UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

3D-CSD+: Extração de características 3D baseada em grafos.

Jean Amaro

Dissertação de Mestrado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

ICMC USP
SÃO CARLOS

Jean Amaro

3D-CSD+: Extração de características 3D baseada em grafos.

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Fernando Santos Osório

USP – São Carlos
Agosto de 2023

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

A4853

Amaro, Jean
3D-CSD+: Extração de características 3D baseada em grafos. / Jean Amaro; orientador Fernando Santos Osório. -- São Carlos, 2023.
100 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2023.

1. Visão Computacional. 2. Extração de características 3D. 3. Reconhecimento de padrões. I. Santos Osório, Fernando, orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Jean Amaro

3D-CSD+: Graph based 3D feature extraction.

Master dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP, in partial fulfillment of the requirements for the degree of the Master Program in Computer Science and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Fernando Santos Osório

USP – São Carlos
August 2023

AGRADECIMENTOS

À minha família, pelo seu amor e suporte incondicionais.

Ao Professor Doutor Fernando Santos Osório, pela sua orientação e paciência nesta jornada.

A todos os meus amigos e colegas de laboratório; pelos momentos de alegria, aprendizado e frustração juntos.

Em especial, um agradecimento à Mariana Rodrigues, pela ajuda em todo o desenvolvimento desta dissertação.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), pelo apoio financeiro (processo 132756/2018-8).

RESUMO

AMARO, J. **3D-CSD+: Extração de características 3D baseada em grafos.**. 2023. 100 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2023.

A área da Robótica é uma das que se beneficia do desenvolvimento tecnológico recente, testemunhando um crescente interesse no desenvolvimento de novas aplicações em diferentes áreas. Em muitas delas, a Visão Computacional desempenha um papel importante, uma vez que muitos robôs dependem de câmeras para o seu funcionamento. Com o desenvolvimento tecnológico, estão hoje disponíveis sensores capazes de obter dados tridimensionais, motivando o desenvolvimento de algoritmos de percepção nesse plano dimensional. Este trabalho de mestrado propõe uma nova técnica para a descrição de objetos 3D, de maneira factível com aplicações limitadas computacionalmente. As características (*features*) extraídas são robustas e invariantes a transformações (p.ex. translação, rotação e mudança de escala), e que permite o reconhecimento de objetos em aplicações embarcadas e com requisitos de tempo real. Em testes usando o *dataset* ModelNet40, chegou-se a uma taxa de acerto Top-3 de 80%, com menos de 20ms de execução por amostra.

Palavras-chave: Visão Computacional, Extração de características 3D, Reconhecimento de padrões.

ABSTRACT

AMARO, J. **3D-CSD+: Graph based 3D feature extraction.**. 2023. 100 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2023.

The field of Robotics is one of those that benefits from recent technological development, witnessing a growing interest in the development of new applications in various areas. In many of these areas, Computer Vision plays an important role, as many robots rely on cameras for their operation. With technological advancement, sensors capable of obtaining three-dimensional data are now available, motivating the development of perception algorithms in this dimensional plane. This master's thesis proposes a new technique for the description of 3D objects, feasible for computationally limited applications. The extracted features are robust and invariant to transformations (e.g., translation, rotation, and scale changes), enabling object recognition in embedded applications with real-time requirements. In tests using the ModelNet40 dataset, a Top-3 accuracy rate of 80% was achieved, with less than 20ms of execution time per sample.

Keywords: Computer Vision, 3D Feature Extraction, Pattern recognition.

LISTA DE ILUSTRAÇÕES

LISTA DE ALGORITMOS

LISTA DE TABELAS

SUMÁRIO

CAPÍTULO

1

INTRODUÇÃO

Este texto apresenta a dissertação de Mestrado desenvolvida junto ao PPG-CCMC do ICMC-USP, no Laboratório de Robótica Móvel (LRM) do ICMC-USP São Carlos, focando em Visão Computacional e Percepção 3D.

Dentre os distintos trabalhos produzidos no Laboratório, é apresentado um projeto com o intuito de ser *hardware agnostic*, ou seja, com a possibilidade de ser utilizado em uma diversa gama de equipamentos, não requerendo processadores com alto poder de paralelização, mas podendo usufruir de tais vantagens, caso elas estejam disponíveis.

1.1 Contexto e Motivação do Trabalho

Avanços recentes na Robótica foram imensamente impulsionados pelo desenvolvimento de algoritmos de Aprendizagem de Máquina, como é o caso das redes neurais e sua sub-área, as redes neurais profundas. Esses algoritmos permitiram que tarefas complexas, como reconhecimento de objetos, navegação e manipulação de objetos fossem feitas com acurácia sem precedentes.

Contudo, sua adoção em sistemas autônomos como veículos não tripulados (terrestres, aéreos ou aquáticos) ou robôs é muitas vezes limitada devido às restrições de memória, processamento e energia — normalmente, esses sistemas são alimentados por uma fonte restrita como uma bateria (SOLAR; LONCOMILLA; SOTO, 2018). Como há uma diferença muito grande de poder computacional entre os dispositivos utilizados para o treinamento e a execução de tarefas (SILVA *et al.*, 2021), nem sempre a portabilidade de modelos pode ser feita de forma simples.

Além das questões de consumo de energia e portabilidade, um outro fator crítico em sistemas robóticos é a velocidade de resposta. Em particular, quando o tema é Visão Computacional, o processamento de informações visuais em tempo real é algo praticamente essencial para diversas aplicações, como os supracitados reconhecimento de objetos, navegação e *tracking*.

Algoritmos pesados, como os utilizados comumente em redes *deep learning*, requerem significante recurso computacional e tempo de processamento, os quais podem limitar sua atuação em cenários do mundo real. Especialmente no cenário de robôs móveis autônomos, a velocidade para se obter uma resposta pode ser algo tão importante quanto uma altíssima acurácia, sempre motivando um equilíbrio entre esses dois fatores.

Com o intuito de contribuir nesta intersecção entre Visão Computacional e Robótica, e sabendo de todo o contexto no qual essa intersecção se insere, propõe-se uma abordagem que possa auxiliar na classificação de objetos 3D, por meio de sua nuvem de pontos e malha da superfície, de maneira rápida e com características importantes para um classificador, como:

- Invariância a escala e rotação, com a possibilidade de transferir conhecimento;
- Capacidade de unir informações geométricas e cromáticas;
- Possibilidade de um analista humano entender como está ocorrendo a classificação, sendo possível observar o porquê das falhas mais claramente.
- Ser agnóstico quanto ao hardware utilizado para processar informações, não necessitando de um acelerador gráfico para lidar com o volume e complexidade dos dados.

Tendo em vista os problemas citados anteriormente, foram buscadas novas formas de descrever e classificar objetos tridimensionais. Uma delas é a abstração do objeto em um conjunto reduzido e representativo de características (*features*) que o descreve. Este conjunto pode ser utilizado por uma análise estatística ou algoritmo de aprendizado de máquina, reduzindo significativamente o custo computacional de todo o processo.

1.2 Objetivos

O objetivo principal deste trabalho é desenvolver, aprimorar, testar e avaliar métodos robustos para o reconhecimento de objetos 3D, por meio da extração de caraterísticas significativas e abstratas, que possam ser aplicados nos mais diversos contextos: desde aplicações embarcadas de baixo poder computacional, podendo ser escalado para sistemas com alto poder computacional e de paralelismo.

Essa abordagem baseada em extração de características representativas, conforme indicado acima, é utilizada no desenvolvimento deste projeto, o qual visa os seguintes objetivos específicos:

- Descrever um objeto, sendo invariante às transformações de escala, rotação e/ou translação, por meio de abstração a um conjunto de características;
- Criar descritores que, ao analisar objetos similares, crie conjuntos contidos dentro uma distribuição normal, facilitando seu uso em diversos algoritmos de classificação, como

algoritmos de aprendizado de máquina ou inferência estatística (para este projeto, foi escolhida a abordagem por inferência estatística no ato da classificação);

- Otimizar os algoritmos e métodos desenvolvidos, de modo a permitir sua execução em aplicações com restrições e limites de tempo (*soft real-time*).

1.3 Estrutura da Dissertação

Esta dissertação está organizada da seguinte forma:

- Capítulo 1 apresenta uma introdução, com o contexto, motivação e principais objetivos do trabalho;
- Capítulo 2 apresenta uma revisão bibliográfica, destacando as principais referências relacionadas com a proposta deste projeto de pesquisa e desenvolvimento de mestrado;
- Capítulo 3 indica os principais métodos e abordagens teóricas usadas no desenvolvimento deste trabalho e também é apresentada a metodologia da pesquisa e desenvolvimento proposto no Capítulo 4;
- Capítulo 5 demonstra os experimentos, validações e discussões sobre o trabalho desenvolvido, terminando com um resumo das principais contribuições obtidas (performance, lacunas e pontos de melhoria do projeto) e as considerações finais.
- Capítulo 6 recapitula os pontos abordados nos capítulos anteriores.
- Apêndice A mostra todas as taxas de acerto, por categoria, por cada esfera utilizada.
- Apêndice B mostra as quantidades de tentativa para acertar cada categoria, por cada esfera utilizada.
- Apêndice C explica brevemente a estrutura do repositório com o código utilizado neste projeto.
- Apêndice D demonstra uma forma de construir uma malha de vértices, usando leituras de um sensor real.

CAPÍTULO

2

REVISÃO BIBLIOGRÁFICA

O reconhecimento tridimensional é uma parte importante na percepção de um ambiente por um sistema computacional que pode ser utilizado em áreas como medicina, segurança e entretenimento (BUSTOS; SIPIRAN, 2012). Esse reconhecimento pode incluir questões como detecção de objetos e sua segmentação, avaliação da posição e ângulo de um objeto, correspondência entre objetos, extração de características e classificação de objetos e formas, e a reconstrução de formas 3D, incluindo ambientes (ORLOVA; LOPATA, 2022).

A classificação de objetos tridimensionais, necessária para a percepção do ambiente, normalmente se baseia em encontrar uma correspondência entre duas formas baseada em alguma métrica de similaridade, normalmente a similaridade visual, ou seja, duas formas que são visualmente parecidas (BUSTOS; SIPIRAN, 2012). Esse processo normalmente inclui as seguintes etapas (NIE *et al.*, 2020):

- Dada uma representação de um objeto 3D, características representativas daquele objeto são extraídas;
- As características são comparadas em relação a objetos já classificados dada uma métrica de similaridade;
- Baseando-se no cálculo de similaridade, a aplicação pode tanto obter uma classificação única ou uma lista de opções com respectivas probabilidades.

Este Capítulo faz uma breve revisão sobre os elementos incluídos no processo acima: representação, descrição e classificação de objetos 3D.

2.1 Representação de dados tridimensionais

A captura de dados de uma cena pode ser convertida em diferentes representações dos objetos presentes em um cenário tridimensional. Nesta seção são mostradas as principais

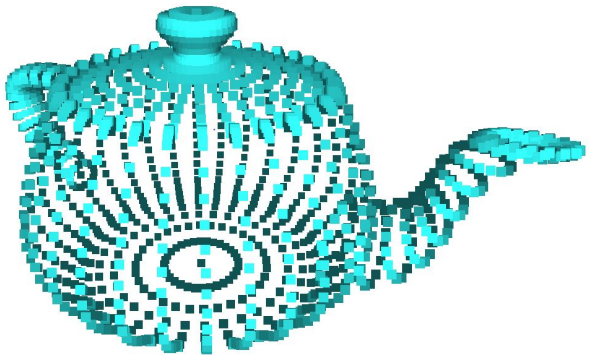
representações de dados 3D de acordo com (ORLOVA; LOPATA, 2022): nuvem de pontos, malha de vértices, modelo voxel.

2.1.1 Nuvem de pontos

Uma nuvem de pontos é um conjunto não-estruturado de dados que pode representar figuras, objetos ou superfícies, sendo utilizadas em muitas áreas da visão computacional como classificação e segmentação, reconhecimento de objetos ou reconstrução de objetos e cenários. A nuvem pode ser facilmente adquirida por um sensor Kinect ou scanners como o LiDAR, mas seu processamento pode ser dificultado por ruídos, dados incompletos ou densidade irregular de pontos.

Ainda, cada ponto pode ser enriquecido com outras informações como cor ou textura, resultando em uma leitura mais detalhada do objeto capturado (GEZAWA *et al.*, 2020). A nuvem de pontos tridimensionais (*3D point cloud*) deve ser preferencialmente representada de uma forma métrica, com as dimensões e distâncias entre os elementos da nuvem de pontos sendo apresentados respeitando uma relação métrica e a geometria dos elementos da cena. A Figura 1 demonstra a representação de um bule por uma nuvem de pontos.

Figura 1 – Representação de um bule em uma nuvem de pontos.



Fonte: Elaborada pelo autor.

2.1.2 Malha de vértices

A malha de vértices é uma combinação de vértices, arestas e faces muito utilizada para armazenamento de objetos 3D e renderização. Os vértices da malha, diferentemente da nuvem de pontos, contém informações de como cada um deles se conecta a outros (GEZAWA *et al.*, 2020). A malha triangular (na qual os vértices se conectam para formar triângulos) é a forma mais popular de representação de dados 3D nos campos de computação gráfica e visualização (ORLOVA; LOPATA, 2022), mas a estrutura nem sempre precisa ser triangular: a malha pode ser formada por diversos polígonos e ter um número arbitrário de vértices (SMITH, 2012). A Figura 2 demonstra o bule representado por uma malha de vértices.

Figura 2 – Representação de um bule em uma malha de vértices.

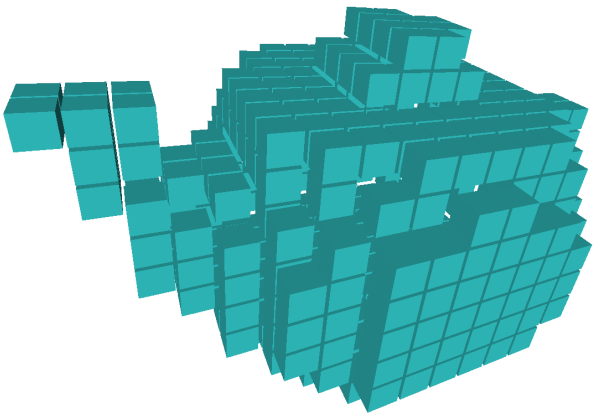
Fonte: Elaborada pelo autor.

2.1.3 Voxel

Voxels são o equivalente 3D de um pixel (VOlumetric piXEL), cada um deles representando um pequeno cubo no espaço tridimensional. A posição de cada voxel é dada por sua localização em uma grade tridimensional em vez de coordenadas, permitindo que a representação seja feita com índices inteiros em vez de números em ponto flutuante, tornando o processamento mais leve e rápido.

Em cada voxel, podem ser armazenadas informações, tanto simples (por exemplo, um booleano indicando se há uma parte do objeto no voxel ou não) quanto mais complicadas (por exemplo, um inteiro representando a densidade ou ocupação do voxel, dados de cor ou material) (SMITH, 2012). Essa representação é muito utilizada na análise de problemas médicos (ORLOVA; LOPATA, 2022). A Figura 3 demonstra um bule representado por voxels.

Figura 3 – Representação de um bule em um espaço *voxelizado*.



Fonte: Elaborada pelo autor.

A maior limitação na utilização do voxel é a demanda por armazenamento, já que tanto espaços ocupados e não ocupados são mapeados e processados, deixando o processo ineficiente. Para enfrentar esse problema, são utilizadas as *octrees*, uma estrutura em árvore onde cada nó

possui oito "filhos" que representam voxels menores. A octree é eficaz para o armazenamento de dados 3D e pode ser utilizada para processamento de voxels de alta resolução, graças ao seu uso eficiente de memória. Entretanto, a octree não é capaz de manter a geometria de alguns objetos, como por exemplo a suavidade de algumas superfícies curvas (GEZAWA *et al.*, 2020).

2.2 Descrição de objetos 3D

A habilidade de descrever um objeto 3D é importante para aplicações de recuperação e análise de objetos, entre as quais a robótica, medicina, biologia molecular, comércio eletrônico, e CAD (computer-aided design) (LOPEZ *et al.*, 2017). Em sistemas de recuperação ou classificação de objetos 3D, existe um banco de imagens já conhecidas que são comparadas com a imagem desconhecida; as imagens conhecidas mais próximas determinam qual a categoria que melhor representa o objeto desconhecido (GEZAWA *et al.*, 2020).

Para que essa comparação seja possível, é necessário que o objeto seja transformado por um *descritor* que o represente na forma mais adequada para a aplicação sendo desenvolvida. O objetivo no uso de descritores é captar a maior quantidade de informação com a menor dimensão possível. É altamente desejável que descritores sejam capazes de (1) lidar com representações de diferentes resoluções, sendo robustos para dados mais densos; (2) reconhecer objetos independentemente de translação, rotação e escala e (3) conseguir fazer a extração e a busca de forma eficiente (LOPEZ *et al.*, 2017).

Existem vários algoritmos para descrever objetos, cuja implementação depende do tipo de representação utilizada (por exemplo, nuvem de pontos ou malha de vértices), o tipo de informação a ser extraído e a aplicação em que ele vai ser utilizado (BRONSTEIN; BRONSTEIN; OVSJANIKOV, 2012).

Os descritores baseados em características (*features*) são um dos tipos mais utilizados na descrição de objetos (CARVALHO; WANGENHEIM, 2019). Esses descritores representam propriedades geométricas e topológicas dos objetos, compactando a representação em um vetor. É possível então calcular a similaridade entre objetos comparando vetores diferentes através de uma dada métrica. Normalmente, são classificados em (TAYBI *et al.*, 2016):

- *Descritores locais*: utilizam a diferença de informações entre um dado local do mapa de profundidade, ou superfície, e outro, gerando uma assinatura local do objeto.

- *Descritores globais*: descrevem o objeto 3D na sua totalidade, reduzindo-o a um único conjunto de informações.

O modo como o descritor é implementado também pode variar, com diversos tipos disponíveis atualmente (o trabalho realizado em (LOPEZ *et al.*, 2017) traz uma descrição de uma ampla gama de implementações). Aqui, algumas delas são destacadas (LOPEZ *et al.*, 2017):

- *Baseados em histogramas*: neste contexto, os histogramas contêm os valores numéricos de algumas características dos objetos, sendo que o espaço é dividido em componentes disjuntos que são por sua vez mapeados para uma classe ou barra do histograma.

- *Baseados em transformação:* esses descritores convertem o espaço contínuo da nuvem em um espaço *voxel* ou malha esférica, gerando descritores compactos e invariantes (qualquer transformação de uma mesma forma terá o mesmo resultado). Isso torna esses descritores muito indicados para medidas de similaridade.

- *Baseados em visão bidimensional:* este método representa um objeto 3D por meio de uma coleção de projeções bidimensionais obtidas sob diferentes pontos de vista. Com esta técnica, é possível utilizar os algoritmos já estabelecidos para imagens 2D; contudo, não há método automático de seleção de quais projeções seriam as mais apropriadas para a descrição e/ou comparação dos objetos.

- *Baseados em grafos:* esses descritores são capazes de codificar propriedades geométricas e topológicas de maneira mais precisa, projetando através de sua estrutura como os componentes do objeto se relacionam. Cálculos de espectro de grafos podem ser utilizados para abstrair o objeto em um vetor de descritores numéricos, e a similaridade de objetos pode ser resolvida através da comparação entre grafos. Estes descritores são muito utilizados na descrição de objetos articulados.

2.3 O uso de *Deep Learning* nas tarefas 3D

O uso de técnicas de *Deep Learning* (DL) vêm dominando várias tarefas de reconhecimento e detecção de objetos; em vários *benchmarks* de visão computacional, as estratégias com melhor resultado aplicam alguma técnica de DL (SOLAR; LONCOMILLA; SOTO, 2018). A disponibilidade de GPUs cada vez mais poderosas contribui muito para que o uso de técnicas de aprendizado profundo sejam aplicadas em tarefas de visão computacional, uma vez que possuem capacidade de treinamento paralelo e alta performance na execução dos modelos, disponibilizando as respostas em tempo real (SILVA *et al.*, 2021; IOANNIDOU *et al.*, 2017).

O trabalho de (IOANNIDOU *et al.*, 2017) apresenta uma revisão de soluções para problemas de visão computacionais 3D baseadas em redes neurais profundas (*deep neural networks* — DNNs), e apresentam dois cenários gerais: (1) as DNNs são alimentadas por descritores extraídos dos objetos para a realização de reconhecimento e recuperação de objetos, entre outras tarefas; e (2) as DNNs são alimentadas diretamente por dados sensoriais, como RGB-D, nuvem de pontos ou voxels, projeções 2D de dados 3D, gerando ao mesmo tempo o descritor e o classificador/recuperador dos objetos. Em (GEZAWA *et al.*, 2020), os autores fazem uma comparação de diferentes métodos de DL alimentados por diferentes tipos de dados,

mostrando que o desempenho de uma técnica é influenciado pela representação utilizada dos dados 3D.

Apesar dos excelente resultados, ainda existem desafios, especialmente na questão de aprendizado contínuo. Treinar o sistema para reconhecer um grande número de classes pode ser um problema, e em alguns casos a inserção de uma nova categoria exige o re-treinamento do modelo utilizado. Isso impacta diretamente a capacidade de aprendizado contínuo no mundo real, ou seja, ter a capacidade de aprender sem supervisão novas classes ou conseguir distinguir entre subclasses incrementalmente uma vez que a "classe mãe" foi aprendida (SOLAR; LONCOMILLA; SOTO, 2018).

2.4 Visão computacional em sistemas autônomos

As tarefas de Visão Computacional são importantes para sistemas como veículos e robôs autônomos. É por meio da visão que estes sistemas são capazes de reconhecer o ambiente no qual estão inseridos e assim tomar a decisão mais apropriada na situação em que se encontram.

Na área de veículos autônomos, as soluções de DL podem ser encontradas em detecção de pedestres ou obstáculos e segmentação de imagens capturadas. Robôs autônomos podem aplicar técnicas DL em várias tarefas de reconhecimento 3D, como segmentação, detecção, localização e classificação de objetos (ORLOVA; LOPATA, 2022). De acordo com (SOLAR; LONCOMILLA; SOTO, 2018), os métodos do estado-da-arte utilizados para detecção e classificação de objetos se baseiam na captura de dados 3D e a utilização de uma DNN para classificação. Entretanto, a transferência dessas técnicas para sistemas autônomos não é direta devido a requisitos diferentes — técnicas de DL normalmente precisam de uma grande quantidade de memória e poder de processamento (que pode incluir o uso de GPUs) para construir uma aplicação, enquanto sistemas autônomos são normalmente operados com uma fonte de energia finita e equipados com componentes de menor capacidade de processamento e armazenamento temporário.

Ainda, sistemas autônomos críticos possuem um risco muito maior do que sistemas simulados — um erro em um robô ou um veículo autônomo pode causar danos permanentes ao próprio sistema, à propriedade de terceiros ou atingir pessoas. É extremamente necessário que o sistema seja robusto e que forneça respostas em tempo real para aumentar a segurança (NIE *et al.*, 2020). Em muitos estudos, não é informada a taxa de atualização necessária para a detecção/classificação do objeto, ou mostram taxas muito distantes do ideal para respostas em tempo real. Ainda, estratégias com respostas próximas ao tempo real necessitam de uma plataforma de grande poder computacional, como GPUs, e não conseguiriam atingir os mesmos resultados em um sistema autônomo de recursos limitados (SOLAR; LONCOMILLA; SOTO, 2018; SILVA *et al.*, 2021).

Além do desafio de recursos computacionais limitados, a detecção/classificação de objetos em sistemas autônomos ainda deve lidar com outras condições como limitação da visão

por causa da geometria do sistema, câmeras de resolução limitada e a posição relativa entre objeto e sensor. Além disso, muitas aplicações autônomas precisam aprender coisas novas à medida que a interação com o mundo aumenta, fazendo com que o aprendizado não-supervisionado de novas classes seja altamente desejável.

2.5 Considerações Finais

São notórias a tendência e predominância do uso de redes neurais profundas em diversas aplicações no contexto da Robótica. Por muitas vezes, requisitando o uso de *hardware* específico de alto desempenho, limitando a implementação de um sistema de visão robusto e genérico.

Tendo em vista o cenário atual, é apresentado um método nesta dissertação que visa a rápida descrição e classificação de um objeto, porém não demandando alto poder computacional. Este método é validado em capítulos posteriores sem o uso de algoritmos de aprendizagem de máquina, apenas usando inferência estatística, mas que pode ser complementado durante a etapa de classificação com tais algoritmos, como uma rede neural profunda.

CAPÍTULO

3

3D-CSD

Neste trabalho, é proposto que toda a classificação de objetos tridimensionais seja baseada no encapsulamento de tais objetos em uma esfera circunscrita, calculando as intersecções dos segmentos formados pelo centro da esfera e todos os vértices na superfície da esfera, e faces do objeto; usando como base o método 3D-CSD (*3D-Countour Sample Distances*) [1], definido previamente em (SALES; AMARO; OSÓRIO, 2017). Por meio desta abordagem, viu-se uma oportunidade de utilizar sua característica que permite explorar a invariância em escala, translação e rotação em torno de um eixo. O método 3D-CSD, ao invés de "olhar de fora" o objeto, a exemplo dos dados brutos coletados por sensores LIDAR ou câmeras de profundidade (*depth cameras*), ele segmenta o objeto de interesse e "olha de dentro para fora"a sua estrutura, permitindo uma abstração do objeto, a partir da posição do observador direcionada ao objeto analisado. Antes de apresentar as melhorias do método, será apresentado o método 3D-CSD.

3.1 Definição

O método 3D-CSD é um descritor 3D invariante à escala, translação e rotação em torno de um eixo para representação da superfície dos objetos presentes no ambiente. Esta seção apresenta o funcionamento deste método.

Sucintamente, o algoritmo 3D-CSD possui quatro etapas:

1. O objeto é segmentado e inscrito em uma esfera descrita por vértices de referência contidos em sua superfície. Um exemplo desta etapa pode ser ilustrado por meio das intersecções dos cortes azimutal e zenital, representando os vértices, na Fig. 4, com um exemplo de recorte na Fig. 5;

2. Em seguida, são criados segmentos entre os vértices de referência e o centro da esfera. Vértices de referência seriam os vértices que formam a malha da esfera, ilustrados

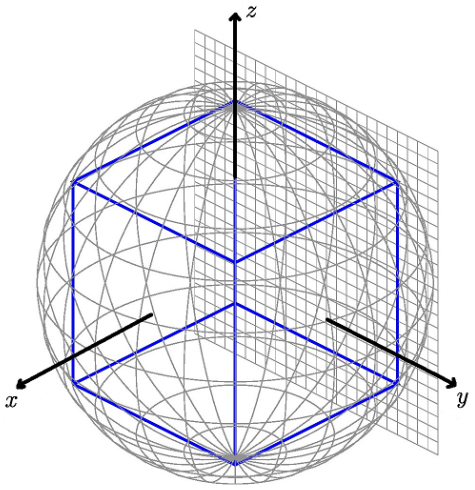
[1] Vídeo de demonstração do Método 3D-CSD: <https://bit.ly/video-demo-descritor-3D>

pelo rótulo **P** na Fig. 5. Os segmentos possuem direção fixada entre o centro da esfera e os diversos vértices de referência da esfera;

3. Cálculos de intersecções destes segmentos com as faces do objeto são feitos;

4. Por fim, são gerados os descritores a partir da distância de cada intersecção até o centro da esfera. Essas distâncias são normalizadas entre [0, 1], sendo 1 o correspondente da maior distância encontrada.

Figura 4 – Esfera circunscrita a um cubo.



Fonte: Sales (2017).

3.2 Cálculo das distâncias

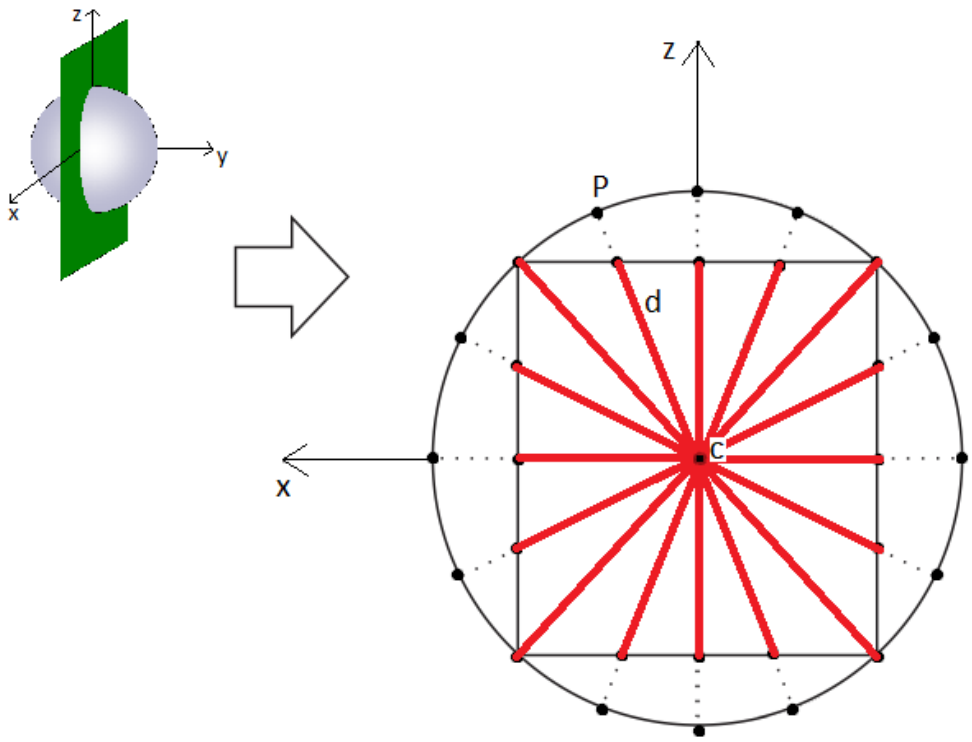
Para o cálculo das distâncias, é necessário escolher um local no objeto, para que seja posicionado o centro da esfera. Pela abordagem original do algoritmo, o centro da esfera é o centro de massa do objeto tridimensional. Para objetos tridimensionais, é possível o cálculo deste centro, analisando os triângulos de suas malhas.

O centro de massa C de um objeto qualquer é dado pela Equação 3.1, sendo A_i a área de um triângulo i qualquer representando parte da malha do objeto, A_t o somatório de todas as áreas e R_i o baricentro do triângulo i, determinado pela Equação 3.2 com a_i, b_i e c_i os vértices do triângulo i.

$$C = \frac{\sum_{i=0}^{N-1} A_i * R_i}{A_T} \tag{3.1}$$

$$R_i = \frac{a_i + b_i + c_i}{3} \tag{3.2}$$

Figura 5 – Secção da esfera, com as distâncias dos segmentos entre intersecções e centro da esfera.



Fonte: Sales (2017).

No método 3D-CSD, o centro de massa do objeto analisado é o centro da esfera; e o raio, a maior distância entre o centro de massa e todos os vértices do objeto. É importante notar que esse cálculo de centro de massa só pode ser utilizado para poliedros com faces triangulares não degeneradas, i.e., faces triangulares com áreas não nulas.

As características do objeto são definidas como a distância euclidiana entre os vértices de referência contidos na superfície e seus respectivos vértices de intersecção contidos nas faces do objeto, caso ele exista. Se o vértice de referência não gera uma intersecção, é atribuído o valor -1 a essa característica.

O Algoritmo 1 descreve a estratégia de verificação de intersecções, apresentado em (JIMéNEZ; SEGURA; FEITO, 2010).

Todas as distâncias geradas pelo algoritmo abaixo formam o conjunto de características do objeto. Originalmente, esse conjunto de características é rotulado e passado como dados de treino para uma rede *Multi Layer Perceptron*, que aprende como identificar diferentes objetos apresentados durante a etapa de treino (SALES; AMARO; OSÓRIO, 2017).

Vale mencionar que outra característica do 3D-CSD é a busca exaustiva das intersecções, i.e., todos os vértices de referência são testados com todas as faces do objeto, gerando um sério problema de escalabilidade do algoritmo.

Algoritmo 1 – Verificação da existência de intersecção entre uma face de um objeto e um segmento entre o centro e a superfície da esfera.

$\overrightarrow{a} \leftarrow SegmentVertex1 - ObjectTriangleVertex3$
$\overrightarrow{b} \leftarrow ObjectTriangleVertex1 - ObjectTriangleVertex3$
$\overrightarrow{c} \leftarrow ObjectTriangleVertex2 - ObjectTriangleVertex3$
$\overrightarrow{d} \leftarrow SegmentVertex2 - ObjectTriangleVertex3$
$\overrightarrow{w_1} \leftarrow \overrightarrow{b} \times \overrightarrow{c}$
$w \leftarrow \overrightarrow{a} \cdot \overrightarrow{w_1}$
$s \leftarrow \overrightarrow{d} \cdot \overrightarrow{w_1}$
$\overrightarrow{w_2} \leftarrow \overrightarrow{a} \times \overrightarrow{d}$
$t \leftarrow \overrightarrow{w_2} \cdot \overrightarrow{c}$
$u \leftarrow -(\overrightarrow{w_2} \cdot \overrightarrow{b})$
$if\ (w > 0)$
  $if\ (s > 0\ ||\ t < 0\ ||\ u < 0\ ||\ (s+t+u) > w)$
    $return\ IntersectionFail$
$else\ if\ (w < 0)$
  $if\ (s < 0\ ||\ t > 0\ ||\ u > 0\ ||\ w > (s+t+u))$
    $return\ IntersectionFail$
$else$
  $if\ (s > 0)$
    $if\ (t < 0\ ||\ u < 0\ ||\ (s+t+u) > 0)$
      $return\ IntersectionFail$
  $else\ if\ (s < 0)$
    $if\ (t > 0\ ||\ u > 0\ ||\ -(s+t+u) > 0)$
      $return\ IntersectionFail$
  $else$
    $return\ IntersectionFail$
$tParam \leftarrow -(w/(s-w))$
$\overrightarrow{direction} \leftarrow SegmentVertex2 - SegmentVertex1$
$intersection \leftarrow SegmentVertex1 + (tParam \cdot \overrightarrow{direction})$
$return\ IntersectionSuccess$

3.3 Benchmark e adversidades

Em testes registrados em (SALES, 2017), usando um processador i7-3520M, aferiu-se um tempo de aproximadamente 3 segundos para calcular todas as distâncias de um objeto com aproximadamente 33.994 faces, em uma esfera com 326 vértices de referência.

Apesar de *datasets* geralmente apresentarem objetos com menor volume de faces, o custo computacional desta etapa é linear. Isso quer dizer que, para cada 100ms de processamento, seriam processadas aproximadamente apenas 1.113 faces na configuração de esfera supracitada, um número bem inferior aos que geralmente aparecem nos *datasets* de objetos reais e sintéticos.

Além do problema relacionado ao custo computacional ligado ao número de faces, viu-se uma limitação no conjunto de chaves que seria utilizado para identificar o objeto: os descritores (distâncias euclidianas entre cada vértice de intersecção e centro da esfera) precisam ser inseridos

de maneira ordenada na rede neural, sendo reconhecida apenas uma pose específica do objeto.

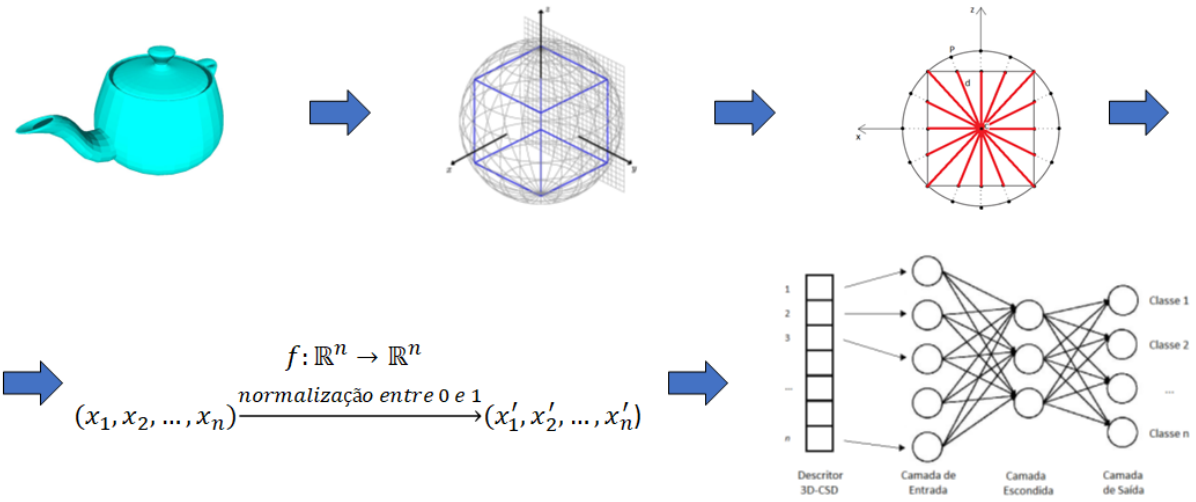
Para se ter a invariância à rotação do objeto, é necessário rotacionar o objeto em poucos graus em torno do eixo de altura, gerando variações de cada objeto do conjunto de treino. Isso faz com que o tempo da etapa de treino cresça linearmente com a quantidade de rotações do objeto, aumentando mais ainda o custo computacional.

Considerando as limitações de custo computacional e de tempo de processamento, pode-se dizer que a abordagem original proposta para a implementação do método 3D-CSD praticamente inviabiliza o seu uso em sistemas embarcados ou com limitações de capacidade de processamento/consumo de energia, ou seja, quando não é feito o uso de GPUs ou de aceleradores de processamento paralelo mais sofisticados. Em função disto, foi proposta uma nova abordagem, o 3D-CSD+ que visa aperfeiçoar a abordagem 3D-CSD original, com um significativo ganho de desempenho.

3.4 Fluxograma

Um ilustração com o fluxo de execução do algoritmo 3D-CSD pode ser vista logo abaixo, com os passos desde a obtenção da malha do objeto até sua classificação. Em destaque na cor vermelha, estão as distâncias de cada segmento entre o centro da esfera e a casca da malha, que serão normalizadas e servir de entrada para a rede neural.

Figura 6 – Fluxograma do algoritmo original 3D-CSD.



Fonte: Elaborada pelo autor.

CAPÍTULO

4

3D-CSD+: UMA NOVA ABORDAGEM

Visando resolver/minimizar os principais problemas elucidados no capítulo anterior, é apresentada uma nova abordagem baseada no algoritmo 3D-CSD, tentando aproveitar seus pontos fortes e mitigar suas limitações. Esta nova abordagem foi denominada de 3D-CSD "plus"(3D-CSD+).

4.1 Representação do Objeto

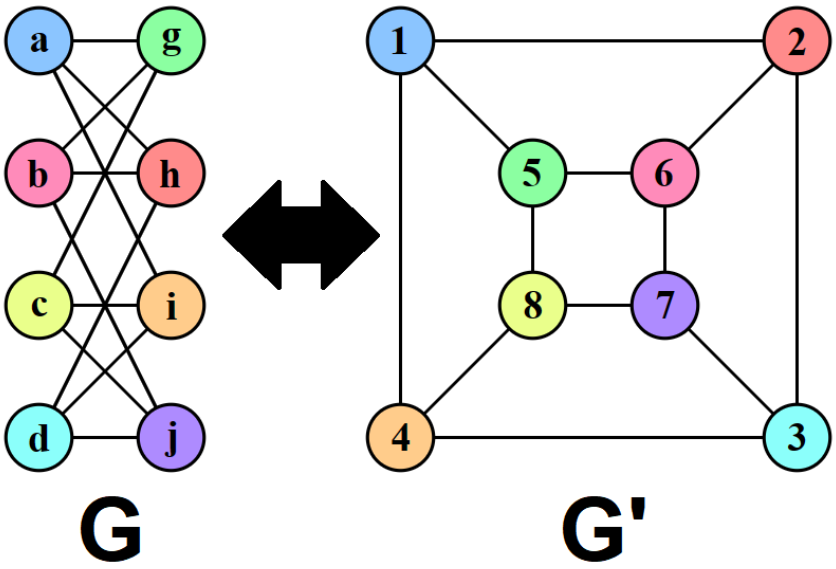
Tendo em vista os problemas citados no capítulo anterior ao se usar um vetor de distâncias como descritor de um objeto, foram estudadas outras formas de representar um objeto descrito pelo 3D-CSD, mas mantendo a sua vantagem de descrever o objeto de maneira parametrizada com uma esfera de referência. A mais proeminente técnica utilizada foi a representação do objeto por meio de um grafo gerado pelos vértices de intersecções. Essa representação faz com que o problema de classificação de objeto seja interpretado como um problema de isomorfismo entre grafos (Fig. 7).

De acordo com CORMEN *et al.*, isomorfismo entre dois grafos é definido como: Dados dois grafos $G = (V,E)$ e $G' = (V',E')$, deve existir uma bijeção $f : V \to V'$, tal que $(u,v) \in E \iff (f(u), f(v)) \in E'$. Ou seja, um grafo é isomorfo a outro se for possível mapear todos os nós de um grafo em outro, mantendo todos os caminhos entre quaisquer nós coerentes nos dois grafos.

Em uma primeira análise, isso tende a ser uma ideia pouco interessante, pois o problema de isomorfismo é resolvido em tempo quasi-polinomial: $2^{O((logn)^3)}$, como demonstrado por BABAI e confirmado por HELFGOTT. Além disso, é necessário buscar um algoritmo que estabeleça o quão "parecido" um grafo seria de outro, não uma correspondência exata; então, ajustes precisariam ser feitos ao algoritmo.

Foram testados métodos que descrevessem o grafo, ao mesmo tempo que fosse possível

Figura 7 – Exemplo de grafos isomorfos.



Fonte: Wikipedia (a), Wikipedia (b).

reduzir a quantidade de dados que descreve um objeto qualquer, como:

- Produtório dos autovalores não nulos da matriz de adjacência ou matriz laplaciana;
- Traço da matriz de adjacência ou matriz laplaciana;
- Autovalores da matriz de adjacência ou matriz laplaciana.

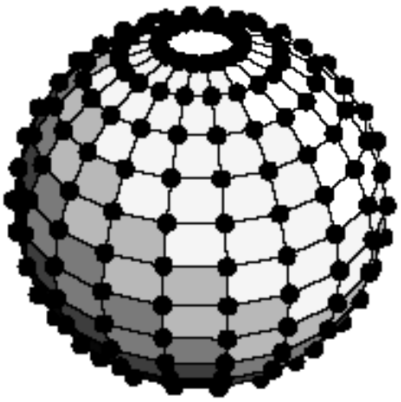
Os dois primeiros métodos possuem a capacidade de reduzir todo o objeto a apenas um número. Porém os mesmos métodos lidam muito mal com objetos em pequena escala e com grande número de vértices. No caso do produtório, pequenas escalas geram parcelas muito próximas de zero, podendo até causar um resultado subnormal, o que é piorado em objetos com alta densidade de vértices. Para o traço, cada parcela trazia consigo algum erro numérico do algoritmo, que era acumulado até defasar o resultado em até um terço do esperado.

O último método aumenta a descritividade do objeto, mas aumenta o conjunto de números linearmente com a quantidade de vértices de referência na esfera. Mas é possível perceber com maior facilidade como o autovalor se modifica, conforme o objeto é modificado. Sobre qual matriz utilizar como entrada no cálculo dos autovalores, não se viu grande diferença nos experimentos, ao utilizar a matriz de adjacência ou a laplaciana. Porém, por uma questão de maior presença no objetivo do cálculo de similaridade, viu-se a recomendação do uso da matriz laplaciana na literatura (KOUTRA *et al.*, 2011).

4.2 Esfera

Outro aspecto importante, porém pouco discutido, do algoritmo 3D-CSD foi a questão da distribuição dos vértices da esfera (Fig. 8) . No algoritmo original, era utilizada apenas uma esfera com vértices de referência gerados por meio de um mapeamento UV em intervalos regulares nos eixos U e V. Isso fazia com que a esfera concentrasse muitos vértices nos polos, gerando maior concentração de vértices de intersecção em uma área; que, na maioria das vezes, não necessitava de grande descritividade, pois criações humanas tendem a ter mais detalhes nas áreas laterais de um objeto, pois estão geralmente a altura dos olhos.

Figura 8 – Esfera utilizada originalmente no algoritmo 3D-CSD.



Fonte: Sales (2017).

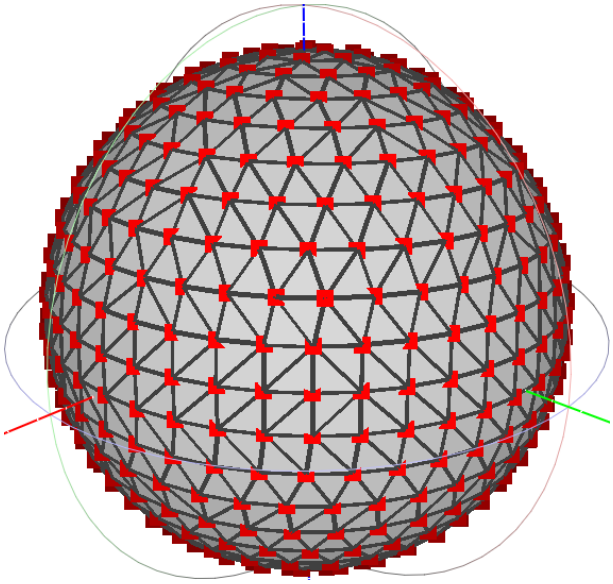
Para evitar esse problema, buscou-se por algoritmos que pudessem gerar esferas com uma melhor distribuição de seus vértices. Dentre as esferas encontradas, pode-se listar:

- *Distributing many points on a sphere* (SAFF; KUIJLAARS, 1997): Técnica que tenta tesselar a esfera regularmente com hexágonos, com exceção de alguns pentágonos. Esta técnica visa reduzir a quantidade de energia na disposição dos vértices na esfera, sendo possível a geração com quaisquer quatro ou mais vértices (Fig. 9).

- *How to generate equidistributed points on the surface of a sphere* (DESERNO, 2004): Regularmente coloca vértices na esfera, sendo que a distância em duas direções ortogonais seja localmente a mesma. Não é possível usar esta técnica para gerar uma esfera de qualquer quantidade de vértices, mas existem poucas restrições de quantidades de vértices (Fig. 10).

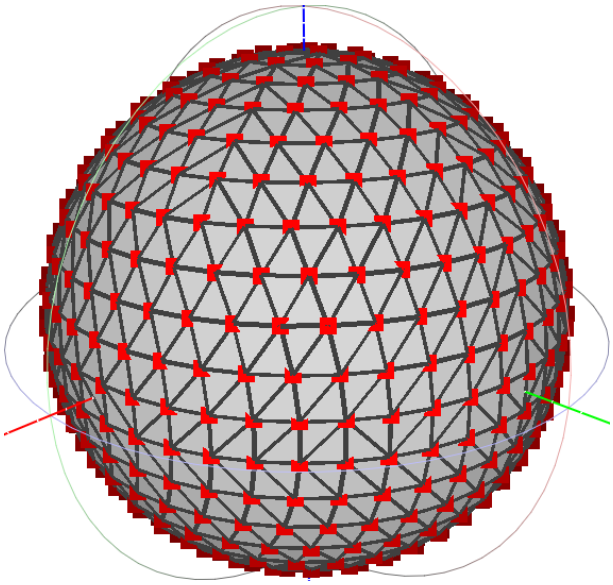
- *HEALPix* (ROUKEMA; CALABRETTA, 2007): Técnica que particiona uma esfera em regiões harmônicas de tamanhos iguais. Para esta técnica, também foi utilizada uma versão da esfera que é composta apenas pela parte do algoritmo que gera os vértices centrais de cada um dos harmônicos. Não é possível usar esta técnica para gerar uma esfera de qualquer quantidade de vértices (Fig. 11 e 12).

Figura 9 – Esfera feita pelo algoritmo de Saff e Kuijlaars.



Fonte: Elaborada pelo autor.

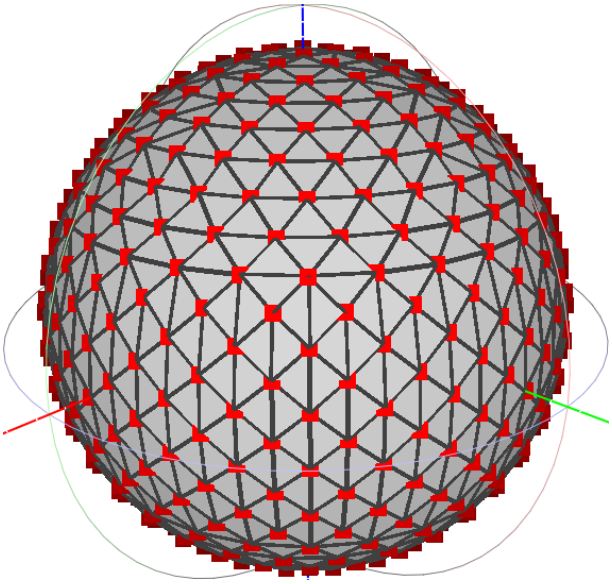
Figura 10 – Esfera feita pelo algoritmo de Deserno.



Fonte: Elaborada pelo autor.

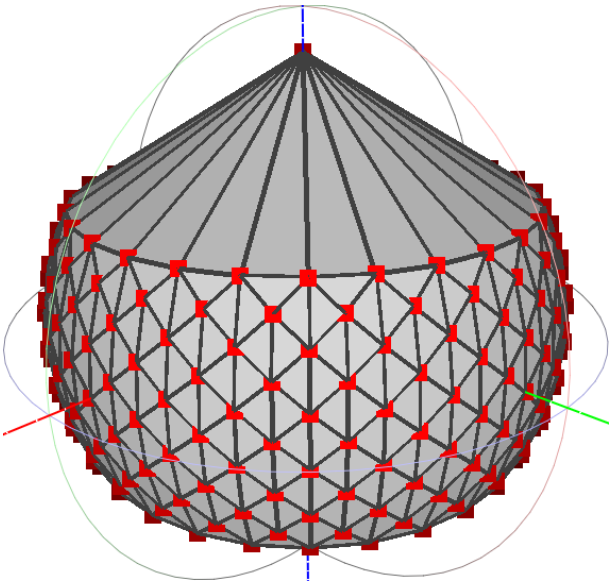
- *A New Computationally Efficient Method for Spacing n Points on a Sphere* (KOGAN, 2017): O método introduz uma técnica baseada em espirais encontradas em evidências empíricas. É possível o uso de quaisquer quatro ou mais vértices na geração da esfera (Fig. 13).

Figura 11 – Esfera feita pelo algoritmo HEALPix.



Fonte: Elaborada pelo autor.

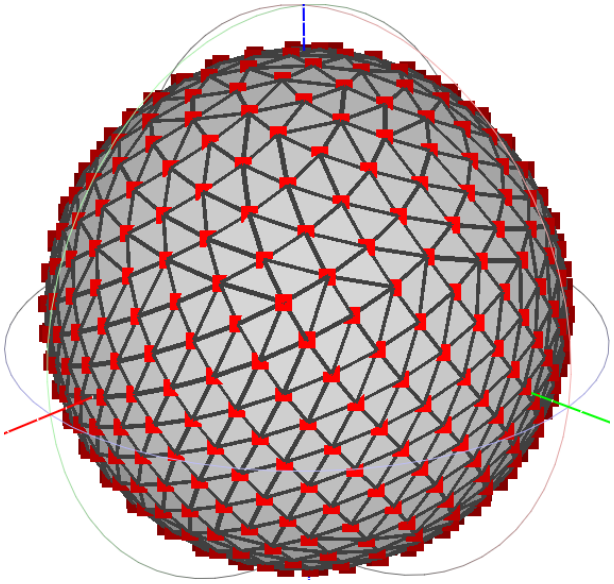
Figura 12 – Parte equatorial da esfera feita pelo algoritmo HEALPix.



Fonte: Elaborada pelo autor.

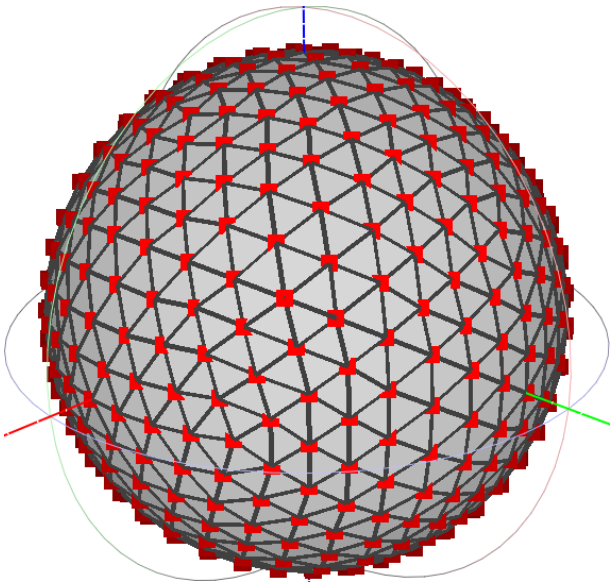
- *Ico-Sphere* (BLENDER, 2019): A partir das coordenadas de cada vértice dos triângulos de um icosaedro, são criados recursivamente novos vértices usando pontos médios das arestas, sendo projetados na casca da esfera (Fig. 14). Esta é a técnica com a maior restrição, seguindo a sequência 12, 42, 162, 642, 2562...

Figura 13 – Esfera feita pelo algoritmo de Kogan.



Fonte: Elaborada pelo autor.

Figura 14 – Esfera feita pelo algoritmo baseado em icosaedro.



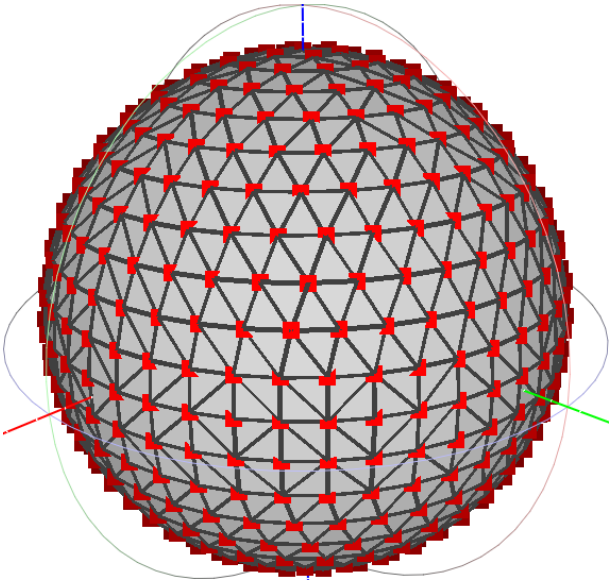
Fonte: Elaborada pelo autor.

- *Disco-Ball*: De uma dada altura do meridiano, vértices são distribuídos com distâncias iguais entre si no paralelo definido pelo corte do meridiano (Fig. 15). Como alguns dos algoritmos citados acima, não é possível criar uma esfera com quaisquer números de vértices na casca (Disco-Ball [1])

Para cada distribuição de vértices gerada, é aplicado o algoritmo de *Convex Hull*, também

[1] Disco-Ball: http://web.archive.org/web/20110806123740/http://www.math.niu.edu/~rusin/known-math/97/disco

Figura 15 – Esfera feita pelo algoritmo de Rusin.



Fonte: Elaborada pelo autor.

chamado de Fecho Convexo, para definir como os vértices de intersecção irão se conectar entre si. O *Convex Hull* é o menor conjunto de vértices usados para formar um fecho convexo em torno de um conjunto de vértices qualquer.

A escolha do *Convex Hull* se deu pela motivação de ter uma forma de conectar todos os vértices da esfera, representada pelas arestas pretas nas figuras acima; com a possibilidade de se ter um objeto reduzido visualmente parecido com seu objeto original, facilitando a compreensão humana de como o algoritmo estaria interpretando cada objeto. Apesar desta escolha, não existem impeditivos para que hajam conexões feitas manualmente, i.e., outro arranjo de conexões pode ser feito, criando conexões em locais de interesse de um dado objeto.

Por fim, houve também a mudança do algoritmo utilizado para circunscrever a esfera no objeto. Nesta nova abordagem, em vez de utilizar o cálculo de centro de massa, usa-se o algoritmo de FISCHER; GÄRTNER; KUTZ, que visa produzir a menor esfera possível para um conjunto de pontos em quaisquer dimensões.

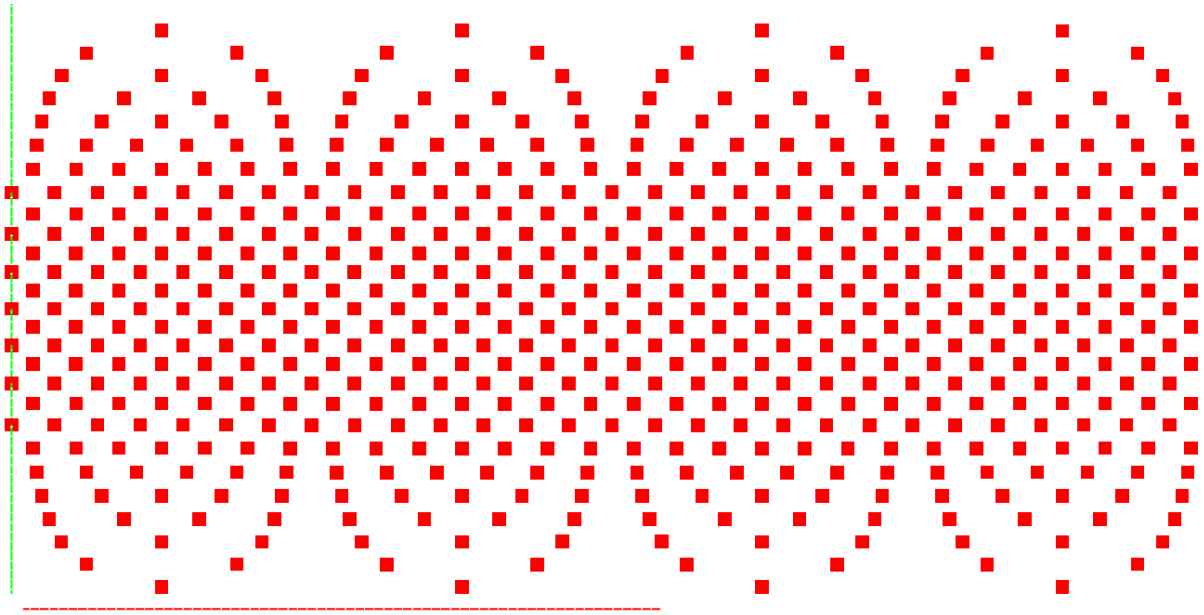
4.3 Busca por intersecções

Mesmo apresentando melhorias no modo de interpretar o objeto, buscando uma invariância a rotação em qualquer eixo, o problema da busca exaustiva das intersecções ainda é presente. Para atacar este problema, reduziu-se a questão das intersecções a uma busca no espaço $\mathbb{R}^2$. Esta abordagem é possível pela conversão das coordenadas cartesianas em coordenadas polares e pelo fato de todos os segmentos gerados terem o centro da esfera como vértice em comum.

O processo para que essa operação possa acontecer pode ser descrito pelos passos:

1. Conversão de todos os vértices contidos na casca da esfera para coordenadas polares, observando os eixos azimutal e polar (Fig. 16)

Figura 16 – Esfera HEALPix convertida para coordenadas polares.



Fonte: Elaborada pelo autor.

2. Conversão de todas as faces do objeto para coordenadas polares, observando os eixos azimutal e polar; e considerando que a informação de raio é igual ao raio da esfera (Fig. 17 e 18)

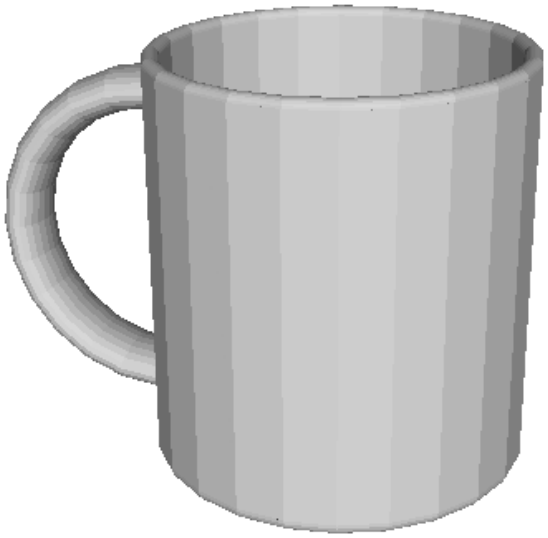
3. Definição de uma região de interesse para cada triângulo da face (Fig. 19).

   - Como a projeção esférica de um triângulo será sempre deformada, não é possível utilizar um simples algoritmo de verificação de um vértice estar dentro de um triângulo; pois, novamente, as arestas do triângulo não são retas na projeção.
   - Para evitar esse problema, faz-se uma caixa de contenção (*bounding box*) em volta dos vértices do triângulo. Isso garante que o triângulo esteja completamente contido em um polígono maior, no caso, um retângulo.
   - Apesar de acabar potencialmente inserindo mais vértices da esfera para serem testados, agora com os retângulos, essa etapa reduz drasticamente a quantidade de testes para cálculo de intersecção de volta no plano $\mathbb{R}^3$.

4. Sobreposição das regiões de interesse e conversão da esfera em um plano tendo as coordenadas azimutais e polares (Fig. 20).

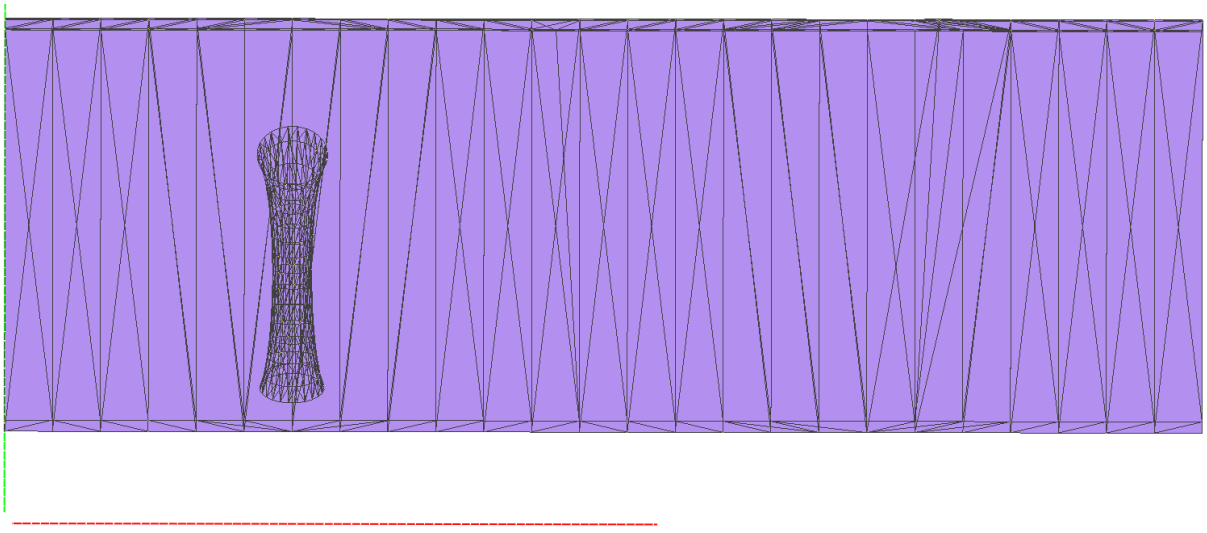
5. Mapeamento das projeções dos vértices da esfera em cada uma das regiões de interesse.

Figura 17 – Objeto 3D cup_0018 do dataset ModelNet40.



Fonte: Elaborada pelo autor.

Figura 18 – Faces do objeto convertidas para coordenadas polares.



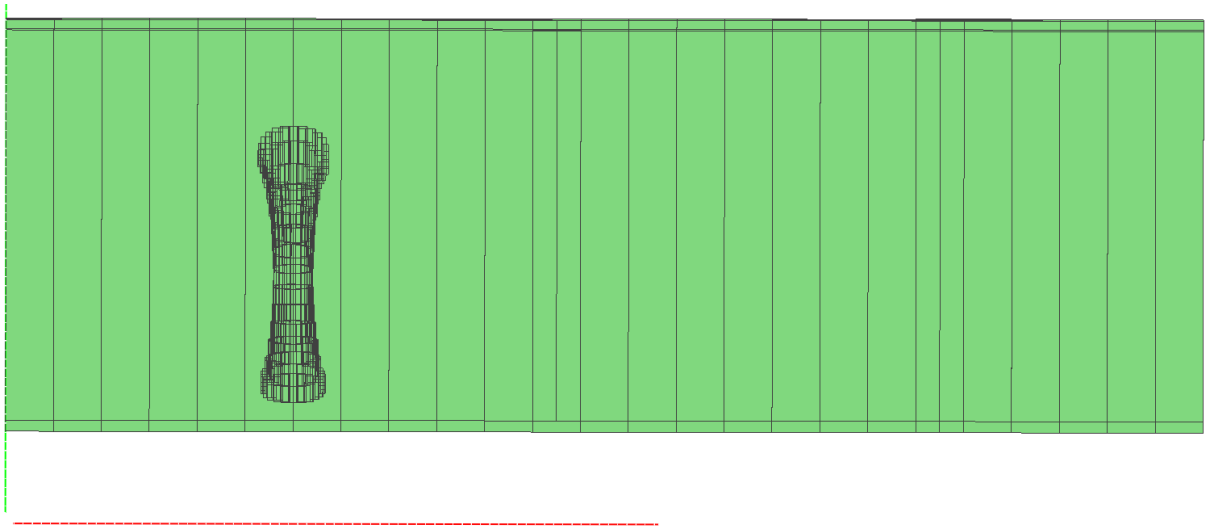
Fonte: Elaborada pelo autor.

6. Correlação do mapeamento no plano $\mathbb{R}^3$.

7. Teste de intersecção.

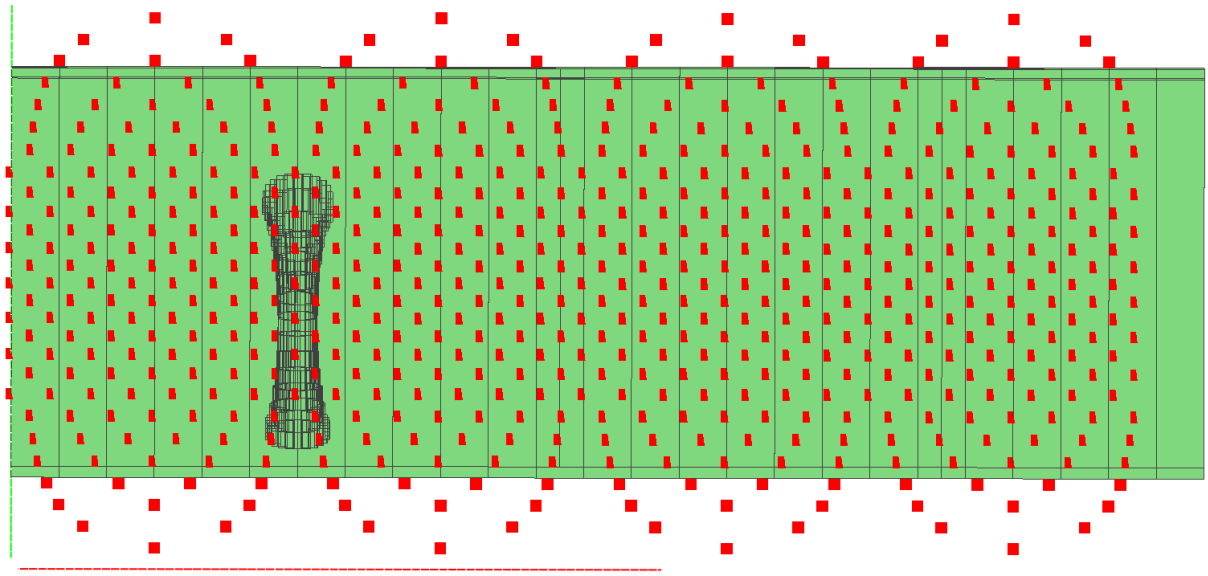
Para regiões em que esse teste não é possível, como triângulos contendo polos (1) e triângulos "na borda"do plano (azimute igual a 0 ou 2π) (2), usa-se as seguintes abordagens para contornar tais problemas:

Figura 19 – Regiões de interesse de cada face do objeto.



Fonte: Elaborada pelo autor.

Figura 20 – Sobreposição das regiões de interesse nos vértices da esfera.



Fonte: Elaborada pelo autor.

1. Identificação da coordenada polar do triângulo mais distante do polo e definição desta coordenada como limiar do mapeamento. Todo vértice da esfera que estiver do polo até esse limiar será inserido no grupo para teste de intersecção.

2. A região de interesse é duplicada, com uma rotação de 2π no eixo azimutal, e todos os vértices que estiverem dentro de qualquer uma das duas regiões serão adicionados no grupo para teste de intersecção.

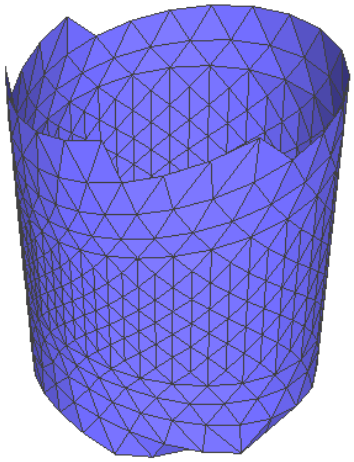
O algoritmo para teste de intersecção no plano $\mathbb{R}^3$ ainda é o mesmo utilizado no algoritmo 3D-CSD, como descrito no capítulo anterior.

4.4 Criação do grafo

Tendo todas as intersecções calculadas, faz-se a separação dessas intersecção em dois grupos:

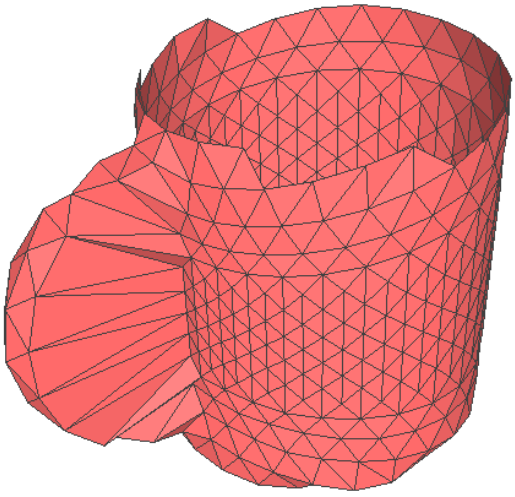
- Intersecções próximas ao centro da esfera, sendo ilustrado um exemplo na Fig. 21.
- Intersecções próximas à casca da esfera, sendo ilustrado um exemplo na Fig. 22.

Figura 21 – Abstração do objeto, usando as intersecções mais próximas do centro da esfera.



Fonte: Elaborada pelo autor.

Figura 22 – Abstração do objeto, usando as intersecções mais próximas da casca da esfera.



Fonte: Elaborada pelo autor.

Depois, são carregadas as vizinhanças de cada vértice, de acordo com cada aresta gerada pelo Fecho Convexo na esfera, gerando dois grafos: um para o grupo de intersecções mais

próximos do centro da esfera e outro para o grupo mais distante. Tendo os dois grafos, pode-se seguir por duas abordagens do algoritmo, sendo N a quantidade de vértices na esfera:

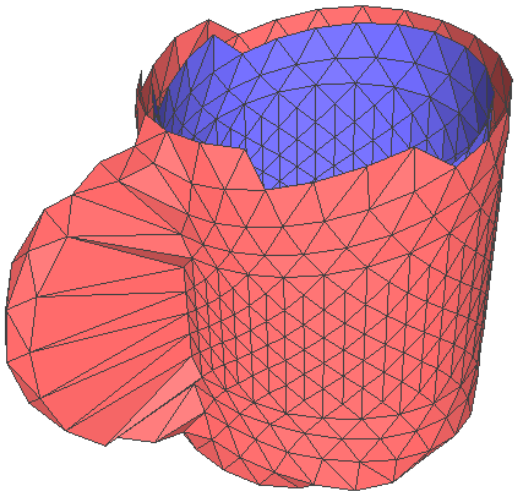
- Uso apenas da camada mais próximas da casca da esfera, gerando uma matriz laplaciana $M_{(N+1)\times(N+1)}$. Para esta abordarem, será utilizado o termo "abordagem externa".

- Uso das duas camadas, juntando-as em uma matriz laplaciana $M_{2(N+1)\times 2(N+1)}$. Para esta abordarem, será utilizado o termo "abordagem completa".

A linha e coluna do $+1$ nas matrizes são preenchidas com as distâncias de cada intersecção até a âncora da esfera; que, neste caso, é o seu centro. Essa distância até o centro é uma forma de inserir intersecções que não possuem vizinhos, coletando a maior quantidade de informações disponível do grafo.

A abordagem completa possui o diferencial de detectar algumas concavidades no objeto, podendo descrevê-las no mesmo conjunto de chaves de autovalores. No exemplo da caneca mostrado na Fig. 23, o grafo gerado a partir da abstração em azul é capaz de informar quão espessa é a parede da caneca, podendo ter até esse refino na descrição de um objeto.

Figura 23 – Abstração do objeto, usando todas as intersecções.



Fonte: Elaborada pelo autor.

Independentemente da abordagem utilizada, para extrair as chaves que descrevem o objeto, é necessário apenas aplicar um algoritmo que gere um conjunto de autovalores ordenados de uma matriz laplaciana. Será necessário aplicar um algoritmo de ordenação nas chaves, caso o algoritmo de geração de autovalores já não faça isso em sua saída.

4.5 Classificação

Finalmente, para saber o quão parecido um objeto é de outro objeto, deve ser feita apenas uma comparação entre as duas distribuições geradas pelos autovalores. Para cada posição i do conjunto de autovalores, faz-se o valor absoluto da diferença entre cada autovalor da posição, somando todos esses valores absolutos. Essa soma final representa a diferença entre os dois objetos.

Uma característica que é importante ser explicitada é a capacidade de controlar quais amostras serão inseridas na base com todo o conhecimento de descritores gerados pelo algoritmo. Isso elimina qualquer necessidade de retreino, caso uma amostra esteja fazendo o algoritmo classificar erroneamente outros objetos, já que bastaria apenas remover o conjunto de autovalores dessa amostra da base de conhecimento.

E, assim como no caso da eliminação, a transferência de conhecimento também se baseia apenas na cópia de conjunto de autovalores de uma base de conhecimento para outra. Isso traz a possibilidade de aplicações adquirirem novos conhecimentos, ou remover conhecimentos indesejados, de maneira simples e até mesmo em tempo de execução.

É importante denotar que a classificação apresentada neste projeto não utiliza redes neurais como ferramenta de classificação. A classificação aqui é feita apenas com uma função de distância, que calcula a similaridade entre dois conjuntos de descritores.

Apesar disso, existe a possibilidade de tais conjuntos serem utilizados como parâmetros de redes neurais para a classificação, semelhante ao que ocorre na abordagem original do 3D-CSD. Contudo, essa mudança trará todas as vantagens e desvantagens adquiridas com o uso de tal técnica. Como há o intuito do algoritmo ser *hardware agnostic*, optou-se pelo seu não uso.

4.6 Autovalores

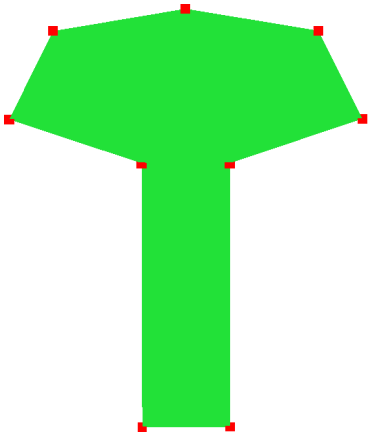
Para uma melhor elucidação do leitor, será dado um exemplo de como os autovalores se comportam, com cinco objetos contidos em $\mathbb{R}^2$. Três objetos são considerados similares e possuem silhuetas semelhantes a de árvores (Fig. 24, Fig. 25 e Fig. 26), enquanto o quarto e quinto objetos são considerados de classe distinta da anterior e possuem uma silhueta que lembra uma lua crescente. Também é adicionada a particularidade de que as duas silhuetas de luas são espelhadas uma da outra nos eixos X e Y (Fig. 27 e Fig. 28).

Como é pré-requisito um número fixo do grafo para todos os objetos, cada silhueta possui exatamente nove pontos dispostos no espaços, sendo representados por marcadores vermelhos em cada uma das silhuetas.

A lista de coordenadas de cada ponto pode ser aferida na Tabela 1.

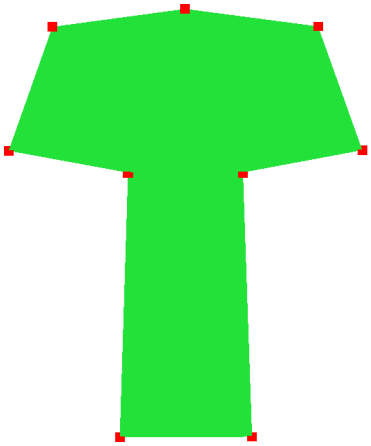
Calculando as distâncias de todos os vértices vizinhos (i) e $(i+1)$, é possível criar a

Figura 24 – Silhueta da árvore 1.



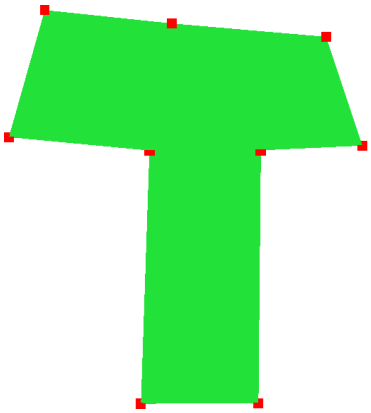
Fonte: Elaborada pelo autor.

Figura 25 – Silhueta da árvore 2.



Fonte: Elaborada pelo autor.

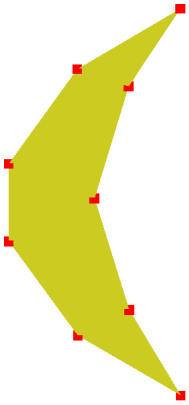
Figura 26 – Silhueta da árvore 3.



Fonte: Elaborada pelo autor.

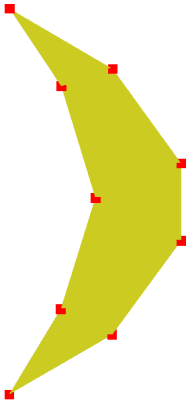
matriz laplaciana de cada um dos grafos. Com a matriz laplaciana construída, pode-se seguir com a geração dos autovalores de cada um dos grafos.

Figura 27 – Silhueta da lua 1.



Fonte: Elaborada pelo autor.

Figura 28 – Silhueta da lua 2.



Fonte: Elaborada pelo autor.

Os autovalores em questão, já ordenados de maneira decrescente, e podem ser vistos na Tabela 2.

Após o cálculo dos autovalores, faz-se o uso da distância L_1 entre cada um dos conjuntos de autovalores, gerando a Tabela 3.

Como é possível observar claramente pela tabela, houve um acerto de 100% na busca pela menor distância. Todos as silhuetas de árvore tiveram a menor distância com outra árvore que está contida dentro da mesma categoria. E, como era previsto, as duas silhuetas de lua tiveram um *match* perfeito entre elas, já que elas possuem as mesmas distâncias entre vértices, gerando os mesmos autovalores.

A ideia apresentada aqui é a que está sendo utilizada nesta dissertação, porém em $\mathbb{R}^3$ e usando a superfície da esfera circunscrita como uma forma de normalizar a quantidade de vértices de cada um dos objetos analisados.

Tabela 1 – Coordenadas de cada silhueta.

Coordenadas	Árvore 1	Árvore 2	Árvore 3	Lua 1	Lua 2
Vértice 1	(0,5; 0)	(0,75; 0)	(0,65; 0,3)	(0; 0)	(0; 0)
Vértice 2	(0,5; 3)	(0,65; 3)	(0,68; 3,1)	(-0,6; 1)	(0,6; -1)
Vértice 3	(2; 3,5)	(2; 3,25)	(1,8; 3,15)	(-1; 2.3)	(1; -2.3)
Vértice 4	(1,5; 4,5)	(1,5; 4,65)	(1,4; 4,35)	(-0,6; 3,6)	(0,6; -3,6)
Vértice 5	(0; 4,75)	(0; 4,85)	(-0,3; 4,5)	(0; 4,5)	(0; -4,5)
Vértice 6	(-1,5; 4,5)	(-1,5; 4,65)	(-1,7; 4,65)	(-1,2; 3,8)	(1,2; -3,8)
Vértice 7	(-2; 3,5)	(-2; 3,25)	(-2,1; 3,25)	(-2; 2,7)	(2; -2,7)
Vértice 8	(-0,5; 3)	(-0,65; 3)	(-0,55; 3,1)	(-2; 1,8)	(2; -1,8)
Vértice 9	(-0,5; 0)	(-0,75; 0)	(-0,65; 0,3)	(-1,2; 0,7)	(1,2; -0,7)

Fonte: Dados da pesquisa.

Tabela 2 – Autovalores de cada silhueta.

Autovalores	Árvore 1	Árvore 2	Árvore 3	Lua 1	Lua 2
Autovalor 1	8,04	8,44	7,84	5,02	5,02
Autovalor 2	7,23	7,11	6,70	4,87	4,87
Autovalor 3	5,19	5,50	5,47	4,11	4,11
Autovalor 4	3,96	4,56	4,33	3,46	3,46
Autovalor 5	2,81	2,92	2,75	2,11	2,11
Autovalor 6	2,24	2,42	2,29	2,01	2,01
Autovalor 7	0,68	0,73	0,77	0,60	0,60
Autovalor 8	0,68	0,81	0,68	0,56	0,56
Autovalor 9	0	0	0	0	0

Fonte: Dados da pesquisa.

Tabela 3 – Distância L_1 entre cada conjunto de autovalores.

	Árvore 1	Árvore 2	Árvore 3	Lua 1	Lua 2
Árvore 1	**X**	1.96	1.63	8.15	8.15
Árvore 2	1.96	**X**	1.67	9.76	9.76
Árvore 3	1.63	1.67	**X**	8.10	8.10
Lua 1	8.15	9.76	8.10	**X**	0
Lua 2	8.15	9.76	8.10	0	**X**

Fonte: Dados da pesquisa.

4.7 Análise de complexidade

Como mencionado anteriormente, o encontro dos vértices de intersecção requeria um tempo $O(n \cdot m)$, sendo n a quantidade de vértices na esfera e m a quantidade de faces do objeto, para quaisquer casos. Para a otimização deste tempo, usa-se o fato de os objetos geralmente não terem grandes concavidades que deformem a superfície para dentro do objeto. Como a leitura do ambiente por meio de sensores convencionais não resultam em tais adversidades, as projeções das faces na superfície da esfera tendem a não ter sobreposições. Isso faz com que seja feito

aproximadamente um teste de descoberta de vértices de intersecção para cada vértice da esfera.

Aplicando as mudanças descritas anteriormente, analisa-se o efeito assintótico do pior caso do algoritmo. Como é possível imaginar, não existem mudanças no Big-O da quantidade de testes para encontrar os vértices de intersecção de um objeto qualquer. Mas pela disposição das projeções de um objeto capturado no mundo real, são feitas exatamente n projeções dos vértices das faces na superfície da esfera, com $O(2 \cdot n \cdot \log_2 n + m)$ testes para encontrar a região de interesse na esfera e verificar se existe um ponto naquela região, e por volta de n testes de encontro de pontos de intersecção no plano tridimensional.

Com o grafo já estruturado, parte-se para a geração do autoespaço. Essa etapa possui a complexidade $O(\frac{4}{3} \cdot n^3)$ (GOLUB; LOAN, 1996) para o cálculo dos autovalores e $O(9 \cdot n^3)$ (GOLUB; LOAN, 1996) para o cálculo do conjunto dos autovalores e autovetores. Essa complexidade é adquirida devido a conversão da matriz laplaciana para uma matriz de tipo Hessenberg e posterior decomposição QR, otimizada para matrizes simétricas.

Uma vez gerados os autovalores, a distância deste conjunto de autovalores precisa ser calculada com todos os outros conjuntos já presentes na base de dados criada durante o treino. Apesar da possibilidade de paralelização desta busca, a complexidade seria $O(n \cdot k)$, sendo k a quantidade de amostras aprendidas na etapa de treino. Contudo, é possível utilizar uma árvore R (GUTTMAN, 1984) nesta busca, podendo reduzir o custo para até $log_k n$ para o caso médio, com pior caso sendo linear em n. Com o uso desta árvore, é possível ao interpretar os autovalores como coordenadas espaciais e buscar pontos de interesses em volta do centro de busca. Funcionando de maneira similar, também poderia ser utilizada uma árvore M (CIACCIA; PATELLA; ZEZULA, 1997), que possui a restrição de usar distâncias com desigualdade triangular, restrição que é satisfeita neste algoritmo.

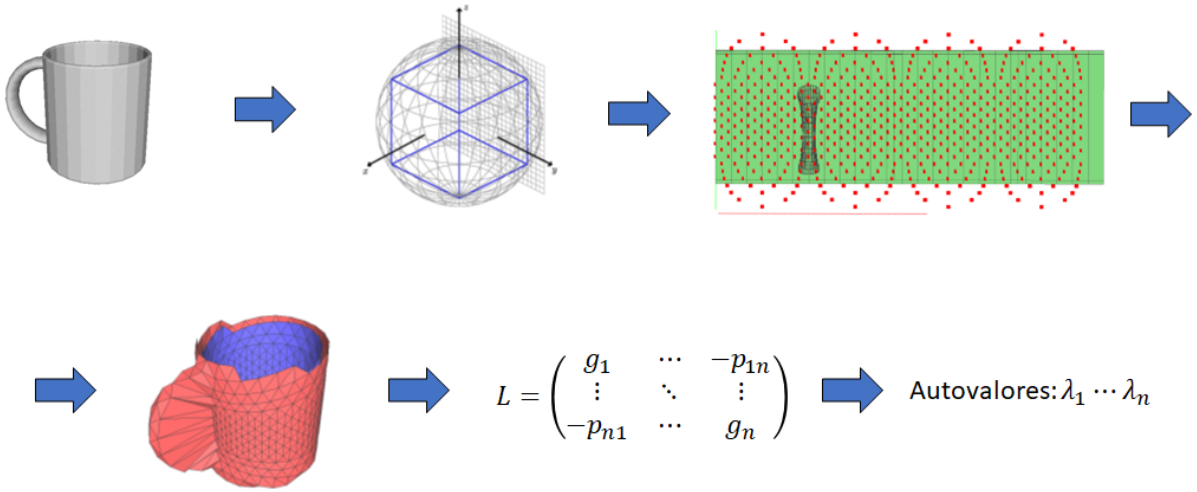
Como é possível observar, o ponto que pode se tornar um gargalo rapidamente para o uso do método apresentado neste texto é o cálculo de autovalores, que possui complexidade polinomial cúbica. Esse comportamento foi observado durante os testes.

4.8 Considerações Finais

Este Capítulo apresentou as melhorias implementadas no descritor 3D-CSD, sendo proposta uma nova abordagem denominada de 3D-CSD+. Foi verificada uma melhora na complexidade, em gargalos presentes no código original, e consequente diminuição do tempo de execução, bem como acrescentada a invariância à rotação em qualquer eixo, limitada anteriormente apenas ao eixo de altura (Z). No próximo capítulo, são demonstradas novas formas de cálculo do centro da esfera para que o objeto seja melhor circunscrito e preparar o descritor modificado para lidar com variâncias a escala. Desta forma, é esperada a obtenção de um descritor de objetos 3D robusto e um método de classificação factível com aplicações limitadas computacionalmente.

Por fim, é apresentado um fluxograma com as modificações feitas no 3D-CSD (Fig. 29). Observa-se principalmente que há a remoção da rede neural utilizada originalmente e a transformação de qualquer objeto em um grafo normalizado pela esfera, extraindo cada um de seus autovalores.

Figura 29 – Fluxograma do algoritmo 3D-CSD+.



Fonte: Elaborada pelo autor.

CAPÍTULO

5

EXPERIMENTOS E VALIDAÇÃO

Neste capítulo, são detalhados o ambiente de execução dos experimentos realizados, bem como a descrição e discussão dos resultados obtidos.

5.1 Definição do ambiente de experimentos

Durante a escrita do código, viu-se que nem todo compilador estaria em conformidade total com a revisão C++20 ou teria a implementação de algumas funcionalidades de bibliotecas do C++23 que seriam interessantes para o desenvolvimento. O único compilador em conformidade com o padrão ISO C++20, com as funcionalidades desejadas do C++23 seria o MSVC 14.34. Por isso, viu-se que o MSVC seria a melhor escolha para validação da arquitetura de software montada, obtenção dos tempos de execução e identificação dos pontos de gargalo do algoritmo.

Pelo MSVC não ter a possibilidade de compilação cruzada para outras plataformas, como é o caso do LLVM, usou-se o sistema operacional Windows kernel versão 10.0.22621.1180.

O computador utilizado nos experimentos possui as seguintes configurações:

- Processador: Intel Core i7-11800H 2.3 GHz;
- Memória RAM: 2 x HyperX 32GB SODIMM DDR4 3200Mhz FURY Impact 1,2V 2Rx8 260 pinos - KF432S20IB/32;
- Memória persistente: SSD XPG S70 Blade 1TB, M.2 NVMe, PCIe Gen4x4, Leitura: 7400MB/s e Gravação: 5500MB/s, 3D NAND - AGAMMIXS70B-1T-CS;

É importante mencionar que foram utilizadas as flags de otimização: /Oi, /Ot, /O2 e /arch:AVX512. Esta última é possível apenas para processadores que possuam instruções de vetorização de dados AVX512.

Apesar do software criado ser capaz de lidar com quaisquer tipos de pontos flutuantes,

optou-se por usar o tipo `float` para aumentar a quantidade de dados possivelmente cacheados durante a execução, melhorando o desempenho do software.

Não foi utilizado o processamento em placa gráfica dedicada em momento algum dos testes.

5.2 Experimentos

Para a validação do método apresentado nesta dissertação, preferiu-se utilizar um dataset já conhecido pela comunidade que tivesse as informações necessárias para que fosse possível usar o método: objetos dispostos em um espaço $\mathbb{R}^3$, com a descrição de sua superfície.

Vale notar que seria possível utilizar outros *datasets* apenas com vértices ou imagens de profundidade gerados por um sensor, gerando a malha da superfície por meio do ponto de perspectiva do sensor. Porém, preferiu-se não utilizar tais *datasets* pois esse passo adicional de construção da superfície não agregaria em valor na validação do método apresentado. Além disso, o fato de se adotar um *dataset* conhecido, permite a reprodutibilidade e inclusive comparações em relação aos experimentos realizados.

Com os critérios acima estabelecidos, encontrou-se o *dataset* ModelNet (WU *et al.*, 2014) [1]. Um *dataset* de objetos 3D sintéticos CAD (Computer-Aided Design), com grande variabilidade de características entre os objetos e divididos em até quarenta categorias.

Para que o método apresentado pudesse ser testado com o cenário mais variado possível, escolheu-se o conjunto ModelNet40, que integra todos os modelos possíveis, incluindo o ModelNet10, que é apenas um subconjunto da ModelNet40.

A ModelNet40 agrupa um total de 12.311 objetos pré-alinhados com altura no eixo Z, separados em 40 categorias, que são divididas em 9.843 (80%) para treinamento e 2.468 (20%) para teste. Os modelos CAD estão em formato de arquivo de objeto de tipo .off (este tipo apresenta o posicionamento de cada vértice, com a estrutura de conexão de cada face triangular do objeto). Também vale a pena mencionar a variedade na qualidade dos objetos, indo de malhas *low poly* até superfícies com alto nível de detalhe e quantidade de vértices.

5.3 Pré-processamento

Analisando o *dataset*, viu-se um problema e uma característica indesejada no *dataset*, que foi removida para aumentar a acurácia do algoritmo.

O problema eram falhas no cabeçalho, fazendo com que o objeto não fosse válido. Para alguns objetos, a frase de cabeçalho "OFF" é concatenada com as coordenadas sem uma quebra

[1] Ranking de classificação da ModelNet: https://paperswithcode.com/dataset/modelnet

de linha. Caso um outro usuário do sistema deseje reproduzir os experimentos, será necessário fazer tais ajustes nos objetos.

O segundo ponto a ser abordado são os vértices não conectados a quaisquer faces. Alguns objetos apresentam vértices soltos, que provavelmente escaparam da limpeza do *dataset*, durante a exportação do objeto CAD. Esses vértices acabam interferindo no cálculo da esfera, aumentando seu raio para além do tamanho do objeto. Para este projeto, estes vértices foram removidos, limpando os objetos.

5.4 Execução

Definidos o ambiente de execução e o *dataset*, fez-se a execução dos métodos propostos através da implementação dos algoritmos descritos no capítulo anterior, e, cujo código foi disponibilizado em um repositório público do GitHub [2]. O algoritmo não possui quaisquer dados prévios de outros treinos, ou seja, todo o conhecimento é gerado partindo-se do zero.

Para uma melhor compreensão humana, a aplicação criada gera um modelo de como ficou a abstração do grafo nas duas abordagens. Então, das abstrações, são gerados os autovalores que descrevem os objetos.

Foram utilizadas duas variações de cada tipo de esfera, com as quantidades de vértices descritas na Tabela 4.

Tabela 4 – Quantidade de vértices por esfera

Esfera	**Menor Resolução (MeR)**	**Maior Resolução (MaR)**
Deserno	44	642
HEALPix	48	588
HEALPix - equador	42	310
Icosaedro	42	642
Kogan	42	642
Rusin	46	664
Saff e Kuijlaars	42	642

Fonte: Dados da pesquisa.

As imagens de esfera apresentadas no capítulo anterior são representações das variações de maior resolução.

A coleta do tempo de execução levou apenas em consideração o tempo entre os passos descritos no capítulo anterior. Os tempos de *parsing* dos arquivos e estruturação dos objetos em memória volátil não foram considerados. É possível encontrar tais tempos na Tabela 5.

Para simplificação na terminologia apresentada nesta dissertação, será utilizado o termo **Treino** para a etapa em que há a geração dos descritores dos objetos e **Teste** para a etapa em que

[2] GitHub de Jean Amaro: https://github.com/Jean-Amaro/Xablau/

há a geração dos descritores dos objetos e busca da amostra mais similar ao objeto analisado, dentro da base de conhecimento. Todos os experimentos desta dissertação usaram a distância L_1 como função de distância. Apesar de haver a possibilidade, não foi utilizado um algoritmo de aprendizagem de máquina, mas sim uma inferência estatística para a classificação.

Tabela 5 – Tempo total em ms para cada etapa

Esfera	**MeR**		**MaR**	
	Treino	**Teste**	**Treino**	**Teste**
Deserno	69.432	13.549	2.018.273	556.150
HEALPix	55.435	7.322	1.276.675	396.378
HEALPix - equador	84.258	6.869	176.308	48.894
Icosaedro	24.886	7.078	1.933.413	548.402
Kogan	29.488	7.342	1.973.865	566.860
Rusin	23.515	7.296	2.442.666	684.765
Saff e Kuijlaars	56.034	8.825	2.066.212	582.353

Fonte: Dados da pesquisa.

As quantidades de objetos no ModelNet40 são: 9.843 para treino e 2.468 para teste. Tendo estes números, faz-se a média de tempo por objeto, sendo exibida na Tabela 6.

Tabela 6 – Tempo médio para cada objeto em ms para cada etapa.

Esfera	**MeR**		**MaR**	
	Treino	**Teste**	**Treino**	**Teste**
Deserno	7,05	5,49	205,04	225,34
HEALPix	5,63	2,97	129,70	160,61
HEALPix - equador	8,56	2,78	17,91	19,81
Icosaedro	2,53	2,87	196,43	222,21
Kogan	3,00	2,97	200,54	229,68
Rusin	2,39	2,96	248,16	277,46
Saff e Kuijlaars	5,69	3,58	209,92	235,96

Fonte: Dados da pesquisa.

Na Tabela 7, são apresentadas as taxas de acerto, para cada uma das variações de esfera.

No Apêndice B, são apresentadas as quantidades de tentativas feitas para que o algoritmo pudesse acertar de qual categoria o objeto pertenceria, para cada uma das variações de esfera.

5.5 Discussão

Analisando todas as tabelas conjuntamente, percebe-se o excelente desempenho computacional da nova abordagem, tendo uma média de até 2,78ms para classificar um objeto do *dataset*, usando uma esfera de menor resolução. Vale novamente ressaltar que toda a execução foi feita usando apenas uma CPU, sem usar um acelerador gráfico do tipo GPU.

Tabela 7 – Comparação de taxas de acerto para todas as esferas.

Esfera	MeR		MaR	
	Externa	**Completa**	**Externa**	**Completa**
Deserno	0.508509	0.469611	0.510535	0.509319
HEALPix	0.54376	0.512561	0.52188	0.499595
HEALPix - equador	**0.549433**	0.545786	0.612642	**0.614263**
Icosaedro	0.496759	0.476499	0.508104	0.506078
Kogan	0.486629	0.466775	0.516613	0.501216
Rusin	0.517018	0.493922	0.516207	0.508104
Saff e Kuijlaars	0.470827	0.448541	0.529579	0.507699

Fonte: Dados da pesquisa.

Outro ponto que chama a atenção é a semelhança a uma distribuição exponencial, que as tentativas para acertar a categoria formam. Vê-se que as tentativas corretas se agrupam muito bem no Top-3, chegando a 80% de acerto na "HEALPix - equador MaR".

Dos pontos negativos, vê-se que a taxa de acerto geral não foi boa, chegando ao máximo de 61,42%, também na "HEALPix - equador MaR". Essa taxa de acerto foi fortemente afetada por alguns objetos com certas particularidades.

Uma dessas particularidades foi a expectativa de objetos com certa simetria, seja ela bilateral ou radial. Isso faria com que a esfera circunscrita posicionasse naturalmente seu centro próximo ao ponto (ou eixo para a simetria bilateral) de simetria, porém isso não seria sempre verdade. A imagem abaixo (Fig. 30) exemplifica bem este caso.

Como é possível perceber, as folhas são distribuídas de maneira irregular, fazendo com que o centro da esfera seja deslocado, não tendo uma garantia de que aquele centro seja representativo suficientemente para outros objetos. Por exemplo: Uma outra planta pode ter caules e folhas maiores ainda, deslocando tanto o centro da esfera, que a planta usada no treino já não seria tão semelhante mais a essa nova planta, causando uma classificação errônea.

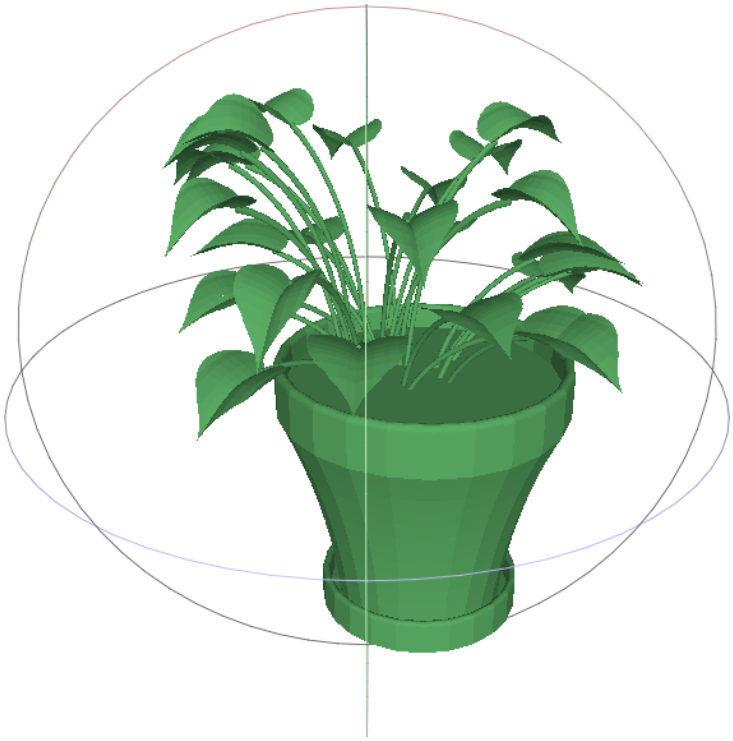
Outro ponto observado é a maior taxa de acerto na classificação, quando o objeto não contém tantos "buracos" em sua superfície. Observe a abstração externa gerada pela "HEALPix MaR", na planta ilustrada pela Fig. 31.

Vê-se claramente que diversas folhas não tiveram algum vértice respectivo na esfera, para que elas fossem representadas na geração do grafo e posteriormente no conjunto de autovalores (Fig. 32).

E semelhante ao problema anterior, mesmo que fossem representadas, não existiriam garantias de que futuras folhas tivessem uma distribuição parecida. Plantas podem possuir folhagens bem distintas umas das outras, mesmo elas sendo da mesma espécie.

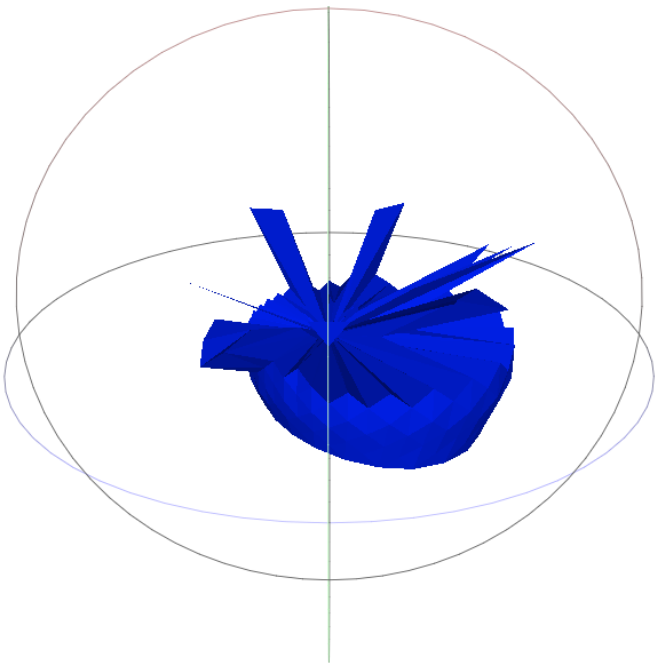
Além dos problemas citados acima, existem outros inerentes ao *dataset*. Durante testes, viu-se que nem todos os objetos possuem uma superfície de contorno ao redor de todo o objeto.

Figura 30 – flower_pot_0166 inscrita na esfera.



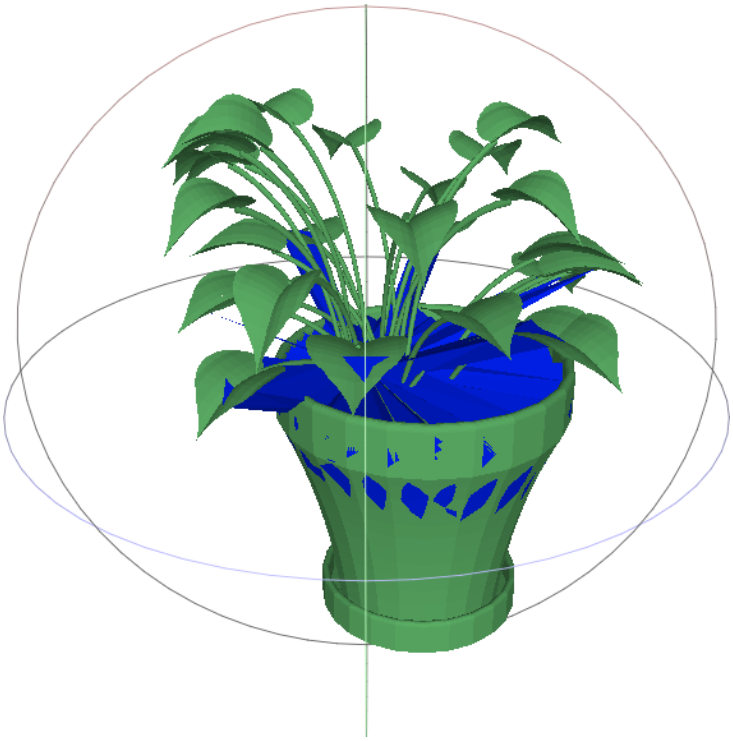
Fonte: Elaborada pelo autor.

Figura 31 – Abstração do objeto flower_pot_0166 inscrita na esfera.



Fonte: Elaborada pelo autor.

Figura 32 – Sobreposição da abstração no objeto flower_pot_0166.



Fonte: Elaborada pelo autor.

Ainda usando o mesmo objeto de cima, vê-se claramente que falta o fundo do vaso na abstração. Isso aconteceu pelo fato de não haver um fundo de vaso no objeto original, como visto na Fig. 33. Apesar de bem impactante, não foram vistos tantos objetos com este problema específico.

O segundo problema do dataset é falta da fidedignidade da escala nos objetos. A escala é uma característica muito importante para esta abordagem, pois os autovalores crescem/decrescem linearmente com o fator de escala. Por exemplo: Dois objetos idênticos, porém um com fator de escala em dez vezes, provavelmente jamais dariam correspondência (*match*) nos autovalores, pois todos os autovalores de um objeto seriam dez vezes maiores que os autovalores do outro.

Pela falta de uma padronização na escala, todos os objetos foram normalizados pelo raio da esfera, i.e., uma vez calculado o raio da esfera, aplica-se uma matriz de escala baseada neste raio, fazendo com que qualquer objeto tenha a mesma escala artificialmente.

Por fim, vê-se claramente o rápido crescimento do tempo de execução ao aumentar a quantidade de vértices na esfera, como o caso da esfera de Rusin, que aumentou o tempo necessário para validar o conjunto de testes em quase 94 vezes, porém tendo um aumento de vértices na esfera em pouco mais de 14 vezes. Isso era esperado, dado que o passo com maior Big-O é exatamente o cálculo de autovalores.

O que não se esperava é a não correlação entre quantidade de vértices de esfera, distribuição dos vértices na esfera e a taxa de acerto. Com exceção das esferas "HEALPix - equador" e "Saff e Kuijlaars", todas as esferas tiveram taxa de acerto praticamente inalteradas. E quanto à

Figura 33 – flower_pot_0166 sem a superfície de fundo.

Fonte: Elaborada pelo autor.

questão da distribuição, a esfera "HEALPix - equador" foi a que teve melhores taxas, mesmo tendo a cobertura de vértices menos uniforme entre elas.

Apesar de analisados os resultados, não foram encontrados padrões que pudessem explicar o porquê destes comportamentos, demandando futuras investigações.

5.6 Possíveis melhorias

Esta seção trata de modificações no algoritmo aqui apresentado, que não foram validadas, nem implementadas. As ideias aqui apresentadas são possíveis melhorias que o algoritmo pode ter, como a formação de um grafo por meio de outra referência geométrica, que não fosse esférica, e de uma possível análise cromática do objeto, usando os conceitos já apresentados nesta dissertação; além disso, fora do escopo de descrição e classificação de objetos, é apresentada uma possibilidade de uso como uma ferramenta de apoio na validação da qualidade do *dataset*, detectando *outliers* e dados repetidos com maior facilidade.

5.6.1 Etapa geométrica

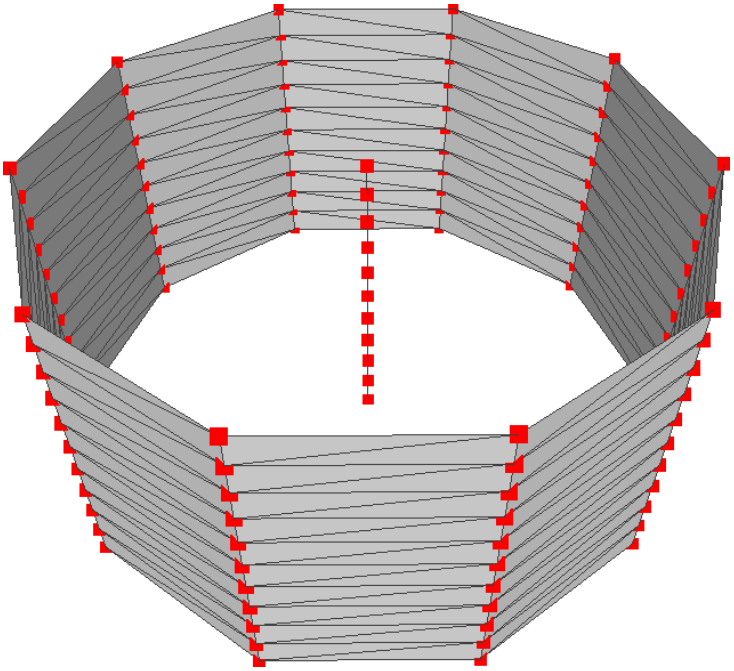
Observando a quase constância e inalteração das taxas de acerto e de como cada esfera se comportou, constata-se que o crescimento do número de vértices da esfera não necessariamente implica em uma melhor taxa de acerto.

A taxa de acerto aparenta estar muito mais conectada com a forma que os vértices são distribuídos na esfera. Isso motiva a busca de distribuições que não sejam necessariamente para emular um objeto regular, como a tentativa de apresentar uma esfera, feita neste trabalho.

E, mesmo que a distribuição fosse motivada por um objeto regular, avaliar como base outras formas geométricas, como é o caso do cilindro. Para o *dataset* ModelNet40, essa forma poderia ser aplicada sem maiores problemas, pois suas amostras estão todas com pose de altura alinhada em torno do eixo Z. Foi tendo o conhecimento prévio dessa característica do *dataset*, que motivou a criação da esfera "HEALPix - equador".

Além da distribuição dos pontos na forma geométrica, avaliar a forma de usar e/ou distribuir o vértice que servirá para ser âncora, na hora de criar um segmento e calcular a intersecção que irá para o grafo. Uma proposição simples seria o cilindro apresentado na Fig. 34, utilizando um vértice central distinto para anel de vértices do cilindro.

Figura 34 – Distribuição de vértices na forma cilíndrica, com múltiplas âncoras.



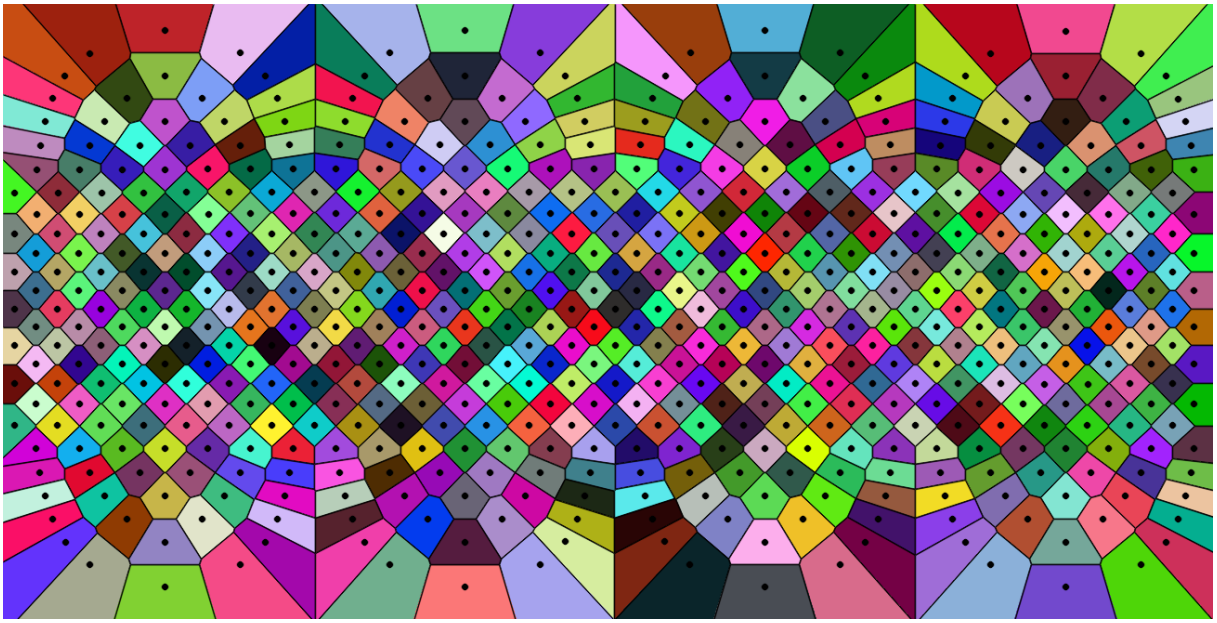
Fonte: Elaborada pelo autor.

Semelhante à esfera "HEALPix - equador", essa nova forma seria recomendada apenas para objetos com pose de altura alinhada com os vértices, pois a invariância à rotação estaria atrelada apenas a este eixo.

Outra mudança do algoritmo que trataria o problema de não ter intersecções em objetos com muitos "buracos", como o caso das folhas da planta, seria utilizar uma região de interesse para cada vértice da esfera, podendo essa região de interesse ser calculada com uma triangulação de Delaunay e diagrama de Voronoi logo após, por exemplo (Fig. 35).

Agora, o novo mapeamento seria feito em toda a região coberta por aquele vértice da

Figura 35 – HEALPix MaR com região de interesse estendida.



Fonte: Elaborada pelo autor.

esfera, i.e., se alguma caixa de contenção do triângulo projetado tiver alguma intersecção com a região do vértice da esfera, um segmento entre o centro e região deverá ser criado e um teste no plano $\mathbb{R}^3$ deverá ser feito.

Apesar de resolver um grande problema do algoritmo, a mudança acima poderá ter um forte impacto no desempenho, já que com certeza haveria no mínimo k testes de intersecção, sendo k o número de faces do objeto inscrito, entre segmento e face do objeto. Além disso, existirá a grande dificuldade de saber se a face projetada está contida em alguma região do diagrama de Voronoi, que pode ser resolvido com um teste de intersecção de polígonos, mas que tem potencial de afetar o desempenho drasticamente.

E como variação mais radical, seria possível utilizar não apenas uma forma geométrica que envelopa o objeto, mas um conjunto de projeções que gerariam múltiplos grafos (e autovalores, subsequentemente), que descreveriam o objeto. Porém, esta abordagem precisaria projetar o objeto de diversos ângulos distintos, causando um grande aumento de inserções de autovalores na base de conhecimento.

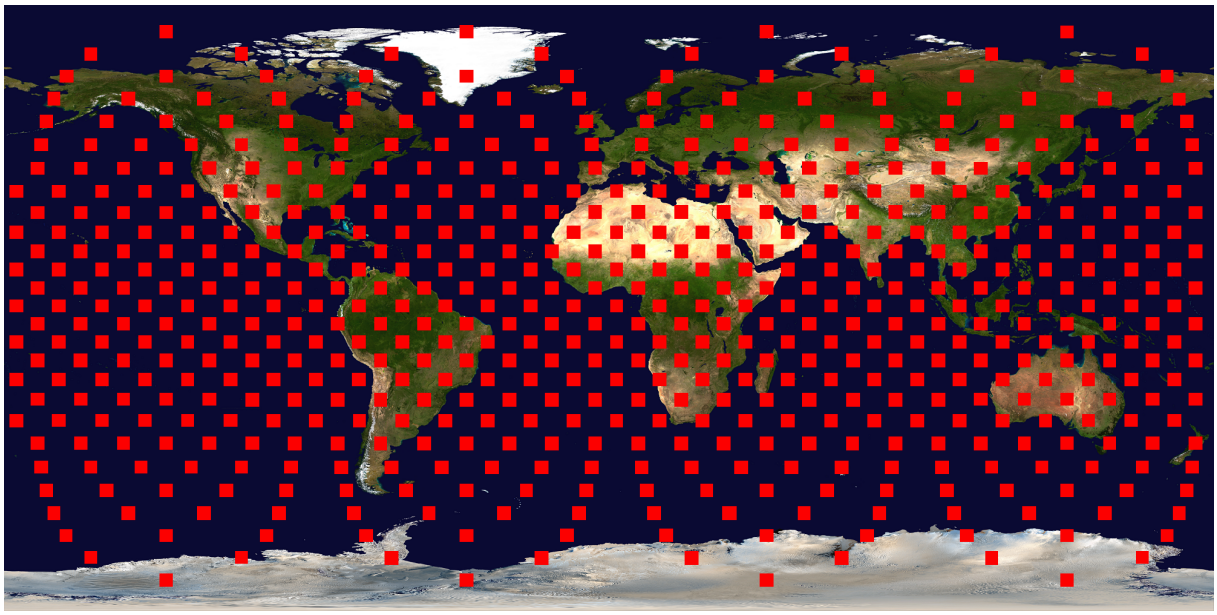
5.6.2 Etapa cromática

Além das mudanças feitas para otimizar o tratamento geométrico do objeto, é perfeitamente possível introduzir um processamento cromático ao algoritmo, usando apenas a cor mapeada na intersecção calculada durante a etapa geométrica. Este tipo de abordagem permitiria uma junção da geometria (3D) com a imagem (2D), com suas cores e texturas, ampliando assim a quantidade de informações que descrevem o objeto (geometria 3D + imagem 2D), e que vão

permitir um melhor reconhecimento deste (maior acurácia). Note que o fato de termos um processamento bastante eficiente em termos de custo computacional, vai permitir que sejam adicionadas etapas complementares de processamento em aplicações embarcadas, como o reconhecimento das características cromáticas do objeto, melhorando assim a acurácia do reconhecimento dos objetos analisados.

Um exemplo que pode ser dado seria distinguir o planeta Terra de Marte. Os dois planetas possuem formatos muito semelhantes, não considerando suas escalas, e provavelmente seriam confundidos em uma classificação estritamente geométrica. Porém, seus grafos cromáticos seriam totalmente distintos. As imagens abaixo (Fig. 36 e 37) mostram como seriam as amostragens de cor, usando a esfera "HEALPix MaR".

Figura 36 – Mapeamento UV do planeta Terra com vértices da esfera HEALPix MaR.

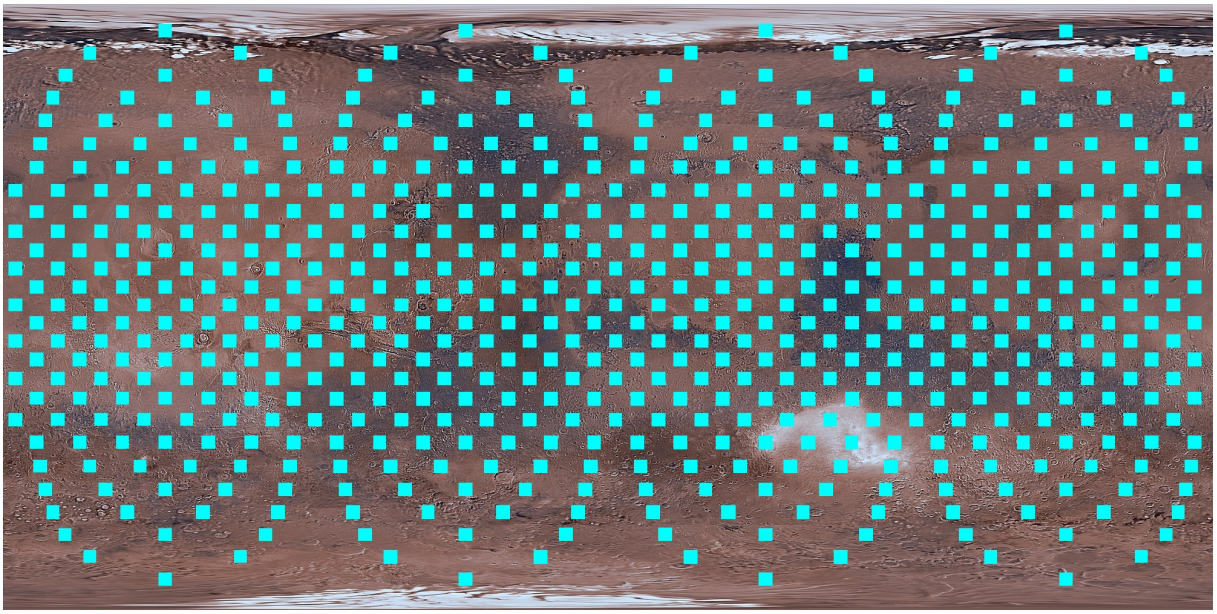


Fonte: Elaborada pelo autor.

Vê-se que as cores dos planetas representadas em cada vértice da esfera (quadrados vermelhos para Terra e cianos para Marte) gerariam grafos bem distintos. Para a etapa cromática, é possível até a escolha de qual função de distância utilizar entre as cores de cada vértice, podendo ser as comumente usadas euclideana em sRGB ou HSV, ou até mesmo distâncias mais sofisticadas, como a CIEDE2000 em CIELab.

Sobre a questão performática, apenas existiria o custo de cálculo das distâncias entre cada uma das intersecções e o cálculo deste novo grafo cromático. Isso quer dizer que, caso fosse utilizada uma distância euclideana em cima do espaço sRGB, haveria um custo de processamento adicional máximo de uma vez o tempo atual para cada esfera, pois o fluxo para etapa cromática seria idêntica à etapa geométrica. Mas lembrando que esse custo adicional pode ser reduzido, pois os cálculos de intersecções e mapeamentos podem ser reaproveitados da etapa geométrica.

Figura 37 – Mapeamento UV do planeta Marte com vértices da esfera HEALPix MaR.



Fonte: Elaborada pelo autor.

5.6.3 Integração

Tendo vista o atual funcionamento e possíveis melhorias propostas nesta seção, crê-se que o algoritmo possui capacidade suficiente para se tornar um algoritmo de apoio na classificação de objetos tridimensionais, participando da decisão com outro classificador.

Também é vista a possibilidade deste trabalho ser usado como um classificador único do pipeline de reconhecimento do objeto, dependendo de suas particularidades ou da disponibilidade de poder computacional disponível, como visto anteriormente, é possível executar este algoritmo sem o uso de uma GPU.

Além de atuar como um classificador, vislumbra-se o uso deste algoritmo como um metrificador de qualidade de um *dataset* de treino de objetos 3D. Ao fim da geração de todos os autovalores do *dataset* de treino, seria feito o seguinte passo: Executar uma classificação em cada uma das amostras de treino com o restante da base.

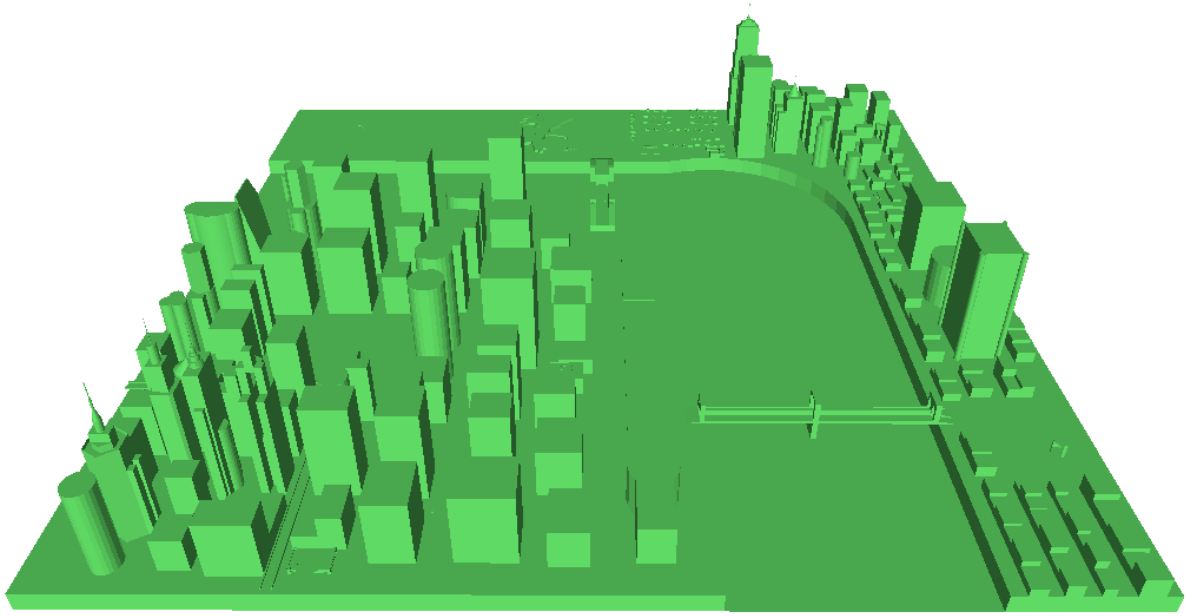
Essa nova classificação poderia auxiliar o processo de treino nos seguintes aspectos:

- Se a amostra foi classificada por várias outras amostras da mesma categoria como primeiras escolhas, pode significar que a categoria esteja bem representada.

- Se a amostra foi classificada por poucas ou nenhuma amostra da mesma categoria, pode significar que ela esteja sendo pouco representada por seus pares, podendo ser um caso espúrio da categoria e demandando mais amostras semelhantes a amostra testada; ou até mesmo significar que ela não deveria estar presente no dataset, pois sua presença não faz sentido com a categoria.

Caso houvesse uma filtragem deste tipo durante este trabalho, a amostra ilustrada pela Fig. 38 poderia ser encaixada como ponto de atenção.

Figura 38 – Representação da amostra car_0021.



Fonte: Elaborada pelo autor.

Observando o exemplo citado anteriormente, vê-se que o *dataset* ModelNet inclui a representação de algo que visualmente parece ser uma cidade dentro de uma classe que deveria representar apenas objetos semelhantes a carros. Após os testes, viu-se que essa semelhança não é apenas visual e causou a classificação incorreta de objetos de teste.

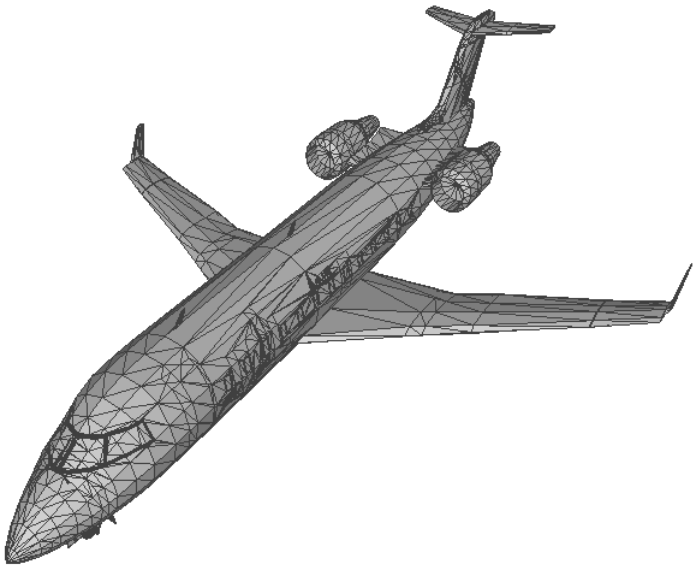
Amostras assim podem ser problemáticas para o 3D-CSD+, especialmente para usos em que é escolhida apenas a melhor amostra que obteve *match* durante a classificação, como é o caso da execução deste trabalho.

Claro, o exemplo da Fig. 38 é um caso bem incomum na ModelNet40, mas o ponto é que podem existir casos em que a diferença entre as amostras não é tão evidente como discutido aqui, podendo amostras categorizadas indevidamente entrarem na base de conhecimento do 3D-CSD+ desapercebidamente.

Por fim, além de detectar objetos com rótulos provavelmente incorretos, seria possível detectar redundâncias de objetos, fazendo com que a base de conhecimento possa ser reduzida, mas sem perder a qualidade na variabilidade de amostras conhecidas. Observe as duas imagens da categoria *airplane*, ilustradas pelas Fig. 39 e 40.

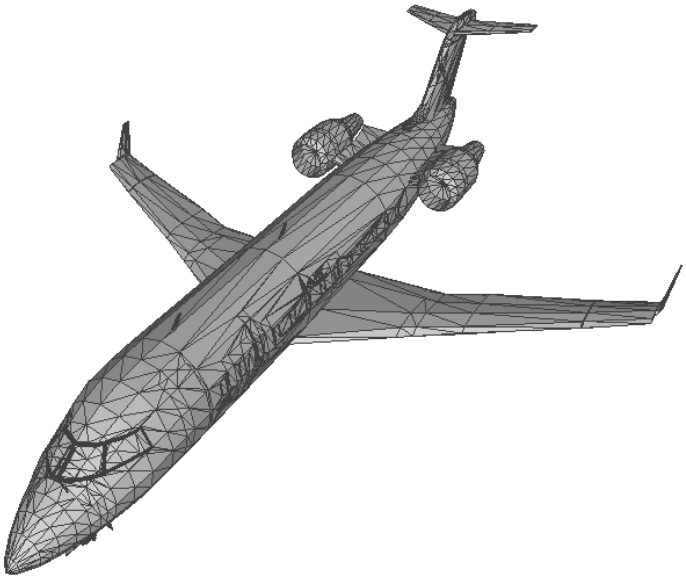
As duas amostras possuem diferenças estruturais na construção do objeto, diferenciando-se no número de faces e até mesmo no número de vértices. Porém, possuem uma alta semelhança visual entre si, causando até seus autovalores serem praticamente idênticos. A pessoa analisando

Figura 39 – Representação da amostra airplane_0324.



Fonte: Elaborada pelo autor.

Figura 40 – Representação da amostra airplane_0564.



Fonte: Elaborada pelo autor.

o *dataset* pode perceber amostras assim e removê-las do treino, aumentando o desempenho durante a etapa de treino e, em menor escala, até mesmo na etapa de teste.

5.7 Considerações finais

Após os experimentos, viu-se que a nova abordagem possui diversos pontos positivos, tendo uma melhora significativa no tempo de execução, quando comparada à abordagem original do 3D-CSD.

Porém existem pontos de melhoria e certas variações que ainda podem ser feitas no algoritmo, mitigando os problemas apresentados neste capítulo, conforme discutido nas seções anteriores.

CAPÍTULO

6

CONCLUSÃO

Sendo apresentados todos os dados de experimento, pontos positivos e negativos, vê-se que o método apresentado nesta dissertação possui potencial para ser usado em sistemas que demandam desempenho, mas que possuem limitações de hardware.

Apesar da relativa baixa taxa de acerto em Top-1 para o *dataset* ModelNet40, o algoritmo demonstrou um melhora significativa no tempo de execução e utilização de recursos, quando comparado à versão original do 3D-CSD.

Seu sucesso pode ser atribuído à habilidade de eficientemente lidar com grandes volumes de dados e classificar objetos 3D em tempo real. Isso torna seu uso viável em aplicações que requerem respostas rápidas e precisas, como em sistemas de percepção de veículos móveis autônomos, sistemas gerais de visão robótica e também podendo ser aplicado em contextos industriais.

Acredita-se que o projeto aqui apresentado serve de base para futuros projetos e pesquisa, dados os pontos de possíveis melhorias apresentados no capítulo anterior e evidenciando que, apesar da tendência predominante de redes neurais profundas, a classificação de objetos 3D pode ser feita por meio de uma abstração generalista sem construir uma aproximação ou inferência estatística tão computacionalmente custosa, como são as abordagens geralmente usadas em redes neurais profundas.

Além da questão performática, vê-se grandes qualidades na junção quase orgânica entre as abordagens geométrica e cromática deste algoritmo. Também, vale a pena ressaltar a fácil manipulação de dados que um pessoa pode fazer, podendo adicionar/remover uma amostra de categoria com apenas uma operação de inserção/remoção de seu respectivo conjunto de autovalores, dispensando a necessidade de repetir todo o processo de treino.

REFERÊNCIAS

BABAI, L. Graph isomorphism in quasipolynomial time. **CoRR**, abs/1512.03547, 2015. Disponível em: <http://arxiv.org/abs/1512.03547>. Citado na página 37.

BLENDER. **Primitives — Blender Manual**. 2019. <https://docs.blender.org/manual/en/dev/modeling/meshes/primitives.html>. Accessado em 05-03-2019. Citado na página 41.

BRONSTEIN, A. M.; BRONSTEIN, M. M.; OVSJANIKOV, M. Feature-based methods in 3d shape analysis. **3D Imaging, Analysis and Applications**, Springer, p. 185–219, 2012. Disponível em: <https://doi.org/10.1007/978-1-4471-4063-4_5>. Citado na página 26.

BUSTOS, B.; SIPIRAN, I. 3d shape matching for retrieval and recognition. In: **3D Imaging, Analysis and Applications**. Springer London, 2012. p. 265–308. Disponível em: <https://doi.org/10.1007/978-1-4471-4063-4_7>. Citado na página 23.

CARVALHO, L.; WANGENHEIM, A. von. 3d object recognition and classification: a systematic literature review. **Pattern Analysis and Applications**, Springer, v. 22, p. 1243–1292, 2019. Citado na página 26.

CIACCIA, P.; PATELLA, M.; ZEZULA, P. M-tree: An efficient access method for similarity search in metric spaces. In: **Proceedings of the 23rd International Conference on Very Large Data Bases**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997. (VLDB '97), p. 426–435. ISBN 1558604707. Citado na página 53.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Introduction to Algorithms, Third Edition**. 3rd. ed. [S.l.]: The MIT Press, 2009. ISBN 0262033844, 9780262033848. Citado na página 37.

DESERNO, M. How to generate equidistributed points on the surface of a sphere. **P.-If Polymerforshung (Ed.)**, p. 99, 2004. Citado na página 39.

FISCHER, K.; GÄRTNER, B.; KUTZ, M. Fast smallest-enclosing-ball computation in high dimensions. In: BATTISTA, G. D.; ZWICK, U. (Ed.). **Algorithms - ESA 2003**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. p. 630–641. ISBN 978-3-540-39658-1. Citado na página 43.

GEZAWA, A. S.; ZHANG, Y.; WANG, Q.; YUNQI, L. A review on deep learning approaches for 3d data representations in retrieval and classifications. **IEEE Access**, Institute of Electrical and Electronics Engineers (IEEE), v. 8, p. 57566–57593, 2020. Disponível em: <https://doi.org/10.1109/access.2020.2982196>. Citado nas páginas 24, 26 e 27.

GOLUB, G. H.; LOAN, C. F. V. **Matrix Computations (3rd Ed.)**. Baltimore, MD, USA: Johns Hopkins University Press, 1996. ISBN 0-8018-5414-8. Citado na página 53.

GUTTMAN, A. R-trees: A dynamic index structure for spatial searching. In: **Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: Association for Computing Machinery, 1984. (SIGMOD '84), p. 47–57. ISBN 0897911288. Disponível em: <https://doi.org/10.1145/602259.602266>. Citado na página 53.

HELFGOTT, H. A. **Isomorphismes de graphes en temps quasi-polynomial (d'après Babai et Luks, Weisfeiler-Leman...)**. arXiv, 2017. Disponível em: <https://arxiv.org/abs/1701.04372>. Citado na página 37.

IOANNIDOU, A.; CHATZILARI, E.; NIKOLOPOULOS, S.; KOMPATSIARIS, I. Deep learning advances in computer vision with 3d data: A survey. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 50, n. 2, apr 2017. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3042064>. Citado na página 27.

JIMéNEZ, J. J.; SEGURA, R. J.; FEITO, F. R. A robust segment/triangle intersection algorithm for interference tests. efficiency study. **Computational Geometry**, v. 43, n. 5, p. 474 – 492, 2010. ISSN 0925-7721. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0925772109001448>. Citado na página 33.

KOGAN, J. A new computationally efficient method for spacing n points on a sphere. **Rose-Hulman Undergraduate Mathematics Journal**, v. 18, n. 2, p. 5, 2017. Citado na página 40.

KOUTRA, D.; PARIKH, A. P.; RAMDAS, A.; XIANG, J. Algorithms for graph similarity and subgraph matching. In: . [S.l.: s.n.], 2011. Citado na página 38.

LOPEZ, G. L.; NEGRÓN, A. P. P.; JIMENEZ, A. D. A.; RODRIGUEZ, J. R.; PAREDES, R. I. Comparative analysis of shape descriptors for 3d objects. **Multimedia Tools and Applications**, Springer, v. 76, p. 6993–7040, 2017. Disponível em: <https://doi.org/10.1007/s10044-019-00804-4>. Citado na página 26.

NIE, W.; WANG, Y.; SONG, D.; LI, W. 3d model retrieval based on a 3d shape knowledge graph. **IEEE Access**, Institute of Electrical and Electronics Engineers (IEEE), v. 8, p. 142632–142641, 2020. Disponível em: <https://doi.org/10.1109/access.2020.3013595>. Citado nas páginas 23 e 28.

ORLOVA, S. R.; LOPATA, A. V. 3d recognition: State of the art and trends. **Automation and Remote Control**, Pleiades Publishing Ltd, v. 83, n. 4, p. 503–519, abr. 2022. Disponível em: <https://doi.org/10.1134/s0005117922040014>. Citado nas páginas 23, 24, 25 e 28.

ROUKEMA, B. F.; CALABRETTA, M. R. Mapping on the HEALPix grid. **Monthly Notices of the Royal Astronomical Society**, v. 381, n. 2, p. 865–872, 10 2007. ISSN 0035-8711. Disponível em: <https://dx.doi.org/10.1111/j.1365-2966.2007.12297.x>. Citado na página 39.

SAFF, E. B.; KUIJLAARS, A. B. J. Distributing many points on a sphere. **The Mathematical Intelligencer**, v. 19, n. 1, p. 5–11, Dec 1997. ISSN 0343-6993. Disponível em: <https://doi.org/10.1007/BF03024331>. Citado na página 39.

SALES, D. O. **Extração de features 3D para o reconhecimento de objetos em nuvem de pontos**. Tese (Doutorado) — Universidade de São Paulo, 2017. Disponível em: <https://teses.usp.br/teses/disponiveis/55/55134/tde-07022018-091205/pt-br.php>. Citado nas páginas 32, 33, 34 e 39.

SALES, D. O.; AMARO, J.; OSÓRIO, F. S. 3d shape descriptor for objects recognition. In: **2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR)**. [S.l.: s.n.], 2017. p. 1–6. Citado nas páginas 31 e 33.

SILVA, A.; FERNANDES, D.; NÉVOA, R.; MONTEIRO, J.; NOVAIS, P.; GIRãO, P.; AFONSO, T.; MELO-PINTO, P. Resource-constrained onboard inference of 3d object detection and localisation in point clouds targeting self-driving applications. **Sensors**, v. 21, n. 23, p. 7933, 2021. Citado nas páginas 19, 27 e 28.

SMITH, W. A. P. Representing, storing and visualizing 3d data. In: **3D Imaging, Analysis and Applications**. Springer London, 2012. p. 139–182. Disponível em: <https://doi.org/10.1007/978-1-4471-4063-4_4>. Citado nas páginas 24 e 25.

SOLAR, J. Ruiz-del; LONCOMILLA, P.; SOTO, N. A survey on deep learning methods for robot vision. **arXiv preprint arXiv:1803.10862**, 2018. Disponível em: <https://doi.org/10.48550/arXiv.1803.10862>. Citado nas páginas 19, 27 e 28.

TAYBI, I. O.; ALAOUI, R.; ZAKANI, F. R.; ARHID, K.; BOUKSIM, M.; GADI, T. A novel efficient 3d object retrieval method based on representative slices. In: IEEE. **2016 5th International Conference on Multimedia Computing and Systems (ICMCS)**. 2016. p. 639–644. Disponível em: <https://doi.org/10.1109/ICMCS.2016.7905617>. Citado na página 26.

WIKIPEDIA. **Left Graph**. Wikimedia Foundation. Disponível em: <https://en.wikipedia.org/wiki/File:Graph_isomorphism_a.svg>. Citado na página 38.

____. **Right Graph**. Wikimedia Foundation. Disponível em: <https://en.wikipedia.org/wiki/File:Graph_isomorphism_b.svg>. Citado na página 38.

WU, Z.; SONG, S.; KHOSLA, A.; YU, F.; ZHANG, L.; TANG, X.; XIAO, J. **3D ShapeNets: A Deep Representation for Volumetric Shapes**. arXiv, 2014. Disponível em: <https://arxiv.org/abs/1406.5670>. Citado na página 56.

APÊNDICE

A

TAXAS DE ACERTO

Neste apêndice, serão apresentadas as taxas de acerto por cada variação de esfera utilizada. Os valores destacados em negrito representam os melhores valores.

Tabela 8 – Taxas de acerto para o algoritmo de Deserno.

Categoria	MeR		MaR	
	Externa	**Completa**	**Externa**	**Completa**
airplane	0.79	0.84	0.93	0.95
bathtub	0.48	0.48	0.24	0.32
bed	0.66	0.46	0.39	0.54
bench	0.2	0.25	0.4	0.25
bookshelf	0.49	0.48	0.6	0.53
bottle	0.8	0.73	0.78	0.76
bowl	0.75	0.55	0.45	0.55
car	0.59	0.42	0.7	0.55
chair	0.44	0.47	0.63	0.61
cone	0.45	0.4	0.55	0.4
cup	0.4	0.15	0.3	0.3
curtain	0.45	0.45	0.45	0.45
desk	0.22093	0.209302	0.255814	0.22093
door	0.7	0.65	0.5	0.4
dresser	0.616279	0.523256	0.476744	0.581395
flower_pot	0.05	0.15	0.05	0.05
glass_box	0.91	0.92	0.89	0.92
guitar	0.66	0.65	0.79	0.84
keyboard	0.85	0.75	0.85	0.5
lamp	0.4	0.4	0.3	0.3
laptop	0.95	0.95	1	1
mantel	0.59	0.65	0.59	0.62
monitor	0.65	0.63	0.58	0.67
night_stand	0.244186	0.22093	0.337209	0.372093
person	0.35	0.35	0.4	0.6
piano	0.27	0.23	0.41	0.35
plant	0.21	0.11	0.21	0.25
radio	0.15	0.2	0.1	0.15
range_hood	0.36	0.33	0.34	0.3
sink	0.25	0.25	0.2	0.25
sofa	0.63	0.45	0.59	0.54
stairs	0.15	0.05	0.3	0.15
stool	0.1	0.2	0.1	0.2
table	0.42	0.54	0.45	0.45
tent	0.25	0.3	0.2	0.15
toilet	0.63	0.57	0.53	0.52
tv_stand	0.28	0.18	0.27	0.25
vase	0.55	0.5	0.49	0.45
wardrobe	0.3	0.4	0.25	0.35
xbox	0.5	0.4	0.55	0.45
Sucesso Médio Total	0.508509	0.469611	**0.510535**	0.509319

Fonte: Dados da pesquisa.

Tabela 9 – Taxas de acerto para o algoritmo HEALPix.

Categoria	MeR		MaR	
	Externa	**Completa**	**Externa**	**Completa**
airplane	0.84	0.9	0.89	0.91
bathtub	0.32	0.34	0.2	0.26
bed	0.61	0.64	0.48	0.5
bench	0.35	0.25	0.25	0.15
bookshelf	0.49	0.42	0.54	0.44
bottle	0.82	0.73	0.84	0.78
bowl	0.75	0.55	0.6	0.3
car	0.65	0.65	0.68	0.49
chair	0.55	0.54	0.66	0.61
cone	0.65	0.6	0.6	0.6
cup	0.45	0.3	0.4	0.3
curtain	0.5	0.55	0.5	0.6
desk	0.267442	0.255814	0.232558	0.244186
door	0.75	0.75	0.55	0.55
dresser	0.604651	0.453488	0.511628	0.569767
flower_pot	0.05	0.15	0.15	0.1
glass_box	0.9	0.89	0.91	0.9
guitar	0.79	0.88	0.84	0.83
keyboard	0.8	0.65	0.8	0.6
lamp	0.5	0.5	0.35	0.4
laptop	0.95	1	1	0.95
mantel	0.65	0.6	0.56	0.58
monitor	0.67	0.58	0.54	0.59
night_stand	0.337209	0.395349	0.302326	0.395349
person	0.4	0.4	0.5	0.55
piano	0.38	0.36	0.42	0.36
plant	0.18	0.13	0.28	0.25
radio	0.15	0.1	0.2	0.1
range_hood	0.44	0.42	0.3	0.29
sink	0.35	0.45	0.15	0.3
sofa	0.54	0.43	0.61	0.52
stairs	0.15	0.25	0.1	0.1
stool	0.2	0.15	0.25	0.25
table	0.5	0.47	0.55	0.49
tent	0.15	0.2	0.2	0.3
toilet	0.63	0.58	0.55	0.5
tv_stand	0.34	0.22	0.28	0.23
vase	0.6	0.57	0.5	0.49
wardrobe	0.5	0.4	0.3	0.3
xbox	0.55	0.35	0.35	0.55
Sucesso Médio Total	**0.54376**	0.512561	0.52188	0.499595

Fonte: Dados da pesquisa.

Tabela 10 – Taxas de acerto para o algoritmo HEALPix - equador.

Categoria	MeR		MaR	
	Externa	Completa	Externa	Completa
airplane	0.88	0.86	0.93	0.93
bathtub	0.34	0.46	0.44	0.48
bed	0.5	0.66	0.59	0.71
bench	0.35	0.25	0.3	0.3
bookshelf	0.56	0.54	0.69	0.63
bottle	0.77	0.73	0.77	0.68
bowl	0.85	0.8	0.7	0.5
car	0.66	0.6	0.76	0.74
chair	0.56	0.57	0.76	0.76
cone	0.5	0.5	0.65	0.5
cup	0.35	0.35	0.4	0.3
curtain	0.4	0.5	0.55	0.6
desk	0.337209	0.372093	0.325581	0.348837
door	0.8	0.75	0.8	0.75
dresser	0.523256	0.476744	0.523256	0.523256
flower_pot	0.05	0	0.1	0.05
glass_box	0.87	0.87	0.88	0.91
guitar	0.8	0.84	0.82	0.83
keyboard	0.8	0.7	0.95	0.75
lamp	0.4	0.35	0.5	0.5
laptop	0.95	0.95	0.9	0.95
mantel	0.62	0.65	0.75	0.84
monitor	0.66	0.7	0.8	0.79
night_stand	0.406977	0.395349	0.418605	0.476744
person	0.55	0.4	0.75	0.8
piano	0.35	0.4	0.45	0.49
plant	0.28	0.22	0.23	0.29
radio	0.2	0.2	0.3	0.3
range_hood	0.44	0.45	0.48	0.47
sink	0.45	0.3	0.55	0.25
sofa	0.62	0.54	0.73	0.79
stairs	0.05	0.25	0.15	0.15
stool	0.35	0.25	0.3	0.25
table	0.5	0.51	0.54	0.6
tent	0.25	0.4	0.2	0.4
toilet	0.64	0.63	0.73	0.76
tv_stand	0.33	0.26	0.42	0.33
vase	0.59	0.59	0.59	0.55
wardrobe	0.45	0.3	0.6	0.4
xbox	0.6	0.5	0.75	0.55
Sucesso Médio Total	0.549433	0.545786	0.612642	**0.614263**

Fonte: Dados da pesquisa.

Tabela 11 – Taxas de acerto para o algoritmo Icosaedro.

Categoria	**MeR**		**MaR**	
	Externa	**Completa**	**Externa**	**Completa**
airplane	0.84	0.81	0.88	0.94
bathtub	0.38	0.34	0.26	0.2
bed	0.55	0.47	0.42	0.47
bench	0.35	0.25	0.25	0.2
bookshelf	0.46	0.5	0.61	0.46
bottle	0.75	0.7	0.8	0.78
bowl	0.7	0.65	0.65	0.6
car	0.61	0.44	0.63	0.54
chair	0.48	0.57	0.57	0.6
cone	0.4	0.45	0.6	0.45
cup	0.4	0.2	0.4	0.3
curtain	0.35	0.55	0.45	0.55
desk	0.232558	0.313953	0.151163	0.267442
door	0.75	0.5	0.6	0.6
dresser	0.476744	0.5	0.5	0.476744
flower_pot	0.05	0	0	0.05
glass_box	0.83	0.86	0.87	0.91
guitar	0.74	0.75	0.83	0.88
keyboard	0.75	0.65	0.85	0.6
lamp	0.3	0.35	0.3	0.45
laptop	0.85	0.85	0.95	1
mantel	0.58	0.64	0.54	0.65
monitor	0.6	0.6	0.53	0.6
night_stand	0.27907	0.209302	0.232558	0.290698
person	0.4	0.55	0.5	0.45
piano	0.33	0.28	0.36	0.34
plant	0.22	0.15	0.3	0.24
radio	0.15	0.25	0.15	0.15
range_hood	0.33	0.34	0.37	0.38
sink	0.2	0.15	0.1	0.2
sofa	0.69	0.58	0.55	0.56
stairs	0.25	0.3	0.1	0.2
stool	0.15	0.1	0.2	0.2
table	0.4	0.51	0.55	0.56
tent	0.15	0.15	0.25	0.25
toilet	0.68	0.57	0.6	0.54
tv_stand	0.23	0.17	0.32	0.27
vase	0.51	0.49	0.5	0.41
wardrobe	0.3	0.2	0.2	0.25
xbox	0.45	0.25	0.55	0.35
Sucesso Médio Total	0.496759	0.476499	**0.508104**	0.506078

Fonte: Dados da pesquisa.

Tabela 12 – Taxas de acerto para o algoritmo de Kogan.

Categoria	MeR		MaR	
	Externa	**Completa**	**Externa**	**Completa**
airplane	0.72	0.85	0.9	0.96
bathtub	0.26	0.38	0.14	0.28
bed	0.53	0.52	0.42	0.55
bench	0.1	0.15	0.35	0.2
bookshelf	0.42	0.53	0.54	0.48
bottle	0.86	0.78	0.84	0.75
bowl	0.75	0.7	0.7	0.5
car	0.59	0.44	0.59	0.43
chair	0.44	0.5	0.61	0.59
cone	0.65	0.55	0.65	0.5
cup	0.15	0.15	0.35	0.35
curtain	0.45	0.45	0.5	0.6
desk	0.197674	0.22093	0.232558	0.232558
door	0.55	0.65	0.6	0.4
dresser	0.674419	0.511628	0.534884	0.44186
flower_pot	0.1	0.05	0.2	0.1
glass_box	0.9	0.9	0.91	0.89
guitar	0.61	0.64	0.77	0.88
keyboard	0.45	0.3	0.75	0.65
lamp	0.45	0.45	0.35	0.5
laptop	0.9	0.95	1	1
mantel	0.68	0.65	0.59	0.63
monitor	0.58	0.54	0.53	0.64
night_stand	0.209302	0.244186	0.255814	0.337209
person	0.25	0.2	0.4	0.5
piano	0.3	0.31	0.38	0.37
plant	0.15	0.15	0.18	0.26
radio	0.15	0.3	0.2	0.25
range_hood	0.33	0.35	0.41	0.28
sink	0.35	0.2	0.15	0.35
sofa	0.62	0.52	0.55	0.51
stairs	0.05	0.05	0.25	0.2
stool	0.15	0.15	0.3	0.2
table	0.36	0.46	0.61	0.56
tent	0.3	0.3	0.2	0.3
toilet	0.59	0.43	0.59	0.48
tv_stand	0.3	0.2	0.3	0.24
vase	0.58	0.44	0.53	0.39
wardrobe	0.45	0.4	0.25	0.3
xbox	0.7	0.4	0.55	0.45
Sucesso Médio Total	0.486629	0.466775	**0.516613**	0.501216

Fonte: Dados da pesquisa.

Tabela 13 – Taxas de acerto para o algoritmo de Rusin.

Categoria	MeR		MaR	
	Externa	**Completa**	**Externa**	**Completa**
airplane	0.86	0.85	0.86	0.92
bathtub	0.36	0.4	0.32	0.3
bed	0.58	0.54	0.41	0.44
bench	0.15	0.2	0.3	0.25
bookshelf	0.48	0.49	0.59	0.6
bottle	0.77	0.71	0.81	0.8
bowl	0.75	0.6	0.35	0.55
car	0.62	0.47	0.7	0.53
chair	0.45	0.53	0.61	0.61
cone	0.65	0.7	0.65	0.4
cup	0.4	0.3	0.4	0.25
curtain	0.4	0.45	0.5	0.5
desk	0.27907	0.325581	0.244186	0.290698
door	0.7	0.5	0.65	0.55
dresser	0.662791	0.569767	0.453488	0.523256
flower_pot	0.05	0.1	0.15	0.05
glass_box	0.9	0.9	0.88	0.91
guitar	0.74	0.73	0.81	0.83
keyboard	0.7	0.55	0.9	0.7
lamp	0.4	0.45	0.4	0.35
laptop	0.9	0.75	0.9	1
mantel	0.6	0.64	0.64	0.59
monitor	0.62	0.59	0.54	0.55
night_stand	0.325581	0.27907	0.232558	0.337209
person	0.35	0.55	0.45	0.5
piano	0.31	0.32	0.36	0.35
plant	0.21	0.17	0.19	0.31
radio	0.2	0.2	0.15	0.25
range_hood	0.29	0.32	0.36	0.33
sink	0.25	0.25	0.3	0.3
sofa	0.63	0.53	0.62	0.49
stairs	0.2	0.1	0.25	0.1
stool	0.15	0.15	0.25	0.25
table	0.48	0.57	0.56	0.54
tent	0.15	0.25	0.15	0.15
toilet	0.72	0.56	0.57	0.58
tv_stand	0.25	0.21	0.26	0.25
vase	0.54	0.49	0.5	0.44
wardrobe	0.4	0.35	0.2	0.15
xbox	0.4	0.35	0.6	0.35
Sucesso Médio Total	**0.517018**	0.493922	0.516207	0.508104

Fonte: Dados da pesquisa.

Tabela 14 – Taxas de acerto para o algoritmo de Saff e Kuijlaars.

Categoria	**MeR**		**MaR**	
	Externa	**Completa**	**Externa**	**Completa**
airplane	0.87	0.87	0.9	0.94
bathtub	0.36	0.42	0.2	0.24
bed	0.46	0.5	0.43	0.52
bench	0.2	0.05	0.25	0.15
bookshelf	0.37	0.37	0.64	0.54
bottle	0.78	0.67	0.84	0.75
bowl	0.65	0.55	0.65	0.55
car	0.56	0.44	0.68	0.47
chair	0.46	0.48	0.65	0.61
cone	0.55	0.55	0.7	0.45
cup	0.25	0.2	0.35	0.25
curtain	0.35	0.45	0.4	0.4
desk	0.197674	0.127907	0.209302	0.197674
door	0.75	0.8	0.7	0.55
dresser	0.662791	0.616279	0.360465	0.511628
flower_pot	0.05	0.05	0.1	0.1
glass_box	0.87	0.86	0.87	0.93
guitar	0.61	0.69	0.79	0.85
keyboard	0.5	0.5	0.9	0.75
lamp	0.3	0.45	0.4	0.45
laptop	0.65	0.75	0.95	0.95
mantel	0.62	0.66	0.65	0.59
monitor	0.44	0.36	0.52	0.61
night_stand	0.348837	0.27907	0.255814	0.337209
person	0.65	0.35	0.55	0.5
piano	0.28	0.29	0.4	0.35
plant	0.11	0.16	0.21	0.23
radio	0.25	0.2	0.2	0.25
range_hood	0.33	0.31	0.44	0.39
sink	0.25	0.2	0.3	0.2
sofa	0.54	0.41	0.6	0.5
stairs	0.1	0.15	0.25	0.15
stool	0.15	0.2	0.2	0.2
table	0.45	0.43	0.6	0.59
tent	0.1	0.05	0.2	0.2
toilet	0.55	0.52	0.61	0.59
tv_stand	0.26	0.19	0.36	0.29
vase	0.52	0.54	0.5	0.43
wardrobe	0.4	0.35	0.3	0.15
xbox	0.45	0.3	0.45	0.4
Sucesso Médio Total	0.470827	0.448541	**0.529579**	0.507699

Fonte: Dados da pesquisa.

APÊNDICE

B

TENTATIVAS PARA ACERTAR

Neste apêndice, serão apresentadas as quantidades de tentativa para acertar a categoria, por cada variação de esfera utilizada.

Tabela 15 – Tentativas para acertar a categoria, para o algoritmo de Deserno.

Tentativas para acertar a categoria	MeR		MaR	
	Externa	Completa	Externa	Completa
1	1255	1159	1260	1257
2	298	274	328	296
3	157	164	179	179
4	107	130	140	138
5	74	88	87	91
6	69	88	84	73
7	52	68	40	62
8	50	64	49	47
9	50	44	41	50
10	45	41	45	36
11	39	37	25	31
12	28	37	20	25
13	40	22	18	17
14	18	34	17	25
15	25	30	15	20
16	16	21	12	14
17	26	20	12	12
18	11	20	11	8
19	15	10	11	12
20	7	19	6	10
21	10	15	12	7
22	9	12	10	7
23	10	9	6	7
24	8	7	4	5
25	7	8	5	5
26	7	4	1	6
27	10	6	4	3
28	4	7	6	4
29	5	5	3	4
30	1	2	5	4
31	2	6	4	2
32	4	2	3	5
33	3	5	1	1
34	3	3	2	1
35	0	1	0	2
36	2	4	0	0
37	1	1	1	1
38	0	0	1	0
39	0	0	0	0
40	0	1	0	1

Fonte: Dados da pesquisa.

Tabela 16 – Tentativas para acertar a categoria, para o algoritmo HEALPix.

Tentativas para acertar a categoria	MeR		MaR	
	Externa	Completa	Externa	Completa
1	1342	1265	1288	1233
2	255	260	285	326
3	173	158	182	186
4	117	118	125	107
5	86	108	108	87
6	71	68	87	84
7	37	81	43	59
8	47	61	53	58
9	39	37	30	33
10	35	34	33	39
11	42	26	37	30
12	27	35	24	28
13	21	28	18	25
14	15	15	13	25
15	24	19	16	19
16	15	19	16	9
17	14	16	13	13
18	15	14	14	11
19	6	12	7	10
20	12	11	12	13
21	13	11	10	7
22	9	9	10	6
23	5	8	8	9
24	9	7	9	8
25	4	7	4	11
26	7	4	4	3
27	5	9	1	4
28	4	5	3	7
29	5	9	2	3
30	5	3	4	3
31	3	1	2	6
32	1	4	1	1
33	0	3	1	2
34	1	0	2	1
35	2	2	2	1
36	1	0	0	0
37	1	1	0	0
38	0	0	0	0
39	0	0	1	0
40	0	0	0	1

Fonte: Dados da pesquisa.

Tabela 17 – Tentativas para acertar a categoria, para o algoritmo HEALPix - equador.

Tentativas para acertar a categoria	MeR		MaR	
	Externa	Completa	Externa	Completa
1	1356	1347	1512	1516
2	281	287	296	301
3	157	181	151	158
4	119	124	114	106
5	68	95	59	73
6	72	52	53	43
7	55	47	46	41
8	40	29	32	23
9	45	40	26	17
10	34	32	21	26
11	31	27	20	21
12	24	16	15	21
13	20	19	9	18
14	20	26	18	20
15	19	28	16	15
16	13	16	11	12
17	14	12	16	10
18	13	16	8	10
19	11	8	6	6
20	9	13	7	3
21	7	4	4	2
22	12	7	7	6
23	5	7	5	4
24	10	3	1	5
25	5	5	3	3
26	6	3	2	1
27	2	4	4	3
28	4	1	1	0
29	3	5	3	0
30	3	3	1	2
31	2	0	0	0
32	3	4	0	0
33	1	0	0	0
34	2	3	0	0
35	0	2	0	0
36	0	0	0	1
37	0	0	0	0
38	0	0	0	0
39	1	0	0	1
40	1	2	1	0

Fonte: Dados da pesquisa.

Tabela 18 – Tentativas para acertar a categoria, para o algoritmo Icosaedro.

Tentativas para acertar a categoria	MeR		MaR	
	Externa	Completa	Externa	Completa
1	1226	1176	1254	1249
2	280	296	343	311
3	172	158	177	187
4	100	97	117	129
5	93	81	103	103
6	71	81	87	75
7	70	89	53	53
8	36	57	40	46
9	38	51	33	35
10	48	50	30	44
11	30	40	30	29
12	30	26	17	21
13	18	37	26	23
14	38	33	14	20
15	18	22	13	13
16	21	16	22	15
17	26	24	17	19
18	23	19	14	11
19	16	15	7	8
20	15	10	7	6
21	11	15	5	14
22	17	16	8	9
23	9	12	5	4
24	11	5	7	4
25	7	6	6	6
26	10	4	5	3
27	6	3	3	7
28	3	6	6	3
29	8	3	8	4
30	3	4	4	7
31	3	1	1	3
32	3	3	1	1
33	1	2	2	2
34	1	1	0	0
35	1	2	1	1
36	3	4	0	2
37	1	2	0	0
38	0	0	1	0
39	0	1	1	0
40	1	0	0	1

Fonte: Dados da pesquisa.

Tabela 19 – Tentativas para acertar a categoria, para o algoritmo de Kogan.

Tentativas para acertar a categoria	MeR		MaR	
	Externa	Completa	Externa	Completa
1	1201	1152	1275	1237
2	290	320	324	312
3	185	172	171	191
4	117	127	128	135
5	105	90	96	101
6	80	65	96	80
7	68	65	57	69
8	45	68	46	39
9	39	48	38	35
10	41	49	37	34
11	38	32	27	16
12	33	30	22	27
13	26	40	12	27
14	28	35	20	27
15	24	19	10	16
16	14	25	9	11
17	23	18	11	13
18	12	15	7	8
19	12	7	9	11
20	13	9	8	7
21	6	8	10	11
22	9	7	9	13
23	8	10	7	4
24	9	6	5	7
25	3	6	5	4
26	7	5	4	9
27	6	4	9	4
28	8	5	3	2
29	4	12	2	3
30	1	3	3	6
31	4	4	0	1
32	2	4	3	2
33	5	2	1	0
34	0	1	1	0
35	0	2	2	3
36	0	1	0	2
37	2	1	0	0
38	0	1	0	0
39	0	0	1	0
40	0	0	0	1

Fonte: Dados da pesquisa.

Tabela 20 – Tentativas para acertar a categoria, para o algoritmo de Rusin.

Tentativas para acertar a categoria	MeR		MaR	
	Externa	Completa	Externa	Completa
1	1276	1219	1274	1254
2	302	300	309	318
3	140	138	162	154
4	97	125	152	120
5	109	84	101	103
6	63	74	64	84
7	51	70	71	65
8	44	58	50	56
9	55	39	34	51
10	40	48	30	40
11	25	42	25	20
12	25	39	28	18
13	26	20	21	34
14	29	26	12	13
15	21	22	11	14
16	23	21	15	12
17	15	21	11	13
18	13	17	12	6
19	9	9	17	12
20	9	16	8	14
21	13	11	5	12
22	11	10	8	6
23	11	8	7	9
24	5	7	7	4
25	14	5	3	3
26	6	10	6	5
27	6	5	3	4
28	6	7	1	3
29	3	2	8	4
30	5	1	6	6
31	3	5	0	2
32	1	0	2	2
33	5	1	0	1
34	1	2	3	2
35	0	1	0	1
36	3	2	0	0
37	2	1	0	2
38	0	1	1	0
39	0	1	1	0
40	1	0	0	1

Fonte: Dados da pesquisa.

Tabela 21 – Tentativas para acertar a categoria, para o algoritmo de Saff e Kuijlaars.

Tentativas para acertar a categoria	MeR		MaR	
	Externa	Completa	Externa	Completa
1	1162	1107	1307	1253
2	311	290	301	304
3	174	192	175	182
4	125	123	123	137
5	93	93	93	102
6	76	93	71	58
7	55	66	51	61
8	69	63	54	48
9	51	47	36	43
10	37	58	29	44
11	35	50	33	28
12	23	27	30	32
13	34	38	20	15
14	33	29	14	17
15	28	25	13	19
16	16	21	8	8
17	20	25	17	12
18	13	22	13	15
19	20	15	12	11
20	19	15	9	12
21	7	11	8	10
22	12	10	6	8
23	10	8	5	6
24	8	6	7	9
25	5	3	5	3
26	3	3	4	2
27	4	5	6	5
28	6	3	1	2
29	1	6	5	5
30	4	2	3	8
31	5	1	2	1
32	3	5	2	3
33	1	1	1	2
34	3	1	2	1
35	1	2	0	1
36	1	0	0	0
37	0	1	0	0
38	0	0	1	0
39	0	1	1	0
40	0	0	0	1

Fonte: Dados da pesquisa.

APÊNDICE

C

REPOSITÓRIO DO GITHUB

Todos os experimentos feitos na validação deste projeto pode ser encontrado no repositório do autor[1]. O nome do repositório não é um acrônimo e foi escolhido apenas por ter uma sonoridade engraçada.

No momento de escrita desta dissertação, podem ser encontrados os seguintes módulos no repositório citado acima:

- **Algebra:** Abstração de tensores e números complexos, além de categorização de tipo de dados usados em diversos conjuntos, reforçando o *type-safety* a este módulo.
- **BoundaryDescriptorClassifierApplication:** Aplicação utilizada nesta dissertação, linkando todos os módulos necessários.
- **Color:** Base para manipulação de espaços de cores. Até o momento, foram implementadas apenas funções de conversão e distância. Contudo deseja-se que este módulo seja a base da integração cromática deste projeto, porém sendo genérico o suficiente para ser utilizado em outros contextos.
- **ComputerGraphics:** Manipulador de objetos tridimensionais, guardando a estrutura entre vértices e faces, permitindo a leitura e escrita dos formatos *.off* e *.obj*.
- **Geometry:** Base para todos os elementos e algoritmos de Geometria deste projeto, porém estendendo-se para muito além de seus requisitos.
- **Graph:** Abstrações de grafos, dígrafos e árvores. Além disso, foram implementados algoritmos não usados nesta dissertação, mas bem conhecidos em Teoria de Grafos, como os de intermediação (*betweenness*) entre nós e arestas.
- **IO:** Extensão da classe de entrada e saída de arquivos `<fstream>` do C++.

[1] Xablau: https://github.com/Jean-Amaro/Xablau/

- **MachineLearning:** Implementação do 3D-CSD+ e *wrapper* para a biblioteca Eigen.
- **Testing:** Módulo com classe auxiliar para expandir testes unitários, usando apenas rótulos de tipos.
- **UnitTesting:** Testes unitários dos módulos acima.

Vale mencionar que todo o código confeccionado visou respeitar as regras de segurança de tipagem, sempre que possível restringindo tipos aceitos por meio de `concepts`[2], garantindo que a aplicação comportamento definido já durante a compilação.

Reforçando o contexto segurança no ambiente de robôs móveis autônomos, foi aplicado o software de análise estática PVS-Studio [3]. Este analisador é capaz de verificar códigos, de acordo com as regras das *guidelines* MISRA C++14 e AUTOSAR, e também pontos vulneráveis que podem ser brechas para um atacante invadir uma aplicação.

Claro, nem todas as regras das *guidelines* acima foram seguidas, pois algumas conflitam com as boas práticas do C++ moderno ou não faziam sentido com o contexto do fluxo de execução, como a lista não exaustiva abaixo:

- Ter apenas um ponto de saída da função;
- Sempre utilizar um `else` ao fim de uma cadeia de condicionais `if ... else if`.
- O `default` de um `switch` sempre deve ter um comentário. Para esses casos, o comentário é inserido nas mensagens das exceções, não demandando um comentário no código.

Contudo, apontamentos feitos em acessos de memória, quando verificados que não falsos positivos, foram corrigidos; garantindo a maior conformidade possível com as *guidelines* MISRA C++14 e AUTOSAR, reforçando seu uso seguro no âmbito de robôs móveis autônomos.

Por fim, vale mencionar que houve a tentativa de escrita de um código que pudesse seguir todos os padrões acima, mas necessitou-se integrar a Eigen[4] ao projeto, por motivos de performance, para que fosse feito o cálculo de autovalores.

Vale mencionar que a dependência citada acima não possui esse rigor de se adequar aos padrões de escrita de código para robôs e foram identificadas algumas manipulações de memória inseguras em seu código. Porém, enxerga-se que essas manipulações foram feitas visando melhor performance, sendo garantidas suas execuções por algum fluxo prévio ao ponto que gerou o *warning*.

[2] Referência da biblioteca `concepts`: https://en.cppreference.com/w/cpp/concepts
[3] PVS-Studio: https://pvs-studio.com/
[4] Biblioteca de Álgebra Linear Eigen: https://eigen.tuxfamily.org/

APÊNDICE

D

RECONSTRUÇÃO DE MALHA

Neste apêndice, será tratada uma técnica para a reconstrução de objetos tridimensionais com malha, a partir de sensores reais. Os dois tipos de sensores abordados são os baseados na visão em *frustum* (como o Kinect) e em campo de visão 360º (como o Velodyne).

Qualquer um dos dois tipos citados acima pode ter a malha reconstruída, por meio do mapeamento matricial dos pontos em cada um dos feixes do sensor.

O passo inicial do algoritmo é a geração de uma matriz (seja de profundidade ou de pontos em $\mathbb{R}^3$), com altura m e largura n. A partir desta matriz, é possível construir triângulos da malha, por meio de células vizinhas da matriz.

Para melhor visualização, observe a Fig. 41. Para cada pixel da imagem, são feitos dois testes de distância de vizinhança com os pixels abaixo e ao lado direito, conforme descrito no Algoritmo 2.

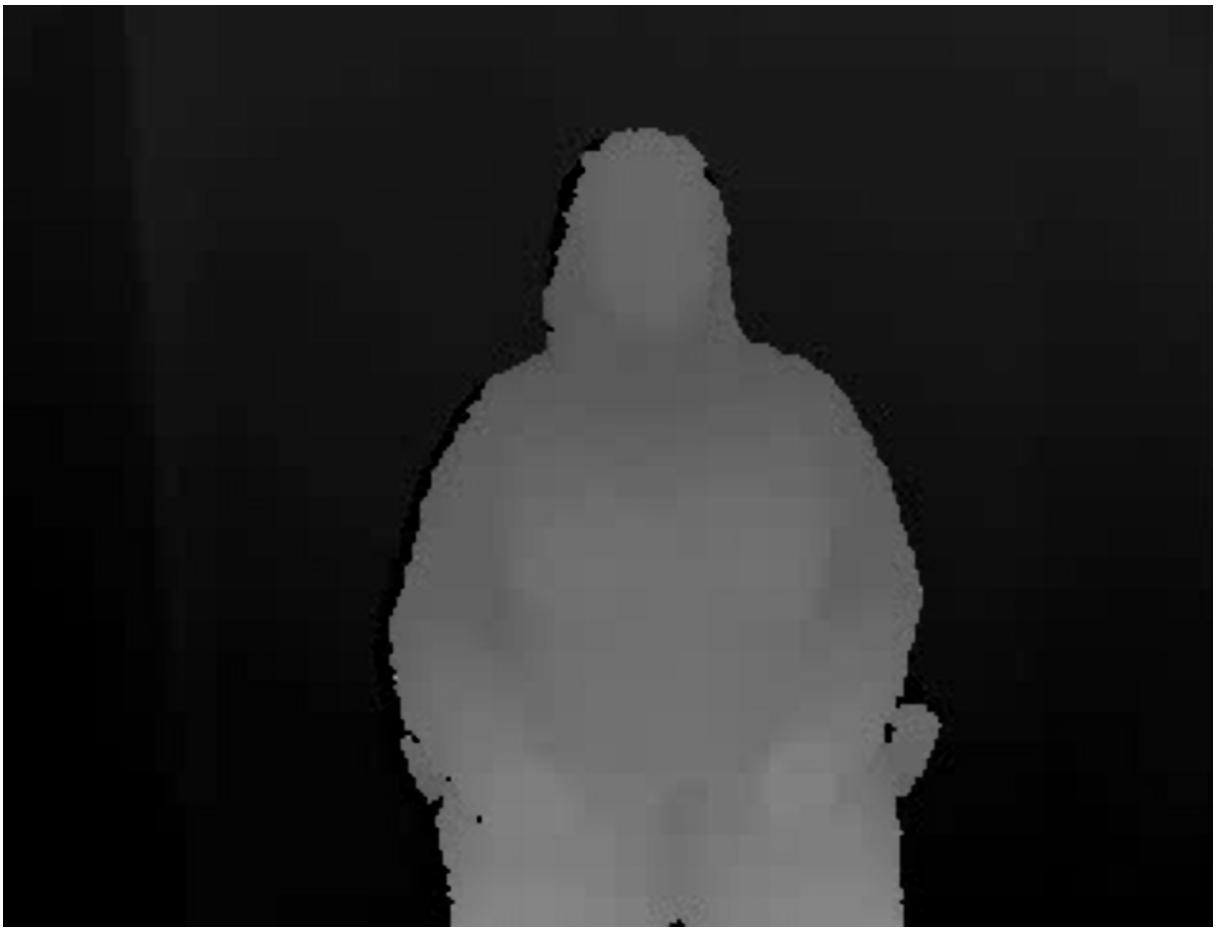
Por mera simplificação, a distância utilizada no algoritmo anterior leva em consideração apenas a distância em profundidade dos dois pixels, mas também é possível a distância euclidiana em $\mathbb{R}^3$, mapeando cada pixel com seu respectivo vértice no plano tridimensional.

Após a construção de todos os triângulos, é possível ter uma malha semelhante à Fig. 42.

Apesar de usar uma leitura do Kinect como exemplo, sensores como o Velodyne também podem organizar seus dados em formato matricial. Para cada leitura do sensor, tem-se sempre a mesma quantidade m de pontos lidos na altura. E para cada revolução do sensor, faz-se uma quantidade máxima n de leituras do sensor, gerando uma matriz m por n. Daqui, pode-se reaproveitar todo o processo de geração de malha apresentado anteriormente.

Sobre os eventuais b̈uracosq̈ue podem aparecer, é possível agrupar os pixels (ou vértices) inválidos por meio da aplicação do algoritmo *flood fill* na matriz. Para cada grupo, verificam-se os valores dos pixels (ou vértices) vizinhos válidos, atribuindo valores válidos semelhantes aos pixels (ou vértices) inválidos.

Figura 41 – Imagem de profundidade.

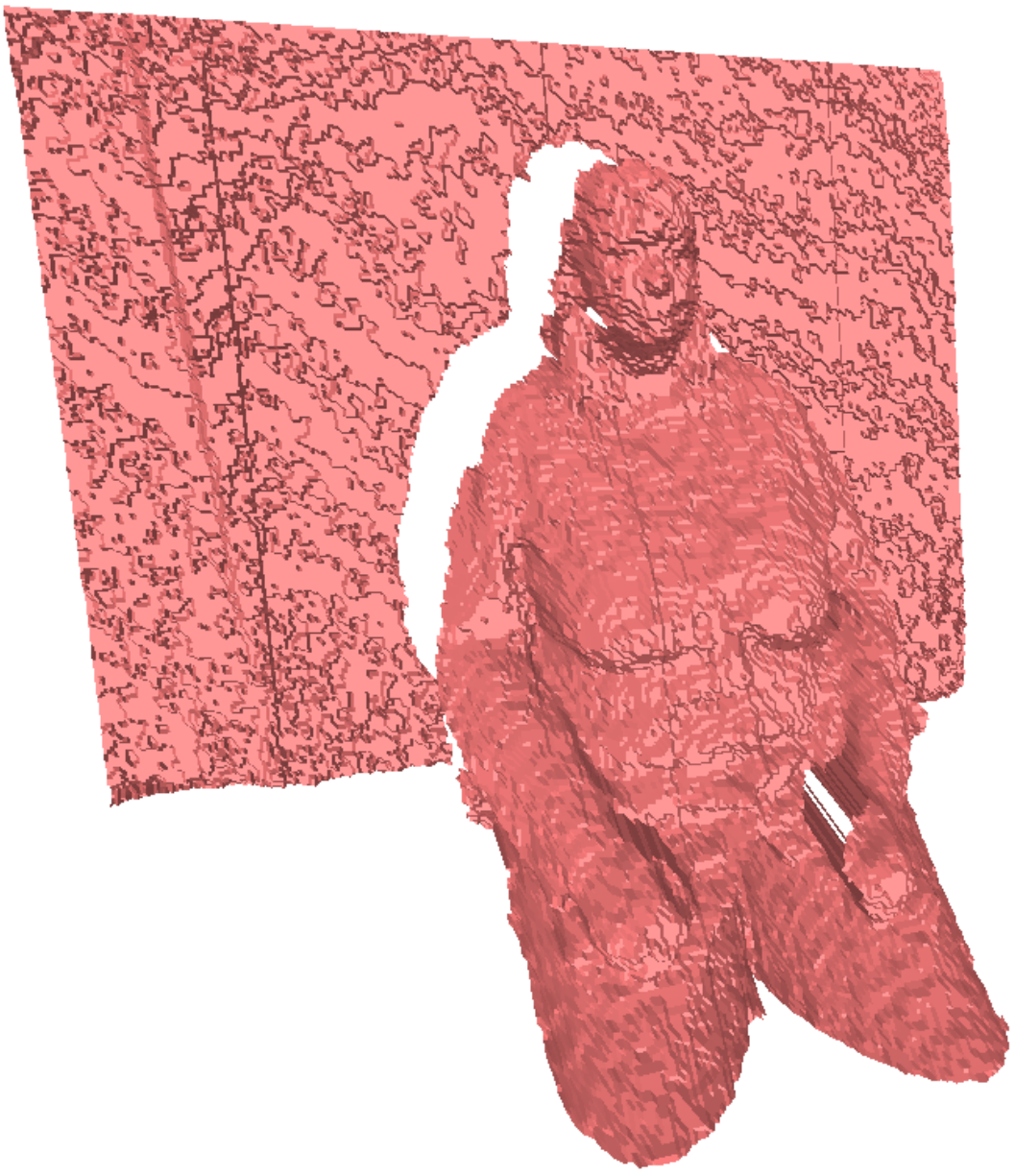


Fonte: Point clouds corresponding to dynamic gestures registered by Kinect

Algoritmo 2 – Criação de triângulos na malha.

$Pixel_{\mathrm{i\,j}} \leftarrow DepthMap_{\mathrm{ij}}$
$Pixel_{\mathrm{i+1\,j}} \leftarrow DepthMap_{\mathrm{i+1\,j}}$
$Pixel_{\mathrm{i\,j+1}} \leftarrow DepthMap_{\mathrm{i\,j+1}}$
$Pixel_{\mathrm{i+1\,j+1}} \leftarrow DepthMap_{\mathrm{i+1\,j+1}}$
$valid\_pixel1 \leftarrow Pixel_{\mathrm{ij}} != 0$
$valid\_pixel2 \leftarrow Pixel_{\mathrm{i+1\,j}} != 0$
$valid\_pixel3 \leftarrow Pixel_{\mathrm{i\,j+1}} != 0$
$valid\_pixel4 \leftarrow Pixel_{\mathrm{i+1\,j+1}} != 0$
$if\ (valid\_pixel1\ \&\&\ valid\_pixel2\ \&\&\ valid\_pixel3\ \&\&\ distance(Pixel_{\mathrm{i\,j}}, Pixel_{\mathrm{i+1\,j}}) <= minimum\_distance\ \&\&\ distance(Pixel_{\mathrm{i+1\,j}}, Pixel_{\mathrm{i\,j+1}}) <= minimum\_distance\ \&\&\ distance(Pixel_{\mathrm{i+1\,j}}, Pixel_{\mathrm{i\,j+1}}) <= minimum\_distance))$
  $build\_triangle1()$
$if\ (valid\_pixel2\ \&\&\ valid\_pixel3\ \&\&\ valid\_pixel4\ \&\&\ distance(Pixel_{\mathrm{i+1\,j}}, Pixel_{\mathrm{i\,j+1}}) <= minimum\_distance\ \&\&\ distance(Pixel_{\mathrm{i\,j+1}}, Pixel_{\mathrm{i+1\,j+1}}) <= minimum\_distance\ \&\&\ distance(Pixel_{\mathrm{i+1\,j+1}}, Pixel_{\mathrm{i+1\,j}}) <= minimum\_distance))$
  $build\_triangle2()$

Figura 42 – Mesh reconstruída.



Fonte: Elaborada pelo autor.

ANEXO

A

LISTA DE PUBLICAÇÕES OBTIDAS

- Lucas Melchiori Pereira, Sheila Walbe Ornstein, Vitória Sanches Lemes Soares, **Jean Amaro**, Ana Judite Galbiatti Limongi França. ***Congruence Mapping of the Activity Flows Allocated in Built Environments: A Pilot Application of Under-Development Software in an Emergency-Care Service.*** In: New Technologies Applied on Built Environment to Enhance Well-Being and Safety (Appl. Sci.), 2023, 13, 1599

- Diego Renan Bruno, Lucas Peres Nunes Matias, **Jean Amaro**, Fernando Santos Osório, Denis Fernando Wolf. ***Computer Vision System with 2D and 3D Data Fusion for Detection of Possible Auxiliaries Routes in Stretches of Interdicted Roads.*** In: Hawaii International Conference on System Sciences (HICSS), 2019, p. 1-10

- Mariana Rodrigues, **Jean Amaro**, Fernando Santos Osório, Kalinka Regina Lucas Jaquie Castelo Branco. ***Authentication Methods for UAV Communication.*** In: International Symposium on Computers and Communications (ISCC), 2019, p. 1210-1215

- José Diaz Amado, Iago Pachêco Gomes, **Jean Amaro**, Denis Fernando Wolf, Fernando Santos Osório. ***End-to-End Deep Learning Applied in Autonomous Navigation using Multi-Cameras System with RGB and Depth Images.*** In: Intelligent Vehicles Symposium (IVS), 2019, p. 1626-1631

- Diego Renan Bruno, Daniel Oliva Sales, **Jean Amaro**, Fernando Santos Osório. ***Analysis and fusion of 2D and 3D images applied for detection and recognition of traffic signs using a new method of features extraction in conjunction with Deep Learning.*** In: International Joint Conference on Neural Networks (IJCNN), 2018, p. 1-8

- Daniel Oliva Sales, **Jean Amaro**, Fernando Santos Osório. ***3D shape descriptor for objects recognition.*** In: Latin American Robotics Symposium and Brazilian Symposium on Robotics (LARS/SBR), 2017, p. 1-6

- Gabriele do Rosário Landim, Dyego da Silva Digiandomenico, **Jean Amaro**, Anja Pratschke, Marcelo Claudio Tramontano, Claudio Fabiano Motta Toledo. ***Architectural Optimization and Open Source Development: Nesting and Genetic Algorithms.*** In: Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA), 2017, p. 340-349


ICMC
USP
SÃO CARLOS