte. As taxas de compressão apresentadas pelo modelo proposto não superaram o estado da arte em algumas amostras porém há amostras em que os valores são muito próximos, além disso, com relação aos métodos utilizados comercialmente, o algoritmo de compressão por manipulação booleanda foi capaz de superar de forma consistente, o que demonstra ser um algoritmo competitivo em termos de capacidade de compressão. A seção 5.2 deste capítulo trata justamente de apresentar formas e idéias para que tornar o método proposto ainda mais eficiente. Além de alertar à futuros trabalhos os caminhos já testados e que não obtiveram muito sucesso até o momento.

Em conclusão, o trabalho aqui desenvolvido sugere uma nova abordagem para a compressão de nuvens de pontos, o que é muito positivo pois introduz novos conceitos e manipulações que poderão contribuir para o desenvolvimento de futuros trabalhos. O fato do novo método chegar tão próximo ao estado da arte demonstra que, se otimizado, existe a possibilidade deste método ser tão eficiente quanto os baseados em octrees, por exemplo, que são os mais eficientes atualmente.

5.1 Códigos Desenvolvidos

Todos os códigos utilizados para a aplicação do algoritmo foram desenvolvidos como parte do projeto e estão disponíveis no repositório do site GitHub no link fornecido.

https://github.com/alves63/Compress-odePointClouds

5.2 Futuro desenvolvimento

Esta seção tem como objetivo apresentar algumas das diversas metodologias testadas neste trabalho e propor novos caminhos que possam ser significativos no desenvolvimento de novas metodologias ou até mesmo agregar à metodologia já proposta.

5.2.1 Agregando Octree

Uma possível adição ao algoritmo desenvolvido neste trabalho é a utilização de *octrees* para os primeiros níveis. Sabe-se que, se determinado cubo-filho não possui nenhum elemento em seu interior, ele é tratado como um bit 0 apenas, ou seja, não existe algoritmo mais eficiente para comprimir espaços vazios dentro de uma nuvem de pontos que as *octrees*. Portanto, é interessante o teste de se utilizar esta metodologia para os primeiros níveis, pois, se existirem grandes espaços vazios na imagem, eles serão muito melhor comprimidos para que, em seguida, nos cubos menores, possa-se aplicar o método de cortes proposto neste trabalho.

Esta proposta de mesclar duas metodologias foi testada apenas a título de testes iniciais, não tendo o mesmo foco que o método proposto. Foram testados até 4 níveis de cubos-filho. Primeiro foi realizada a divisão e atribuição dos bits 1 e 0 para cubos ocupados e vazios. Em seguida, aplicou-se o método de cortes proposto neste trabalho a cada cubo preenchido, garantindo assim uma decodificação completa, sem perda de informação. Os resultados não foram superiores à utilização apenas do método de cortes na imagem inteira, isto se deve principalmente ao fato de que a compressão do algoritmo JBIG não é diretamente proporcional ao tamanho da imagem, ou seja, se uma imagem 512x512 comprimida por JBIG resultou em um arquivo de tamanho próximo a 1Kb, a compressão de uma imagem 256x256 não gera um arquivo de 0,25Kb. Na realidade, o arquivo gerado pela imagem de 256x256 é maior que 0,25Kb porque como exposto em tópicos anteriores, o JBIG se vale do contexto da imagem, e quanto maior a imagem, maior o contexto e consequentemente maior a taxa de compressão. Esse motivo

já é suficiente para que a compressão de uma nuvem de pontos 512x512x512 seja mais eficiente do que comprimir os 8 filhos de 256x256x256.

5.2.2 Recursividade

Imaginemos que dois cortes consecutivos de uma nuvem de pontos, A e B gerem os novos cortes Y^{ab}, $Y^{ab}{}_1$ e $Y^{ab}{}_2$. Imaginemos agora que os próximos dois cortes, C e D, gerem também seus respectivos cortes Y^{cd}, $Y^{cd}{}_1$ e $Y^{cd}{}_2$. É possível, com as imagens Y^{ab} e Y^{cd}, aplicar novamente o algoritmo e gerar então os cortes Y^{abcd}, $Y^{abcd}{}_1$ e $Y^{abcd}{}_2$. Dessa forma, os cortes necessários para se recuperar, sem perda de informação, os cortes iniciais A,B,C e D são: Y^{abcd}, $Y^{abcd}{}_1$, $Y^{abcd}{}_2$, $Y^{cd}{}_1$, $Y^{cd}{}_2$, $Y^{ab}{}_1$ e $Y^{ab}{}_2$. Se seguirmos esse raciocínio, é possível contemplar todos os cortes em um único corte Y, e, voltando recursivamente, pode-se utilizar os Y_1 e Y_2 de cada nível para recuperar todos os cortes originais. Supondo um número N de cortes iniciais, ao final de todo processo recursivo haveriam $N * 2 - 1$ cortes derivados dos iniciais. A suposição por trás dessa abordagem é que os cortes finais possuiriam um contexto muito mais bem definido e que seria bem aproveitado pelo algoritmo JBIG. Algo que se mostrou verdadeiro, porém, o método original sem recursividade tem, ao seu final, $N * 3/2$ cortes derivados dos iniciais, um número extremamente menor e que torna o método recursivo não factível. Embora seu contexto seja de fato melhor, o grande número de cortes gerados o torna menos eficiente. Entretando, assim como o teste da utilização de cubos-filho de octrees, esse modelo foi testado sem grande profundidade, mais testes podem ser realizados para se encontrar uma solução.

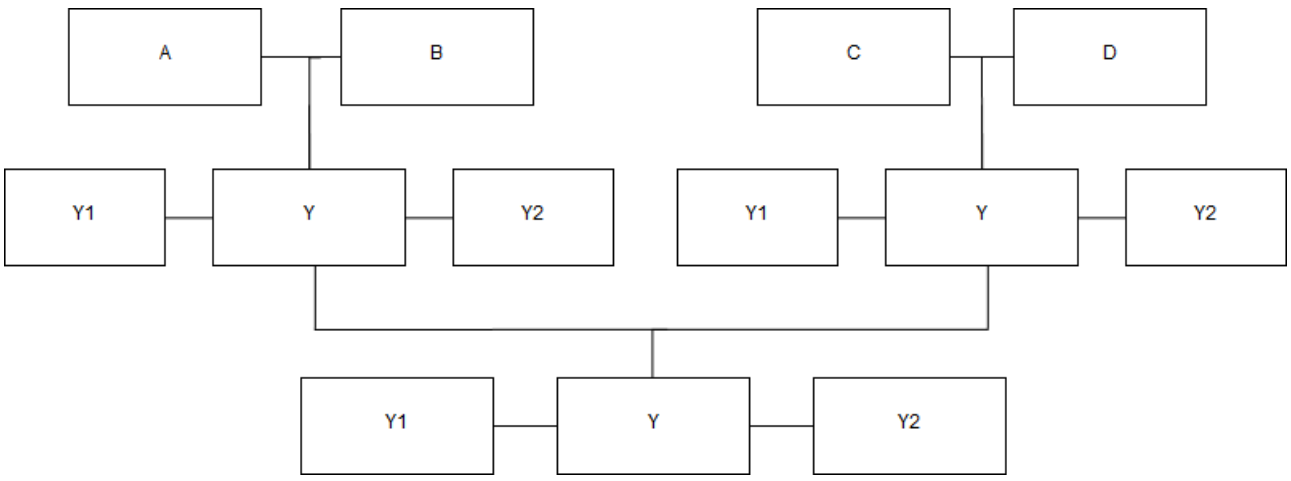

Figura 28 – Recursividade na obtenção dos cortes.

5.2.3 Compressão Adicional

O método proposto mostrou-se superior ao JBIG em algo próximo de 70% dos cortes analisados, como pode ser visto na Figura 27, o que nos leva a imaginar que, se for possível de alguma forma comprimir ainda mais o *bitstream* formado, conseguiremos uma taxa ainda menor. Testes realizados com um algoritmo de codificação Huffmann simples foi capaz de diminuir o tamanho ocupado pelos cortes Y_1 e Y_2 em adicionais 5%. Caso seja criado um codificador próprio para o *bitstream* gerado, esse nível de compressão pode aumentar ainda mais. Essa é uma proposta que pode ser avaliada em futuros trabalhos.

5.2.4 Contexto tridimensional

Embora a utilização de recursividade para contemplar o contexto de todos os cortes não tenha funcionado pelo fato de o número de imagens criadas ser muito grande, a utilização de um contexto que englobe mais que apenas dois cortes pode ser uma boa sugestão para diminuir a taxa. Uma idéia a ser proposta é utilizar a mesma metodologia de codificação aritmética utilizada pelo JBIG, porém, ao invés da vizinhança ser puramente bidimensional, acrescentar os *voxels* de cortes já processados para, dessa forma, calcular a probabilidade do *voxel* em análise ser 1 ou 0. Como essa sugestão não envolve separar a nuvens de pontos em cortes, talvez seja interessante também mesclar esse tipo de algoritmo com as octrees, que sem sombra de dúvidas são o método mais eficiente para codificar espaços vazios. Utilizar octrees primariamente em um ou dois níveis e codificar os cubos filhos resultantes através de outro modelo é uma proposta que continua muito interessante, embora não tenha se mostrado muito eficiente nos testes deste trabalho especificamente.

Referências

[1] Autodesk. About point cloud, 2018.

[2] S. Bajaj. Data compression with arithmetic coding, 2017.

[3] L. Boxer. The future of point cloud?, 2017.

[4] P. Cohen. A history of hard drives, 2014.

[5] A. Daskalaki, C. Lobos, Y. Payan, and N. Hitschfeld. Techniques for the generation of 3d finite element meshes of human organs. pages 126–158, 01 2010.

[6] R. Ellis, A. Kahng, and Y. Zheng. Technical Report : JBIG Compression Algorithms for " Dummy Fill " VLSI Layout Data. pages 1–31, 2002.

[7] P. Franti, P. Kopylov, and E. Ageenko. Evaluation of compression methods for digital map images. *IASTED Int. Conf. on Automation, Control and Information Technology (ACIT 2002)*, pages 401–405, 2002.

[8] D. C. Garcia and R. L. de Queiroz. Intra-frame context-based octree coding for point-cloud geometry. 2018.

[9] D. A. Huffman. A method for the construction of minimum redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40(9):1098–1101, 1952.

[10] IBM. 305 ramac manual of operation, 1975.

[11] ISO/IEC. Progressive bi-level image compression. *ITU-T Recommendation*, 82:11544, 1993.

[12] ITU. International morse code. *Recommendation ITU-R M.1677-1*, 2009.

[13] S. B. Jr. Are point clouds still the future?, 2012.

[14] G. Langdon and J. Rissanen. Compression of black-white images with arithmetic coding. *IEEE Trans. Communications*, 29(6):858–867, 1981.

[15] S. Lefebvre, S. Hornus, and F. Neyret. Chapter 37. octree textures on the gpu, 2005.

[16] T. Lin, F. Da, and H. Yuan. Compression algorithm of scattered point cloud based on octree coding. *2016 2nd IEEE International Conference on Computer and Communications, ICCC 2016 - Proceedings*, pages 85–89, 2017.

[17] D. Mongus, D. Špelič, B. Žalik, and B. Rupnik. Geometry compression of scanned point-clouds. *ICSTE 2010 - 2010 2nd International Conference on Software Technology and Engineering, Proceedings*, 1:279–282, 2010.

[18] J. Peng and C. Kuo. Octree-based progressive geometry encoder. *Proc. Symposium on Point-Based Graphics*, pages 301–311, 2006.

[19] R. L. D. Queiroz, D. C. Garcia, P. A. Chou, and D. A. Florencio. Distance-Based Probability Model for Octree Coding. 25(6):739–742, 2018.

[20] K. Sayood. *Introduction to Data Compression*. Morgan Kaufmann, 2012.

[21] R. Schnabel and R. Klein. Octree-based point-cloud compression. *Spbg*, pages 111–120, 2006.

[22] C. E. Shannon. A mathematical theory of communication. *Bell system technical journal*, 27:379–423, 1948.

[23] L. Tang, F.-p. Da, and Y. Huang. Compression algorithm of scattered point cloud based on octree coding. In *Computer and Communications (ICCC), 2016 2nd IEEE International Conference on*, pages 85–89. IEEE, 2016.