



UNIVERSIDADE FEDERAL DO RIO GRANDE - FURG
CENTRO DE CIÊNCIAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO
CURSO DE MESTRADO EM ENGENHARIA DE COMPUTAÇÃO

Dissertação de Mestrado

Ajuste de Primitivas Geométricas a Nuvens de Pontos de Larga Escala utilizando Aprendizado Profundo

Igor Pardo Maurell

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal do Rio Grande - FURG, como requisito parcial para a obtenção do grau de Mestre em Engenharia de Computação

Orientador: Prof. Dr. Paulo Lilles Jorge Drews Junior

Rio Grande, 2024

Ficha Catalográfica

M453a Maurell, Igor Pardo.
Ajuste de primitivas geométricas a nuvens de pontos de larga escala utilizando aprendizado profundo / Igor Pardo Maurell. – 2024.
123 f.

Dissertação (mestrado) – Universidade Federal do Rio Grande – FURG, Programa de Pós-Graduação em Computação, Rio Grande/RS, 2024.
Orientador: Dr. Paulo Lilles Jorge Drews Junior.

1. Aprendizado Profundo Geometrico 2. *As Built* 3. Larga Escala 4. Nuvens de Pontos 3D 5. Primitivas Geométricas I. Drews Junior, Paulo Lilles Jorge II. Título.

CDU 004

Catalogação na Fonte: Bibliotecário José Paulo dos Santos CRB 10/2344

Universidade Federal do Rio Grande
Centro de Ciências Computacionais
Programa de Pós-Graduação em Computação
Curso de Mestrado em Engenharia de Computação

DISSERTAÇÃO DE MESTRADO

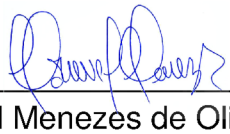
Ajuste de Primitivas Geométricas a Nuvens de Pontos de Larga Escala utilizando Aprendizado Profundo

Igor Pardo Maurell

Banca examinadora:



Prof. Dr. Antonio Wilson Vieira (Unimontes)



Prof. Dr. Manuel Menezes de Oliveira Neto (UFRGS)

Documento assinado digitalmente
gov.br MARCELO DE GOMENSORO MALHEIROS
Data: 26/02/2024 09:03:29-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Marcelo de Gomensoro Malheiros (FURG)



Assinado de forma digital por Paulo Lilles Jorge Drews Junior:00124140009
Dados: 2024.02.16 14:46:59 -03'00'

Prof. Dr. Paulo Lilles Jorge Drews Junior (FURG)
Orientador

RESUMO

MAURELL, Igor Pardo. **Ajuste de Primitivas Geométricas a Nuvens de Pontos de Larga Escala utilizando Aprendizado Profundo**. 2024. 122 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Computação. Universidade Federal do Rio Grande - FURG, Rio Grande.

O presente trabalho se contextualiza na temática da realização de *As Built* automático de grandes estruturas industriais, principalmente as de óleo e gás, como plataformas de petróleo *off-shore*, FPSO e refinarias. Nesse contexto, apesar da captura de nuvens de pontos 3D já gerar uma representação digital dessas estruturas, a reconstrução de um modelo paramétrico a partir desses dados permite uma comparação mais precisa entre a estrutura como está construída e como foi projetada inicialmente. Entre os modelos paramétricos possíveis de serem usados, a Representação por Contornos (B-Rep) é um formato CAD que permite descrever a estrutura a partir de múltiplas geometrias interconectadas. Nesse contexto, o presente trabalho propõe um método para identificar as geometrias presentes em nuvens de pontos dessas estruturas industriais de larga escala.

Ao se explorar a literatura, é possível compreender que, apesar desse problema não ser amplamente explorado em contextos de larga escala, os principais métodos de pequena escala, progridem na direção do uso de técnicas baseadas em aprendizado profundo supervisionado [31, 55, 63]. Apesar do problema ser consideravelmente mais complexo em estruturas de larga escala, um fator determinante para desenvolver métodos baseados em aprendizado é a existência de um *dataset* que permita o treinamento e validação dos mesmos. Esse requisito já foi superado no contextos de pequena escala a partir da publicação do *Dataset* ABC [27], entretanto, esse *dataset* não conta com estruturas como as que motivam o trabalho, não sendo adequado para utilização no mesmo. Logo, a produção de um *dataset* que possa ser usado no contexto de aplicação desejado também se torna um requisito importante do trabalho.

A abordagem adotada para geração dos dados a serem utilizados foi a geração sintética dos mesmos utilizando modelos CAD como dado de entrada. Dessa forma, as geometrias presentes nesses modelos paramétricos são extraídas e utilizadas para gerar o *ground truth* geométrico, ou seja, uma anotação de todas as geometrias presentes no modelo com seus parâmetros e tipos associados. Conjuntamente, a geração de nuvens de pontos desses modelos CAD é feita a partir da simulação de um sensor LiDAR coletando uma quantidade numerosa de pontos a partir de múltiplas vistas ao redor do modelo de entrada, a qual é uma abordagem autoral e uma das contribuições do trabalho. Por consequência, o *Dataset* LS3DS foi produzido pelo trabalho, o qual conta com 86 modelos CAD de larga escala de contexto industrial e também suas respectivas nuvens de pontos e *ground truth* de geometrias.

Com um *dataset* de larga escala disponível, o presente trabalho discute sobre as características principais que diferenciam os contextos de larga escala, como as plantas industrias, nesse caso, e os contextos de pequena escala, como peças mecânicas e objetos individuais. Neste escopo, entre outras diferenças importantes, a quantidade de pontos da nuvem para que se possa representar as principais geometrias com resolução adequada surge como a mais impactante. Logo, somada a limitação severa de número de pontos máximo associada a esses métodos baseados em aprendizado, o trabalho propõe o uso estratégias de divisão da nuvem original em pequenas partes para serem processadas separadamente. Assim, a metodologia nomeada Ajuste de Geometrias por Divisão e Agregação (SMGF) é apresentada, a qual busca adaptar os principais métodos da literatura de ajuste de primitivas geométricas a nuvens de pontos de pequena escala [55, 63, 24], para os contextos de larga escala.

Com base nisso, diversas versões e configurações dessa metodologia são comparadas de maneira qualitativa e quantitativa, construindo um *benchmark* de ajuste de geometrias de larga escala. Nesse escopo, é demonstrado que todas as abordagens que utilizam a metodologia SMGF superam o estado-da-arte no contexto de larga escala em até **36%** pontos nas métricas utilizadas, que é o ERANSAC [53]. Junto a isso, se demonstra de maneira clara a importância da resolução da nuvem de pontos na qualidade dos resultados dos métodos baseados em aprendizado e as limitações associadas ao aumento do número de pontos de cada versão. Ao fim, é feita uma discussão sobre os experimentos e os possíveis trabalhos futuros são apresentados, os quais podem se beneficiar do *Dataset* LS3DS e da metodologia SMGF para desenvolver soluções ainda melhores para o problema central do trabalho.

Palavras-chave: Aprendizado Profundo Geométrico, *As Built*, Larga Escala, Nuvens de Pontos 3D, Primitivas Geométricas.

ABSTRACT

MAURELL, Igor Pardo. **Geometric Primitive Fitting to Large-Scale Point Clouds using Deep Learning**. 2024. 122 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Computação. Universidade Federal do Rio Grande - FURG, Rio Grande.

The following work is based on the theme of automatic AsBuilt design of large industrial structures, mainly oil and gas structures such as oil platforms, FPSOs and refineries. In this context, although capturing 3D point clouds already generates a digital representation of these structures, reconstructing a parametric model from this data allows more accurate comparisons between the structure as built and as designed. Among many parametric models that can be used, Boundary Representation (B-Rep) is a CAD format that allows the structure to be described from multiple interconnected geometries. In this context, this paper proposes a method for identifying the geometries present in point clouds of these large-scale industrial structures.

By exploring the literature, it is possible to understand that, although this problem is not widely explored in large-scale contexts, the main small-scale methods are progressing towards the use of techniques based on supervised deep learning [31, 55, 63]. Although the problem is considerably more complex in large-scale, a determining factor in developing learning-based methods is the existence of a dataset that allows them to be trained and validated. This requirement has already been overcome in small-scale contexts with the publication of the ABC Dataset [27], however, this dataset does not have structures such as those that motivate this work and is not suitable for use in it. Therefore, the production of a dataset that can be used in the desired application context also becomes an important requirement of the work.

The approach adopted to generate the data to be used was to generate it synthetically using CAD models as input data. In this way, the geometries present in these parametric models are extracted and used to generate the geometric ground truth, i.e. an annotation of all the geometries present in the model with their associated parameters and types. At the same time, the point clouds of these CAD models are generated by simulating a LiDAR sensor collecting a large number of points from multiple views around the input model, which is an authorial approach and one of the contributions of this work. As a result, the LS3DS Dataset was produced by the work, which includes 86 large-scale CAD models from an industrial context, as well as their respective point clouds and ground truth geometries.

With a large-scale dataset available, this paper discusses the main characteristics that differentiate large-scale contexts, such as industrial plants in this case, and small-scale contexts, such as mechanical parts and individual objects. In this context, among other

important differences, the number of points in the point cloud to be able to represent the main geometries with adequate resolution is the most significant one. Therefore, in addition to the severe limitation of the maximum number of points associated with these learning-based methods, the work proposes the use of strategies for dividing the original cloud into small parts to be processed separately. Thus, the methodology called Split and Merge Geometry Fitting (SMGF) is presented, which seeks to adapt the main methods in the literature for fitting geometric primitives from small-scale contexts [55, 63, 24], to large-scale ones.

On this basis, various versions and configurations of this methodology are compared qualitatively and quantitatively, building a benchmark for adjusting large-scale geometries. Within this scope, it is shown that all approaches using the SMGF methodology outperform the state-of-the-art in the large-scale context by up to **36%** points in the metrics used, which is ERANSAC [53]. It also clearly demonstrates the importance of point cloud resolution in the quality of the results of learning-based methods and the limitations associated with increasing the number of points in each version. Finally, a discussion of the experiments is made and possible future work are presented, which could benefit from the LS3DS dataset and the SMGF methodology to develop even better solutions to the main problem of the work.

Keywords: Geometric Deep Learning, As Built, Large-Scale, 3D Point Clouds, Geometric Primitives.

LISTA DE FIGURAS

1 Nuvem de Pontos de larga escala de uma plataforma de extração de petróleo adquirida a partir do processo de fotogrametria em imagens capturadas por uma câmera acoplada a um drone. 15
2 Hierarquia de um arquivo STEP AP 203. [2] 16
3 Forma básica de quatro tipos diferentes de primitivas geométricas. . . 17
4 Diferentes tipos de Segmentação em Geometrias. 20
5 Diagrama dos diferentes formatos que fluem no *Software* de geração de dados. Em **A**, se tem um CAD B-Rep que serve como entrada para o sistema. Em **B**, se tem a lista de geometrias obtidas a partir da exploração topológica do arquivo CAD, as quais são armazenadas em uma estrutura chave-valor e serializadas em um arquivo. Em **C**, um processo de triangulação gera uma malha triangular como representação 3D parcial. Ao fim, em **D** as nuvens de ponto com *ground truth* são geradas utilizando duas abordagens possíveis, que são Simulação de LiDAR e Amostragem Uniforme. 39
6 Malhas triangulares geradas para um CAD simples de um cilindro e um cubo. Em **(a)** é possível ver a malha gerada pela Gmsh API, a qual tem triângulos mais regulares e vértices se distribuindo ao longo das faces. Já em **(b)**, é possível ver a malha gerada pela OCCT, com vértices apenas nas arestas e triângulos que se expandem entre eles ao logo de toda extensão das faces. 42
7 Processo de Tesselação de Superfícies feito pelo OCCT [51]. A superfície contínua em amarelo está sendo recoberta por polígonos como o triângulo em verde. A distância d é limitada no processo pelo parâmetro de deflexão linear. Da mesma forma, o maior desvio angular entre os pares dos vetores normais $N1$, $N2$ e $N3$ é limitado pelo parâmetro de deflexão angular. [52] 43
8 Malha triangular de uma cena do *dataset* Scannet [12], na qual é possível observar qual o padrão de formação e distribuição dos triângulos em malhas geradas a partir de dados reais. 44
9 Malha triangular (a) e nuvem de pontos (b) geradas a partir da ferramenta. A cor dos pontos demonstra a relação mantida entre as duas representações 3D, identificando em qual triângulo cada ponto foi gerado. 46
10 Semi-Elipsoide ajustado para cobrir toda a estrutura representada por uma malha triangular. 49

11 Pontos de Vista para o Drone dispostos ao longo da superfície do Semi-elipsoide. 50
12 Nuvens de Pontos capturadas em diferentes pontos de vistas de observação do *Drone*. Em **(a)** é possível ver um ponto de vista frontal, indicado pelo *drone* em roxo na parte inferior esquerda da imagem. Já em **(b)** é possível observar um ponto de vista traseiro, indicado pelo *drone* em rosa atrás da estrutura. 51
13 Oclusão gerada por partes que são internas à outras. Em (a) pode-se ver o modelo CAD como entra no gerador de *datasets*. Já em (b), pode-se ver a visualização desse mesmo modelo com um grau de explosão aplicado, possibilitando a visualização de algumas partes internas. 53
14 Cena do *Dataset* SemanticKITTI [4] de uma nuvem de pontos capturada por um LiDAR 360° a partir de um único ponto de vista em cenário urbano. As marcações em azul mostram as regiões de maior esparsidade, já as em vermelho mostram regiões oclusas que não puderam ser observadas pelo sensor. 53
15 Oclusão e Esparsidade Local exemplificados em nuvens de pontos de um modelo CAD geradas a partir da técnica de Simulação de LiDAR por *Ray Casting*. 55
16 Nuvens de pontos geradas sobre a malha triangular de um modelo CAD. Ambas nuvens tem $2.878.465$ pontos. Em (a) é possível ver uma nuvem gerada a partir do processo de amostragem uniforme, sem oclusão ou esparsidade local. Já em (b), é possível ver uma nuvem gerada a partir do processo de simulação de LiDAR através de *Ray Casting*, que conta com trechos mais esparsos (cor mais fraca) e com regiões oclusas (brancas). 56
17 Processo de *Points Healing* aplicado a uma nuvem de pontos de uma esfera amostrada uniformemente em uma malha triangular de baixa resolução da mesma. 57
18 Ilustação so processo de adição de ruído artificial de $\pm 10°$ nos vetores normais dos pontos. 59
19 Proposta *"Split and Merge Geometry Fitting"(SMGF)* para resolver o problema de ajuste de primitivas geométricas à nuvens de pontos de larga escala. 62
20 Nuvens de pontos amostradas uniformemente na malha triangular de uma estrutura de larga escala. As nuvens estão segmentadas por instâncias de geometria de uma estrutura com 67.899 superfícies geométricas. Partindo de uma quantidade de 10 milhões de pontos em (a), até 10 mil em (d), é possível notar uma redução da capacidade de representar geometrias, iniciando em 28.153 geometrias e finalizando com apenas 156 restantes. 64
21 Processo de divisão ilustrado utilizando uma nuvem de pontos gerada pelo trabalho em cima de uma estrutura de um heliponto. As nuvens completas (a) e (b) contam com $4,4$ milhões de pontos. 66
22 Modelo CAD Utilizado no Estudo de Caso Simplificado. 75

23 Gráfico do Número de Triângulos por Tipo de Primitiva no Estudo de Caso Simplificado. 76
24 Gráfico do Número de Instâncias por Tipo de Primitiva no Estudo de Caso Simplificado. 76
25 Nuvens de Ponto Gerada no Estudo de Caso Simplificado com Cores Representando Diferentes Informações do *Dataset*. Em **(a)**, apenas uma cor baseada na altura de cada ponto, em **(b)**, a cor representa o identificador do triângulo, em **(c)**, representa a instânica geométrica e em **(d)**, o tipo de primitiva geométrica. 78
26 Gráficos *Box Plot* da distribuição dos erros de distância e normal ponto-primitiva. Os erros são mostrados por instância geométrica. . . 79
27 Multiplicidade de escalas dos modelos do *Dataset LS3DS*. Em **(a)** está demonstrado o modelo com menor área de base, cerca de $5\ m^2$. Já em **(b)** é possível observar o modelo com maior área de base, cerca de $2000\ m^2$. 81
28 Quantidade de entidades Geométricas de cada tipo no *Dataset* LS3DS. 82
29 Distribuição da área das superfícies por tipo ao longo dos modelos do *Dataset*. 83
30 Nuvem de pontos do maior modelo do *Dataset* LS3DS com o *ground truth* para segmentação de tipos geométricos e de instâncias Geométricas. 85

31 Resultados de Segmentação de Instâncias Geométricas do Experimento 2. 110
32 Resultados de Segmentação de Tipos Geométricos do Experimento 2. 111
33 Resultados de Segmentação de Instâncias Geométricas do Experimento 2. 112
34 Resultados de Segmentação de Tipos Geométricos do Experimento 2. 113
35 Resultados de Segmentação de Instâncias Geométricas do Experimento 2. 114
36 Resultados de Segmentação de Tipos Geométricos do Experimento 2. 115

37 Resultados de Segmentação de Instâncias Geométricas do Experimento 3. 117
38 Resultados de Segmentação de Tipos Geométricos do Experimento 3. 118
39 Resultados de Segmentação de Instâncias Geométricas do Experimento 3. 119
40 Resultados de Segmentação de Tipos Geométricos do Experimento 3. 120
41 Resultados de Segmentação de Instâncias Geométricas do Experimento 3. 121
42 Resultados de Segmentação de Tipos Geométricos do Experimento 3. 122

LISTA DE TABELAS

1 Descrição dos Métodos a serem comparados nos experimentos. . . . 86
2 Características dos *datasets* gerados utilizando a metodologia SMGF com diferentes parâmetros no *Dataset* LS3DS. 86
3 Percentual de Perda de Pontos (PLR) e Percentual de Perda de Geometrias (GLR) de cada um dos *datasets* gerados para os experimentos. 91
4 Validação nas Partes no *Dataset* LS3DS-4-7K. 92
5 Validação Completa no *Dataset* LS3DS-4-7K. 92
6 Validação nas Partes dos métodos nos diferentes *datasets* que contam com 7.000 pontos em cada parte. 94
7 Validação Completa dos métodos nos diferentes *datasets* que contam com 7.000 pontos em cada parte. 95
8 Validação nas Partes dos métodos nos diferentes *datasets* que foram divididos usando cuboides de dimensão $4 \times 4 \times 4$. 97
9 Validação Completa dos métodos nos diferentes *datasets* que foram divididos usando cuboides de dimensão $4 \times 4 \times 4$. 98

LISTA DE ABREVIATURAS E SIGLAS

CAD	Desenho Assistido por Computador (*Computer-Aided Design*)
B-Rep	Representação por Contornos (*Boundary Representation*)
LiDAR	Detecção e Medição por Luz (*Light Detection and Ranging*)
SMGF	Ajuste de Geometrias por Divisão e Agregação (*Split and Merge Geometry Fitting*)
PLR	Percentual de Perda de Pontos (*Points Loss Rate*)
GLR	Percentual de Perda de Geometrias (*Geometries Loss Rate*)
AP	Protocolo de Aplicação (*Application Protocol*)
ISO	Organização Internacional de Padronização (*International Organization for Standardization*)
FPSO	Unidade Flutuante de Armazenamento e Transferência (*Floating Production Storage and Offloading*)
LS3DS	Sintético 3D de Larga Escala (*Large-Scale 3D Synthetic*)
STEP	Padrão para o Intercâmbio de Dados de Produtos (*Standard for the Exchange of Product model data*)
GNN	Redes Neurais de Grafos (*Graph Neural Networks*)
CNN	Redes Neurais Convolucionais (*Convolutional Neural Networks*)
OCCT	*Open Cascade Technology*
API	Interface de Programação de Aplicações (*Application Programming Interface*)
YAML	*YAML Ain't Markup Language*
JSON	Notação de Objeto JavaScript (*JavaScript Object Notation*)
PKL	*Python Pickle*
OBJ	Formato de Arquivo Wavefront OBJ (*Wavefront OBJ file format*)
PCA	Análise de Componentes Principais (*Principal Component Analysis*)
GPU	Unidade de Processamento Gráfico (*Graphics Processing Unit*)
VRAM	Memória de Acesso Aleatório de Vídeo (*Video Random Access Memory*)
IoU	Interseção sobre União (*Intersection over Union*)

SUMÁRIO

1 Introdução **13**
1.1 Contexto . . . 13
1.2 Motivação . . . 14
1.3 Problema da Pesquisa . . . 17
1.4 Justificativa . . . 18
1.4.1 Ajuste de Primitivas Geométricas na larga escala . . . 18
1.4.2 *Datasets* de larga escala com anotação Geométrica . . . 21
1.5 Objetivos . . . 22
1.6 Contribuições . . . 23
1.7 Estrutura do Trabalho . . . 23

2 Revisão Bibliográfica **24**
2.1 *Datasets* para Aprendizado e *Benchmark* em Nuvem de Pontos . . . 24
2.2 Aprendizado Profundo em Representações 3D . . . 26
2.3 Ajuste de Primitivas Geométricas em Representações 3D . . . 29
2.3.1 Métodos Clássicos . . . 29
2.3.2 Métodos Baseados em Aprendizado . . . 30

3 Geração de *Datasets* Sintéticos para Aprendizado Geométrico **36**
3.1 Exploração Topológica e Anotação de Geometrias . . . 38
3.2 Geração de Malha Triangular e Relação com as Geometrias . . . 41
3.3 Geração de Nuvem de Pontos . . . 45
3.3.1 Amostragem Uniforme de Pontos . . . 45
3.3.2 Simulação de LiDAR através de *Ray Casting* . . . 47
3.3.3 Diminuição da Lacuna entre dados Reais e dados Sintéticos . . . 52
3.4 Pós-processamentos . . . 57

4 Modelo de Ajuste de Primitivas Geométricas em Nuvens de Pontos de larga escala **60**
4.1 Ajuste de Primitivas Geométrica em Modelos de larga escala . . . 61
4.2 Divisão dos Modelos em Partes Menores . . . 65
4.3 Ajuste de Primitivas Geométricas nas Partes . . . 67
4.4 Agregação dos Resultados Obtidos nas Partes . . . 70

5 Resultados **74**
5.1 Validação da Geração de *Datasets* Sintéticos em Estudo de Caso Simplificado . . . 74

5.2 *Dataset* de Larga-escala de Modelos CAD 3D (LS3DS *Dataset*) 80
5.2.1 Nuvens de Pontos e *Ground Truth* para Treinamento e Validação 83
5.3 *Benchmark* de Ajuste de Primitivas Geométricas a Nuvens de Pontos de Larga-Escala 84
5.3.1 Experimento 1: validação de funcionamento 91
5.3.2 Experimento 2: dimensão das partes 93
5.3.3 Experimento 3: limitações na quantidade de pontos 96
5.3.4 Conclusões Obtidas a partir dos Experimentos 99

6 Conclusão **101**

Referências **103**

ANEXOS

A Resultados visuais do Experimento 2 **109**

B Resultados Visuais do Experimento 3 **116**

1 INTRODUÇÃO

1.1 Contexto

No meio industrial e da construção civil, realizar *As Built* de uma construção significa coletar dados métricos dela e, com base neles, desenvolver um projeto em 2D ou 3D da estrutura em questão o mais próximo possível de *como está construída*. Por se tratar de um processo longo e caro, porém altamente necessário e benéfico, o *As Built* automático é um desafio de extremo interesse em diversos setores da economia, dentre eles, a indústria e a construção civil. Quando este é feito de forma manual, sensores são utilizados por inspetores para realizar a medição dos aspectos físicos de construções de engenharia, sejam elas de larga escala, como plantas industriais ou construções civis, quanto de pequena escala, como peças mecânicas, instrumentos industriais e objetos individuais. Ao final do processo de medição, os dados são normalmente passados para projetistas que se utilizam dos mesmos para desenhar alguma forma de representação em Desenho Auxiliado por Computador (CAD) da estrutura em questão de uma maneira mais próxima possível de como ela está construída. Analisando de forma mais específica o problema no caso das estruturas de larga escala, a medição manual em conjunto com o requisito de um modelo 3D detalhado tornam o *As Built* uma tarefa longa, custosa e extremamente suscetível ao erro humano. Dessa forma, o desenvolvimento de um método automático que realize essa tarefa é de extrema importância para os setores industriais das mais diversas vertentes, bem como, para o setor da construção civil.

Para enfrentar o problema de *As Built* automático de estruturas de larga escala, surge o projeto *AeroScan*, uma iniciativa do Instituto Tecnológico de Aeronáutica (ITA) em parceria com a Universidade Federal do Rio Grande (FURG), que tem como principal objetivo a construção de uma solução tecnológica completa utilizando UAV's para resolver esse problema no contexto das plataformas de petróleo *off-shore*. Nas bases do projeto, a equipe projeta e constrói o drone com os sensores necessários para que possa obter uma nuvem de pontos completa da estrutura a ser reconstruída, como exemplificado na Figura 1. Logo após, como a reconstrução completa leva considerável tempo de voo e requer que o UAV colete dados da estrutura a partir de múltiplas vista, efeitos do vento causam

movimentos involuntários tanto no veículo quanto na plataforma, estabelecendo também a necessidade de algoritmos de correção da nuvem, requisito que também é coberto pelo projeto. Apesar da própria nuvem de pontos já ter capacidade de representar digitalmente a estrutura em questão, o projeto ainda visa ir um nível além, buscando maneiras mais otimizadas e escaláveis de representar as estruturas para simplificar análises e edições futuras. O presente trabalho se contextualiza dentro desta última problemática apresentada, no qual se estabelece como premissa que a nuvem de pontos já foi capturada e corrigida, restando o problema de transformação da mesma para uma representação otimizada e de mais fácil manuseio. Sharma et al. [55] traz os arquivos CAD como uma representação com essas características, visto que é baseada em três propriedades geométricas muito importantes, são elas: continuidade, editabilidade e flexibilidade. Contudo, é importante pontuar que esta é uma representação majoritariamente descrita por geometrias, como linhas, círculos, planos, cilindros e outras, conduzindo a solução do problema ao processo de busca das geometrias que estão subjacentes nos pontos da nuvem reconstruída pelo sistema do projeto. A resolução desse problema, denominado como ajuste de primitivas geométricas a nuvens de pontos, foi levantado por Pătrăucean et al. [41] como parte fundamental de muitas propostas para uma solução automatizada do *As Built*.

1.2 Motivação

Uma das grandes utilidades do *As Built* é fornecer um modelo paramétrico que possibilite a medição facilitada das partes construídas para que se possa conferir o grau de correspondência com o projeto original. Dentro dos muitos tipos de modelo paramétrico 3D, a Representação pelos Contornos (B-Rep) é o tipo utilizado pelos formatos padrões de modelo CAD. Essa representação descreve o modelo 3D a partir da relação topológica e delimitação entre entidades geométricas de múltiplas dimensões. Englobados nesse tipo de representação, estão os modelos em formato STEP. Como definido por Qin et al. [46], STEP é uma norma ISO (ISO 10303) legível por computador, que representa e faz intercâmbio de dados de produtos ao longo de todo o ciclo de vida dos mesmos. Essa representação incluí: desenho, manufatura, controle de qualidade, inspeção e suporte. Ao longo da história dessa norma ISO, múltiplos protocolos de aplicação (APs) foram publicados para representar dados de produtos de aplicações industriais. A partir do AP 203 [25], sendo atualizado no AP214 e AP242, o formato STEP se tornou um padrão para intercâmbio de arquivos CAD. A Figura 2 representa a estrutura topológica de um arquivo STEP AP 203. Como descrito por Al-wswasi and Ivanov [2], arquivos do formato STEP AP 203 são representados a partir de múltiplas partes descritas por uma mesma hierarquia que começa com uma casca fechada, ou *Closed Shell*, que é contornado por múltiplas faces, ou *Faces*, as quais são contornadas por arestas, ou *Edges*, que por sua vez são limitadas por vértices, ou *Cartesian Points*.



Figura 1: Nuvem de Pontos de larga escala de uma plataforma de extração de petróleo adquirida a partir do processo de fotogrametria em imagens capturadas por uma câmera acoplada a um drone.

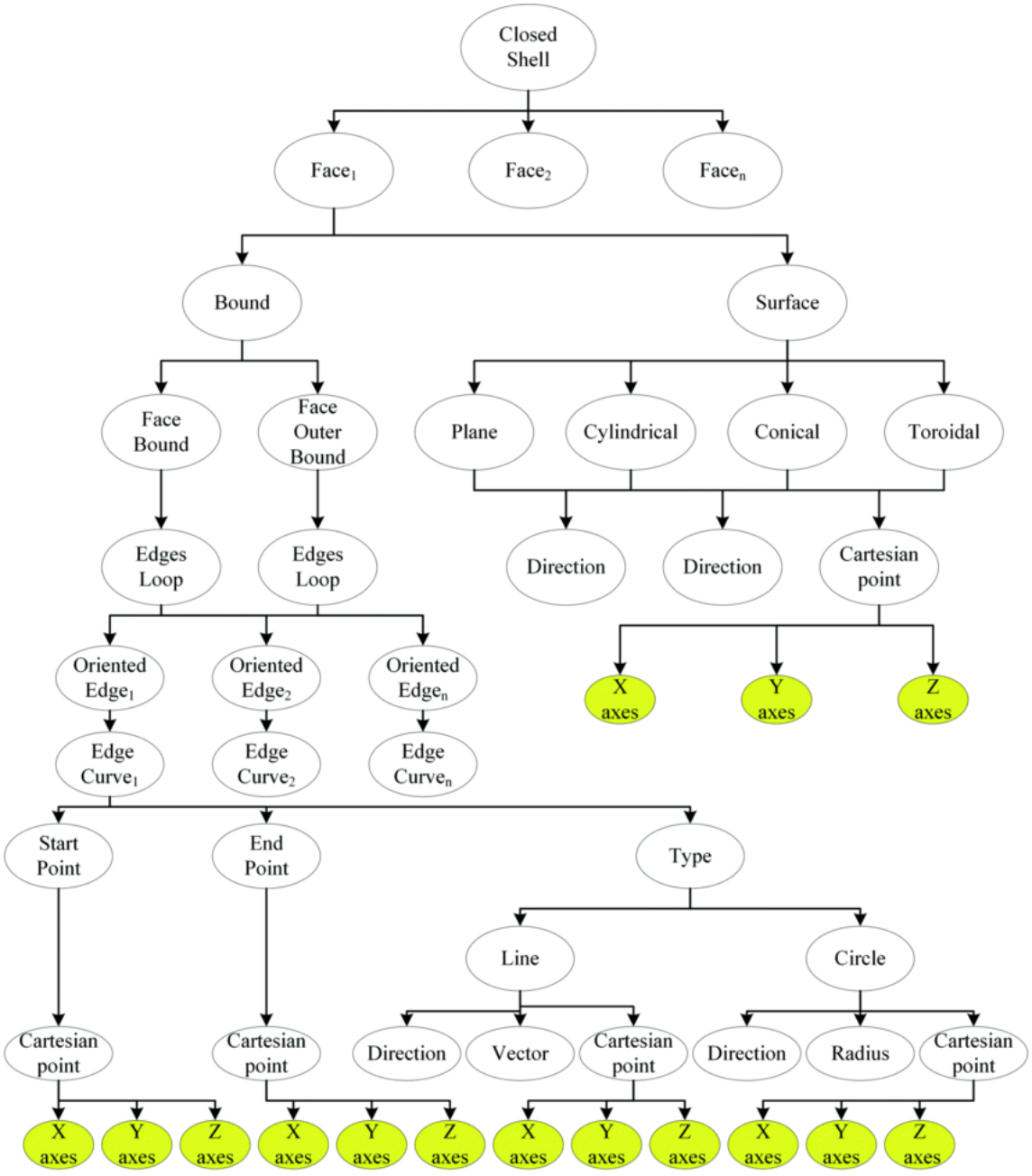


Figura 2: Hierarquia de um arquivo STEP AP 203. [2]

Analisando detalhadamente a Figura 2, é possível observar que os modelos STEP são descritos em sua base por as entidades geométricas de tipos primitivos de superfícies, como planos e cilindros, e curvas, como linhas e círculos. Entretanto, mesmo que se encontre todas as geometrias presentes na nuvem em um processo de transformação para STEP, ainda assim haveria informação faltante para se construir um modelo B-Rep. Porém, é lógico afirmar que quanto mais alta a dimensão da entidade identificada na nuvem maior é seu nível na hierarquia e, logo, menos eurísticas são necessárias para montar um arquivo STEP e fechar o ciclo do processo de *As Built*. Analisando os nós mais próximos da raiz da árvore da Figura 2 estão Cascas Fechadas, as quais apesar de serem o elemento topológico de mais alto nível, não são explicitamente descritas por primitivas geométricas. Seguindo, temos as *Faces*, que são descritas pelas *Bounds* e pela *Surface*. Nesse contexto, como as *Bounds* só vem a ser definidas como geometrias na profundidade 8 da árvore, o ramo que descreve as *Faces* como superfícies geométricas, denominado *Surface*, categoriza as entidades de maior dimensão que são explicitamente descritas a partir de um tipo de geometria com parâmetros associados. Todavia, mesmo encontrando todas as superfícies geométricas ainda faltam informações para completar a montagem dos arquivos STEP, como por exemplo o ramo de *Bound*s anteriormente comentado, o qual delimita as *Faces* através de uma série de *Edges* que, por sua vez são delimitadas por um *Start Point* e um *End Point*. Apesar disso, o que motiva o presente trabalho é fornecer, para uma solução fim-a-fim de *As Built* a ser desenvolvida pelo projeto, as entidades geométricas de mais alta dimensão possíveis que podem ser descritas através de primitivas geométricas na topologia dos arquivos STEP, ou seja, as superfícies geométricas. Alguns tipos dessas geometrias estão exemplificadas em conjunto com seus parâmetros na Figura 3.

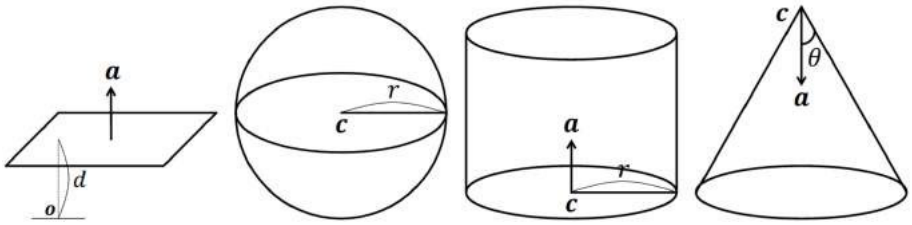


Figura 3: Forma básica de quatro tipos diferentes de primitivas geométricas. Em ordem da direita para a esquerda, se tem: plano, representado pela distância da origem e vetor normal; esfera, representada por centroide e raio; cilindro, representado pelo centroide da base, raio e eixo de revolução; cone, representado pelo centroide da base, eixo de revolução e ângulo de abertura. [32]

1.3 Problema da Pesquisa

A forma mais fundamental de representação 3D são as nuvens de pontos. Esse padrão de dados consiste em um vetor no qual suas n posições abrigam pontos do espaço 3D,

os quais podem ser representados não somente por suas coordenadas cartesianas, como também opcionalmente por outras características, como: cor, intensidade, vetor normal, curvatura da superfície, etc. Dessa forma, o problema de ajuste de primitivas geométricas consiste em segmentar o vetor de entrada em geometrias de diferentes tipos, que estão implícitas na representação da estrutura por nuvem de pontos, seguido pelo ajuste dos parâmetros de cada instância geométrica encontrada. De maneira formal, estendendo a definição de Schnabel et al. [53], dado um conjunto P de n pontos representados por suas posições cartesianas $P = \{p_1, p_2, ..., p_n\}$, com opcionalmente outras características como cor ou vetor normal, associadas $F = \{f_1, f_2, ..., f_n\}$, o algoritmo deve produzir um conjunto S de k primitivas $S = \{s_1, s_2, ..., s_k\}$, as quais são representadas por um tipo, pelos parâmetros associados ao tipo, também exemplificados na Figura 3, e pelos seus respectivos conjuntos de pontos, onde $\{P_{s_1} \subset P, P_{s_2} \subset P, ..., P_{s_k} \subset P\}$. Contudo, além da definição formal do problema, cabe ressaltar que o presente trabalho visa resolve-lo para contextos de larga escala, onde nuvens de altíssima dimensão, número de pontos e quantidade de geometrias tornam a solução do ajuste de primitivas geométricas um problema ainda mais desafiador.

1.4 Justificativa

O presente trabalho utiliza técnicas de aprendizado profundo como ferramenta para o desenvolvimento de um método para lidar com o problema definido. Dessa forma, a Seção 1.4.1 fará um detalhamento mais profundo do problema de ajuste de primitivas geométricas, correlacionando com os problemas comuns abordados pelas técnicas de aprendizado. Junto a isso, por se tratar de um problema normalmente enfrentado utilizando técnicas de aprendizado supervisionado (Sec. 2.3), a construção de um *dataset* é parte fundamental do trabalho, o que será justificado na Seção 1.4.2.

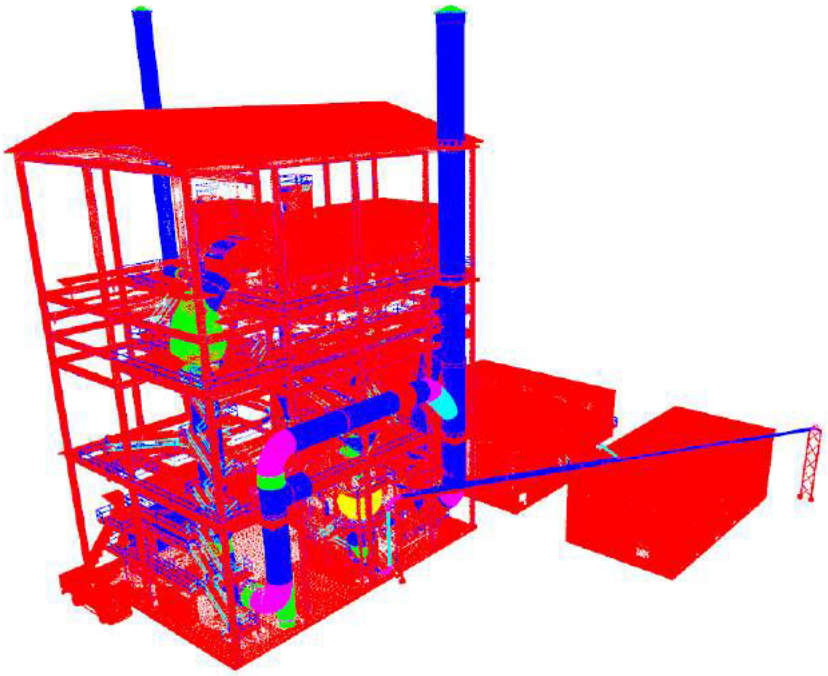
1.4.1 Ajuste de Primitivas Geométricas na larga escala

O ajuste de primitivas geométricas a nuvens de pontos não é somente um problema de ajuste de parâmetros com base nos pontos e na equação geral da geometria, ele também envolve a segmentação dos pontos em diferentes tipos e instâncias geométricas. Assim, relacionando com os grandes tópicos explorados por técnicas de aprendizado profundo, o sub-problema de segmentação não é um problema de segmentação semântica, ilustrado Figura 4a, o qual teria como objetivo nesse contexto a atribuição de um tipo de geometria para cada ponto da nuvem de entrada. Ao contrário, o problema de segmentação é, por essência, um problema de segmentação de instâncias, ilustrado na Figura 4b, visto que as diferentes instâncias de um mesmo tipo de geometria representam partes completamente diferentes da estrutura representada pelos pontos e, assim, serão ajustadas a parâmetros com valores distintos. Contudo, é necessário justificar que, dentro da

segmentação de instâncias, cada entidade encontrada também recebe um tipo, habilitando a geração também de uma ilustração como a da Figura 4a.

Embora seja um problema composto, não é possível definir de maneira ótima qual tarefa deve ser realizada primeiro, o que categoriza o problema de ajuste de primitivas geométricas a nuvens de pontos como um problema de interdependência causal, como dito por Li and Feng [31]. Essa constatação se dá pelo fato de que, tendo os parâmetros de todas as primitivas, a segmentação se torna um problema de associação ponto-primitiva, que pode ser solucionado a partir de métricas simples de distância ponto-superfície. Por outro lado, no caso de se ter uma segmentação perfeita dos pontos, diferentes métodos robustos de ajuste paramétrico podem ser aplicados, os quais gerarão parâmetros de qualidade para a distribuição dos dados de entrada e para as instâncias e tipos dispostos na segmentação. Nesse contexto, é importante pontuar que o problema de ajuste paramétrico de uma única primitiva a um conjunto de pontos é um problema bem resolvido, o que faz com que algumas das abordagens mais recentes da literatura não o explorem ou o tratem apenas como um pós-processamento do processo de segmentação de instâncias geométricas [24, 33]. Já outros, buscam utilizar os parâmetros de maneira implícita ou explícita para propiciar um melhor aprendizado durante o treinamento [63, 21]. Apesar de o principal desafio enfrentado pelo presente trabalho ser, de fato, a segmentação da nuvem de pontos em instâncias de primitivas geométricas, o processo de aprendizado pode ser facilitado ou potencializado pela integração com o ajuste paramétrico, visto que ambos problemas estabelecem uma interdependência causal, a qual pode ser melhor enfrentada durante o processo de treinamento das técnicas de aprendizado profundo supervisionadas.

Apesar de o problema de ajuste de primitivas geométricas em nuvens de pontos ser explorado na literatura, a escala dos dados de entrada é um fator determinante em grande parte das soluções, principalmente no sub-problema de segmentação de primitivas, visto que a quantidade de pontos tem alta influência tanto da escala da estrutura representada pela nuvem quanto com relação à resolução que o processo de captura proporciona. Além disso, os aspectos construtivos das estruturas, que podem ser vistos como a maneira na qual as geometrias estão distribuídas no espaço, tem alta dependência da escala e do contexto da estrutura, tendo forte influência na solução adotada. Apesar dos métodos de ajuste de primitivas geométricas a nuvens de pontos 3D serem de exímia importância para o processo de *As Built* automático e outras tarefas, poucos são os métodos que buscam solucionar o problema quando o contexto de aplicação são estruturas de larga escala. Como demonstrado por Kaiser et al. [26], quando se trata de nuvens de pontos de larga escala, os métodos geralmente focam somente no ajuste planar [38, 35, 39] ou não objetificam um ajuste de primitivas com a qualidade necessária para o *As Built*, como em [47], que utiliza as primitivas geométricas pra acelerar o processo de registro de nuvens de pontos. Contudo, principalmente no contexto das estruturas de pequena escala, além de algumas técnicas que buscam o ajuste de múltiplos tipos de primitivas a nuvens



(a) Nuvem de Pontos Segmentada por tipo de Geometria. A cor de cada ponto simboliza o tipo de primitiva geométrica do ponto. Os tipos possíveis são: Plano (●), Cilindro (●), Cone (●), Esfera (●), Toro (●) e B-Spline (●.



(b) Nuvem de Pontos Segmentada por instância de Geometria. A cor de cada ponto simboliza o índice da instância de primitiva geométrica do ponto. Pontos de mesma cor pertencem a mesma geometria.

Figura 4: Diferentes tipos de Segmentação em Geometrias.

de pontos [53, 32, 55, 63], também existem soluções mais completas que, no contexto de *As Built*, objetificam a recuperação de modelos CAD com base no ajuste de primitivas [6, 34, 21].

Além das soluções clássicas que normalmente eram adotadas para enfrentar esse tipo de problema, os últimos anos estão registrando um rápido desenvolvimento de uma definição denominada *Geometric Deep Learning* [9], que tenta unificar a definição de técnicas de aprendizado profundo independente do domínio de ação. Dentro desses possíveis domínios, as nuvens de pontos integram o grupo dos grafos, as quais cada vez mais são processadas por técnicas dessa abordagem, como largamente demonstrado por Ahmed et al. [1]. Dessa forma, é lógico imaginar que o ajuste de primitivas geométricas a nuvens de pontos de larga escala tenha suas próximas soluções dentro dessa mesma direção, assim como aconteceu no contextos de pequena escala, através dos métodos SPFN [32], ParSeNet [55], HPNet [63], PrimitiveNet [24], ComplexGen [21] e SED-Net [33]. Apesar do processo ser mais complexo em nuvens de pontos de larga escala, dada a dimensão dos dados de entrada, soluções baseadas em aprendizado já são utilizadas para resolver diversas tarefas em nuvens de larga escala, como é o caso de problemas como classificação, detecção e segmentação [30, 8, 23, 54]. Contudo, o método existente que mais se aproxima da solução do problema que se planeja resolver em contextos de larga escala é o PrimitiveNet [24], que realiza segmentação da nuvem de pontos em instâncias de primitivas geométricas, sem objetificar a tarefa de ajuste dos parâmetros dessas primitivas, diferente dos métodos de pequenas estruturas anteriormente citados.

1.4.2 *Datasets* de larga escala com anotação Geométrica

No contexto das técnicas de aprendizado profundo, de forma mais específica no aprendizado supervisionado, a presença de um *dataset* robusto, diverso e anotado, costuma ser o ponto de inflexão para muitos anos de desenvolvimento e comparação de métodos que buscam resolver as problemáticas apresentadas no conjunto de dados. Porém, em contextos de larga escala, para outros problemas como: detecção, segmentação de instância, segmentação semântica e segmentação panótica, uma diversidade de *datasets* de larga escala estão disponíveis, o que desafia os pesquisadores a desenvolver algoritmos capazes de resolver o problema e proporciona o avanço cientifico na área em questão. Grande parte desses *datasets* são coletados em ambientes reais e anotados manualmente como, por exemplo: SemanticKITTI [4], Semantic3D [22], S3DIS [3], entre outros. Contudo, abordagens de geração sintética de dados também são aplicadas para construir os *datasets*, como feito no STPLS3D [10], 3D-FRONT [15] e DM-SR [7].

No problema específico de ajuste de primitivas geométricas, a anotação manual dos dados é impraticável pois demanda além da segmentação em instâncias geométricas, a especificação com alto grau de precisão de cada um dos parâmetros das primitivas. No contexto de estruturas de pequena escala, o *dataset* ABC [27] estabeleceu um novo

padrão, disponibilizando um imenso conjunto de dados sintéticos, gerados a partir de desenhos CAD 3D, com o *ground truth*, ou seja, a anotação, para ajuste de primitivas geométricas. Porém, os autores desse *dataset*, até o presente momento, não disponibilizaram a ferramenta de geração dos dados que possibilitaria uma mais rápida transposição para os contextos de larga escala. Dessa forma, não há, que seja de conhecimento, *datasets* públicos de larga escala com nuvens de pontos que tenham suas geometrias anotadas, seja de maneira sintética ou real. Essa afirmação é reforçada por Huang et al. [24], que justifica a não utilização da PrimitiveNet para segmentação de múltiplos tipos de primitiva em larga escala, com a não existência de um *dataset* preparado para treinar e validar o modelo nesse contexto.

1.5 Objetivos

O objetivo do trabalho é desenvolver um método de ajuste de primitivas geométricas baseado em aprendizado profundo supervisionado que seja capaz de fornecer as informações necessárias para embasar uma técnica de *As Built* automático em estruturas de larga escala. Para que esse objetivo seja cumprido, alguns outros objetivos específicos devem ser alcançados anteriormente, são eles:

- Explorar os principais *datasets* que possam ser utilizados no problema;
- Explorar as técnicas do estado da arte em ajuste de primitivas geométricas a nuvens de pontos, sejam clássicas ou baseadas em aprendizado;
- Compreender o estado da arte em Aprendizado Profundo em Nuvens de Pontos;
- Desenvolver uma ferramenta de geração de *datasets* sintéticos que utilize modelos CAD de larga escala como entrada;
- Diminuir a lacuna entre os dados sintéticos e dados reais a partir da simulação da aquisição de nuvem de pontos por LiDAR;
- Construir um *dataset* sintético com nuvens de pontos de larga escala e primitivas geométricas no *ground truth*;
- Desenvolver uma proposta metodológica para o problema de ajuste de primitivas geométricas a nuvens de pontos de larga escala;
- Disponibilizar um *benchmark* comparando diversas soluções da literatura e diferentes características da metodologia proposta.

1.6 Contribuições

As principais contribuições do trabalho são:

- Ferramenta de geração de *datasets* sintéticos para aprendizado profundo geométrico.
- Método para geração de nuvens de pontos realistas em estruturas de larga-escala através da simulação de um drone com um LiDAR embarcado.
- ***Dataset* LS3DS:** um conjunto de dados de 86 modelos industriais de larga escala com nuvens de pontos realistas e *ground truth* para o problema de ajuste de primitivas geométricas.
- **Metodologia SMGF:** uma abordagem baseada em divisão e agregação para se adaptar diferentes técnicas de segmentação de geometrias da literatura para contextos de larga escala.
- ***Benchmark* de Ajuste de Primitivas Geométricas**: uma extensa discussão sobre os métodos adaptados com a metodologia SMGF junto com a comparação qualitativa e quantitativa dos mesmos.

1.7 Estrutura do Trabalho

O trabalho está dividido da seguinte maneira: o Capítulo 2 apresenta uma revisão bibliográfica sobre o problema em questão, o Capítulo 3 detalha a metodologia seguida para desenvolver a ferramenta de geração de *datasets* sintéticos, o Capítulo 4 descreve uma metodologia para desenvolver um modelo capaz de cumprir o objetivo geral do trabalho, o Capítulo 5 apresenta os resultados e o Capítulo 6 concluí o trabalho.

2 REVISÃO BIBLIOGRÁFICA

A discussão a respeito dos trabalhos relacionados busca cobrir todos os conceitos inter relacionados ao trabalho desenvolvido. Primeiramente, será feita uma discussão sobre os principais *datasets* disponíveis (Seção 2.1). Logo após, os principais avanços em aprendizado profundo em representações 3D serão comentados e relacionados ao presente trabalho (Seção 2.2). Finalizando, os métodos existentes de ajuste de primitivas geométricas serão descritos e discutidos (Seção 2.3).

2.1 *Datasets* para Aprendizado e *Benchmark* em Nuvem de Pontos

Em contextos de larga-escala, os *datasets* mais comuns são os de detecção, classificação e segmentação. Focando em segmentação, que é um sub-problema do ajuste primitivas geométricas, os que mais se destacam na literatura são:

- **SemanticKITTI [4]:** *dataset* real de múltiplas cenas capturadas por um LiDAR acoplado a um carro ao longo de uma trajetória percorrida em ambientes externos. Além do *ground-truth* para segmentação semântica e de instâncias em nuvens de pontos, o *dataset* também conta com anotação dos mesmos tópicos para imagens e de outras informações como odometria do carro durante a trajetória.

- **Semantic3D [22]:** *dataset* real capturado por escâner 3D em ambientes externos. Assim como o SemanticKITTI, cada nuvem de pontos representa apenas uma leitura completa do LiDAR, sem unir nuvens de pontos de múltiplas vistas do mesmo ambiente.

- **S3DIS [3]:** *dataset* real capturado por câmera RGB-D em ambientes internos. Diferentemente dos anteriores, cada cena é obtida a partir do registro das nuvens de pontos capturadas em cada uma das múltiplas vistas.

- **STPLS3D [10]:** *dataset* hibrido de nuvens de pontos reconstruídas usando fotogrametria aérea em imagens externas capturadas por *drones*. O *dataset* é hibrido porque os cenários são gerados sinteticamente através de blocos prontos das mais

diversas classes, como prédios, ruas e árvores. Logo após, esses blocos são dispostos no ambiente a partir de uma topologia extraída de imagens reais de satélite.

Além disso, focando no problema de ajuste de primitivas geométrica ainda em contextos de larga-escala, o *dataset* do método RAPter [38] tem nuvens de pontos reais para a entrada e as referências para ajuste geométricos de planos. Contudo, ao que sabemos, até o presente momento não há nenhum *dataset* com *ground truth* para ajuste de múltiplos tipos de primitivas geométricas à nuvens de pontos de larga-escala.

Desconsiderando a exigência de que os dados sejam de larga-escala, o ABC [27] é um *dataset* sintético para diversas tarefas de aprendizado profundo em representações 3D, como: estimativa de normal, reconstrução de superfícies e ajuste de primitivas geométricas ou superfícies paramétricas. A construção do *dataset* é feita por um *software* desenvolvido pelos autores que, a partir de um arquivo CAD de entrada, diversas informações são extraídas ou geradas para compor o *dataset*. Primeiramente, o *software* realiza a amostragem de uma malha triangular na superfície do CAD, utilizando a Gmsh API [17], a qual é gerada em formato OBJ e contém os vértices (posição, normal e curvatura) e os triângulos. Além disso, o *software* também gera, entre outras informações, um arquivo de *features* em formato YAML que lista todas curvas e superfícies presentes no arquivo de entrada, as quais são obtidas a partir da biblioteca *Open Cascade Technology* (OCCT).

Nesse contexto, o arquivo de *features* pode conter os seguintes tipos de curva: linha, círculo, elipse, *b-spline* e outros. Cada um deles, além de serem definidos a partir dos parâmetros individuais de cada tipo, também são definidos a partir do parâmetro que armazena os índices dos vértices da malha triangular que descreve a curva. Já com relação as superfícies, os tipos possíveis são: plano, cilindro, cone, esfera, toro, superfície de revolução, superfície de extrusão, *b-spline* e outros. Nos quais, além dos parâmetros individuais de cada tipo e os parâmetros de definição dos vértices, também há o parâmetro de definição das faces, que define qual o conjunto de triângulos que representa a superfície em questão na malha triangular.

Com base na malha triangular, é possível facilmente se obter uma nuvem de pontos utilizando uma abordagem de amostragem uniforme de pontos. Dessa forma, o *dataset* ABC pode gerar tanto as nuvens de pontos que vão servir como entrada para os modelos de aprendizado quanto o *ground truth*, que está disponível no arquivo de *features* YAML. Contudo, até o momento de escrita deste trabalho, o código que realiza a geração dos dados com base nos modelos CAD de entrada não está disponível. Dessa forma, não é possível utilizar diretamente o mesmo *software* do ABC para formar um *dataset* para larga-escala.

2.2 Aprendizado Profundo em Representações 3D

Ahmed et al. [1] faz um levantamento dos avanços no uso de aprendizado profundo em diferentes representações de dados tridimensionais. Para isso, cada representação pode ser classificada em:

- **Euclidiana:** as representações desse tipo tem uma estrutura euclidiana implícita, na qual as propriedades de uma topologia em grade são mantidas, como uma parametrização global e um sistema de coordenadas comum para os dados. Os principais exemplos são: descritores, projeções, imagens RGB-D, dados volumétricos e dados de múltiplas vistas. [1]

- **Não-Euclidiana:** as representações desse tipo pertencem ao espaço dos vetores, não seguem uma parametrização global ou um único e comum sistema de coordenadas. Essas características tornam a extensão de técnicas de aprendizado profundo 2D um grande desafio. Os principais exemplos são: nuvens de pontos, malhas e grafos. [1]

Comparando as nuvens de pontos com as imagens, sobre as quais há muito conhecimento agregado em aprendizado profundo, as nuvens não tem informação de conectividade local ou ordem, o que torna impraticável a aplicação direta dos muitos modelos de redes neurais convolucionais (CNNs) desenvolvidos ao longo dos anos de pesquisa em aprendizado profundo em imagens. Como demonstrado por Bronstein et al. [9], as nuvens de pontos estão no mesmo domínio dos grafos, sendo elas um caso especial de um grafo totalmente desconexo, ou seja, sem nenhuma aresta, apenas nodos. Dessa forma, até mesmo as redes neurais para grafos (GNNs), que são alvo de muitas pesquisas e atingem ótimos resultados no processamento dessa estrutura de dados, conseguem lidar diretamente com a falta de informações de conectividade das nuvens de pontos. As abordagens mais comuns para aprender usando nuvens de pontos são a projeção da nuvem em imagem ou a voxelização. Nos dois casos, a abordagem tem o mesmo fundamento: transformar a nuvem de pontos em uma representação euclidiana e processar com redes neurais do estado da arte em cada uma dessas representações.

A transformação da nuvem de pontos em imagem é uma abordagem muito utilizada em alguns contextos, principalmente em *datasets* onde a nuvem de pontos é coletada a partir de uma única leitura de um LiDAR [22][4]. Geralmente, o processo de captura dos dados faz os pontos sofrem uma projeção, que normalmente é cilíndrica ou esférica, para gerar uma imagem que represente a nuvem de pontos. Contudo, ao realizar essa operação, há uma alta taxa de oclusão e problema de escala, visto que objetos de mesmo tamanho na nuvem original podem ser representados com tamanhos diferentes na imagem devido à distância ao centro de projeção. Contudo, nos casos em que o modelo e parâmetros do LiDAR são conhecidos, uma abordagem com menos perdas e ambiguidades pode ser

utilizada, a qual é denominada *scan unfolding*, utilizada em [56]. Através desse processo, as imagens originais capturadas pelo sensor, as quais ele utilizou para gerar a nuvem de pontos, como a imagem de profundidade, a de intensidade de retorno do feixe de luz e as imagens de coordenadas x, y e z, podem ser recuperadas e utilizadas nos modelos de aprendizado.

Na abordagem de voxelização, a nuvem de pontos é transformada em um *Voxel Grid*, que é uma representação em grade de três dimensões, o qual é processado a partir de redes neurais convolucionais que são transpostas para o mundo 3D. Contudo, por ter uma dimensão a mais, a resolução do *Voxel Grid* tem que ser muito reduzida para se ter um tempo de processamento que habilite o processo de treino em tempo hábil e mais eficiente em memória, o que causa uma perda significativa de informação. Contudo, a utilização de convoluções esparsas [19][36][20] possibilitou a otimização significativa das convoluções em *Voxel Grids*, dada a esparsidade natural dessa representação 3D. Esse avanço foi alavancado principalmente pelo desenvolvimento de uma biblioteca para desenvolvimento de redes neurais para tensores esparsos chamada *Minkoski Engine* [11].

Apesar das abordagens indiretas ainda serem muito utilizadas, a PointNet [44] e sua sucessora PointNet++ [45] foram uma das primeiras redes neurais a receber como entrada diretamente as nuvens de pontos. A PointNet++ faz um processo de hierarquização, utilizando cada vez menos pontos nas camadas mais profundas da rede, o que auxilia substancialmente na redução do número de parâmetros da rede e, consequentemente, na redução do custo computacional e consumo de memória.

Trazendo uma visão geral das diferentes abordagens utilizada para se aprender em dados tridimensionais, é possível concluir que o seguimento a ser escolhido é muito dependente da forma de coleta do *dataset*, da precisão exigida dos resultados e do problema que se está tentando resolver. Com base no que já foi comentado, a projeção em imagem 2D é facilitada pelos *datasets* em que cada nuvem de pontos é gerada com apenas uma leitura do sensor, o que não é geralmente o caso de nuvens de pontos de larga escala. A análise das outras abordagens depende fundamentalmente da técnica a ser utilizada, visto que ambas parecem possíveis de ser usadas no contexto de aplicação do trabalho. Por um lado, a voxelização transforma os dados em formato mais estruturado e fácil de ser processado, contudo, esse processo deve ser ajustado de maneira fina pra que se escolha uma resolução adequada para os *voxels*, visando ao mesmo tempo, não somente precisão na segmentação e parametrização das primitivas geométricas, como também custo computacional e consumo de memória controlados.

Existem ainda outros modelos mais novos buscam focar especificamente em nuvens de pontos de larga-escala, como por exemplo:

- **SPGraph [30]:** o método realiza um longo pré-processamento para dividir a nuvem de entrada em formas geométricas simples, que são chamadas de super pontos e, logo após, inicia seu processo supervisionado através da criação de um grafo de

super pontos onde as *features* armazenadas nas arestas são calculadas através de uma PointNet e finaliza através de uma segmentação semântica com contexto ao longo dos super pontos e das arestas utilizando convoluções em grafos.

- **RandLA-Net [23]:** ao contrário do SPGraph, o método não faz nenhum tipo de pré-processamento, ou seja, a nuvem de pontos entra diretamente na rede neural, o que acaba por gerar uma limitação, visto que a rede suporta no máximo cerca de 1 milhão de pontos na entrada. Com foco total na eficiência, a abordagem da RandLA-Net é muito parecida com a da PointNet++, contudo, em vez de realizar a redução progressiva do número de pontos através da hierarquização, uma abordagem de *downsampling* aleatória é utilizada.

- **ConvPoint [8]:** o método realiza um pré-processamento mais simples que o SPGraph, visto que, a cada iteração de treino um ponto é sorteado e uma região de tamanho fixo em volta dele é utilizada para selecionar os pontos que entrarão para a rede. Nessa região, 8192 pontos são amostrados aleatoriamente para formar a entrada da rede. Já no processo de teste, as regiões de amostragem aleatória são selecionadas de maneira que toda a nuvem seja processada. Após o pré-processamento, a ConvPoint utiliza uma generalização da equação de convoluções em imagens para um ambiente esparso e sem as relações de vizinhança de uma imagem. Dessa forma, as posições de cada ponto não são utilizadas como *feature*, e sim como um índice contínuo para a aplicação das convoluções. Assim, uma CNN *encoder-decoder* pode ser generalizada e utilizada para realizar a segmentação da nuvem de pontos.

Como comentado, os métodos focados na larga-escala também não conseguem lidar diretamente com nuvens de pontos muito grandes na entrada, onde alguma abordagem de pré-segmentação antes do processamento ou hierarquização ao longo das camadas da rede sempre é adotada. Contudo, aquelas que fazem hierarquização costumam, ainda assim, ter uma limitação considerável na quantidade de pontos na entrada [23] [45]. Logo, nos contextos onde baixar muito a quantidade de pontos acaba por gerar uma perda expressiva de detalhes e de informação na nuvem, as abordagens baseadas em pré-segmentação são mais indicadas, mesmo no problema de segmentação de instância em nuvens de pontos, já que abordagens do estado-da-arte como Mask3D [54] e ISBNet[59] também realizam esse pré-processamento. Todavia, após obter os resultados em cada um dos segmentos, deve haver um processo de agregação dos resultados para se obter a solução para toda estrutura representada na nuvem de pontos. Dessa forma, aplicar os conceitos da literatura de outros problemas que também ocorrem em contextos de larga escala, como divisão e agregação, pode ser a chave para se obter um método capaz de resolver o problema do trabalho.

2.3 Ajuste de Primitivas Geométricas em Representações 3D

Apesar do trabalho ser baseado em aprendizado profundo, as metodologias clássicas que buscam resolver o problema também serão detalhadas, as quais são discutidas na Seção. 2.3.1. Logo após, na Seção 2.3.2, os métodos baseados em aprendizado serão apresentados, explicados e co-relacionados para que se possa compreender quais são os principais pontos positivos e as limitações dos métodos existentes na literatura.

2.3.1 Métodos Clássicos

Kaiser et al. [26] avaliam os principais métodos de ajuste de primitivas geométricas que foram publicados até o ano de 2016, nessa revisão os métodos são tabelados, principalmente, com base em: tipos de primitivas que ajustam, contexto dos dados, tipo de dado de entrada e a fundamentação teórica que o método se baseia. Dessa forma, 72 métodos são analisados e, com base em seu processo de classificação, é possível compreender a força da relação entre cada um e a problemática do presente neste trabalho. Assim, analisando com relação aos tipos de primitivas, o foco de observação é naquelas técnicas que não buscam apenas o ajuste planar, visto que essas geralmente se baseiam em premissas que só podem ser utilizadas em planos. Já com relação ao contexto de aplicação, tanto as técnicas voltadas para cenas internas, que geralmente são estruturas maiores, quanto aquelas focadas em objetos individuais podem ser fortemente relacionadas, visto que mesmo as técnicas voltadas para estruturas pequenas tem a possibilidade de obter bons resultados no ajuste para nuvens de pontos de grandes estruturas. Seguindo, com relação aos tipos de dado de entrada, o foco é naqueles métodos que recebem nuvens de pontos, sejam essas com a informação do vetor normal ou não. E finalizando, nenhuma fundamentação teórica específica faz parte dos objetivos, logo, os métodos não são selecionados com base nesse critério. Assim, com base nas premissas mencionadas, o ERANSAC [53] foi o método de maior importância mostrado na literatura de métodos publicados até o ano de 2016.

ERANSAC [53]. O ERANSAC pode ser considerado como o estado da arte clássico, ou seja, não baseado em aprendizado, em ajuste de primitivas geométricas de múltiplos tipos. Apesar de ser classificado em Kaiser et al. [26] como um método focado apenas na pequena-escala, essa informação não é especificada no artigo original, no qual inclusive, alguns resultados são demonstrados para o ajuste em nuvens com uma quantidade alta de pontos. Contudo, essas não necessariamente representam estruturas de larga-escala, porém, pelo menos o método mostra a capacidade de trabalhar com dados de alta dimensionalidade. Para resolver o problema, o fundamento teórico do método é uma estratégia estocástica clássica nomeada *Random Sample Consensus (RANSAC)* [14], na qual amos-

tras aleatórias de pontos são geradas iterativamente, bem como, primitivas são ajustadas a elas, e consequentemente, é feita uma avaliação de como o resto dos dados se relacionam com a primitiva estimada. Dessa forma, primitivas aceitas pela maior parte dos pontos restantes, que estão fora da amostra, são aceitas e as demais são recusadas.

Através do paradigma de *Random Sample Consensus*, o ERANSAC busca ajustar cinco tipos de primitivas, são elas: plano, esfera, cilindro, cone e toro. Para isso, os dados de entrada são tanto as posições dos pontos, como também os vetores normais da superfície ao redor de cada ponto, o que acaba por diminuir o tamanho mínimo das amostras, visto que quantidade de dados por ponto é dobrada. Assim, a cada iteração, um conjunto de amostras é gerado a partir de uma estratégia de amostragem localizada baseada em *Octree*, na qual, consequentemente, cada primitiva estimada é avaliada e apenas a melhor fica, caso seja boa o suficiente. Finalizando, o algoritmo tem bases probabilísticas fortes na amostragem, na análise da qualidade das primitivas e na condição de parada do método. Com base no que foi apresentado, é importante pontuar que é um método de essência aleatória e com uma série de parâmetros a serem ajustados, o qual, apesar de ser um ótimo método, deixa uma lacuna para pesquisas de novas técnicas de ajuste de primitivas geométricas devido a sua falta de determinismo e sua necessidade de parametrização manual.

2.3.2 Métodos Baseados em Aprendizado

Em contraponto aos métodos com fundamentação teórica clássica, a revolução ocasionada pelos métodos aprendizado profundo também se mostra muito presente no tratamento de dados 3D. Ahmed et al. [1] faz uma revisão sobre essas diferentes técnicas, que possibilitaram o grande avanço na direção da resolução de diversos problemas do processamento de dados tridimensionais utilizando aprendizado profundo. Baseado nessas técnicas, com a intenção de criar novos métodos de ajuste de primitivas geométricas que superem o ERANSAC, algumas técnicas baseadas em aprendizado foram propostas.

SPFN [31]. O SPFN foi o primeiro método baseado em aprendizado profundo, desenvolvido com a intenção de resolver o problema, que teve resultados competitivos. Para isso, se criou um método modular, onde cada módulo desempenha uma função específica e executa operações diferenciáveis, assim, a junção de todos forma um modelo que pode ser treinado fim-a-fim. Durante o treinamento, um número máximo de primitivas por modelo deve ser escolhido, sendo esse um valor que impacta diretamente no tamanho do modelo e no tempo de treino. Além disso, o grupo de tipos de primitivas a serem ajustadas também deve ser selecionado, onde os quatro tipos possíveis, que representam, segundo Li et al. [32], 94% das primitivas que fazem parte de desenhos 3D no *dataset* utilizado, são: planos, esferas, cilindros e cones. Seu princípio de funcionamento é totalmente

dependente de um modelo de rede neural profunda chamado PointNet++ [45], que tem como objetivo o cálculo de características de relevância global para cada um dos pontos. Nesse contexto, três camadas totalmente conectadas, separadas entre si, são adicionadas ao final dessa rede com o objetivo de gerar três tipos diferentes de características para o resto do modelo, são elas:

- **Associações:** essas características fazem uma relação de pertencimento entre os pontos e as primitivas detectadas. Essas relações são representadas por uma matriz $\widehat{W} \in [0,1]^{N \times K}$, onde N é o número de pontos e K é o número máximo de primitivas por modelo;

- **Normais:** essas características representam a normal da superfície ao redor dos pontos. As normais são representadas por um vetor $\widehat{N} \in \mathbb{R}^{N \times 3}$, onde N é o número de pontos e 3 é o número de coordenadas do vetor normal, representado por $\overrightarrow{N} = (N_x, N_y, N_z)$;

- **Tipos:** essas características fazem a relação do tipo de primitiva de cada ponto. Essas relações são representadas por uma matriz $\widehat{T} \in [0,1]^{N \times L}$, onde N é o número de pontos na entrada, e L é o número de tipos de primitiva, que tem como valor máximo 4.

A partir das características calculadas, o SPFN ajusta cada tipo de primitiva a partir da resolução de diferentes problemas de mínimos quadrados. Finalmente, as funções de perda são computadas, fechando a malha de supervisão, e retro-propagando o erro para ajustar os pesos das PointNet++ e da camada totalmente conectada de cada característica. No artigo, SPFN é constantemente comparado ao ERANSAC, onde os resultados são consideravelmente melhores, mesmo sem receber os vetores normais na entrada, nos dados utilizados pelo método. Porém, é importante pontuar que o SPFN foi treinado especificamente para objetos individuais, os quais tinham nuvens na ordem de dez mil pontos, o que o torna mais limitado, porém, superior ao ERANSAC nesse contexto especifico.

ParseNet [55]. Seguindo o mesmo fundamento teórico do SPFN, o ParseNet busca o ajuste de superfícies paramétricas a partir de um modelo de aprendizado profundo. Ideologicamente, o método busca gerar um resultado que respeite os três princípios básicos dos modelos CAD, são eles: continuidade, editabilidade e flexibilidade. Para isso, foi criado um modelo também modular e totalmente diferenciável para que possa ser treinado fim-a-fim. Como os métodos anteriormente apresentados, algumas superfícies paramétricas são escolhidas como objetivos de ajuste do método, são elas: planos, esferas, cilindros, cones e curvas *b-spline* (definidas por Piegl and Tiller [42]). Nesse caso, é importante perceber que o termo superfícies paramétricas, ao contrário de primitivas geométricas como falado

anteriormente, se faz necessário, visto que o método é capaz de ajustar curvas *b-spline*, que não são primitivas geométricas. Logo, para fazer o ajuste desses tipos de superfície, o primeiro módulo tem o objetivo de segmentar semanticamente a nuvem de entrada, o qual é chamado de módulo de decomposição, e ele realiza três processos, são eles: rede de *embedding*, agrupamento e classificação do segmento.

- **Rede de *Embedding*:** são utilizadas três camadas *EdgeConv* da DGCNN [61] seguidas de três camadas totalmente conectadas para gerar, para cada ponto, através da análise de 256 pontos a sua volta, uma representação em um espaço de 128 dimensões;

- **Agrupamento:** uma versão diferenciável do algoritmo *mean-shift* é utilizada com o objetivo de encontrar, no espaço de 128 dimensões, agrupamentos de pontos que configurem uma possível primitiva;

- **Classificação de Segmento:** os mesmos dados de saída das três camada de *EdgeConv* da rede de *embedding* são utilizados, os quais são passados por camadas totalmente conectadas e ReLU's, seguindo, finalmente, para uma camada de *softmax*, que gera a probabilidade de cada ponto ser de cada um dos tipos de superfície paramétrica falados anteriormente. Logo após, o tipo de primitiva dos agrupamentos do processo anterior é obtido através de um esquema de votação de seus pontos.

Ainda no ParseNet, depois de feita a decomposição, os grupos pertencentes a alguma das primitivas geométricas (plano, esfera, cilindro e cone) são ajustados da mesma maneira que no SPFN, através de múltiplas soluções de problemas de mínimos quadrados diferentes. Já aqueles que são classificados como uma curva *b-spline*, passam por uma SplineNet [50] e, logo após, são finalizados por um módulo de pós-processamento. Finalmente, a parte de supervisão calcula as funções de erro e retro-propaga essa informação para alterar os pesos das camadas de rede neural de todo processo. Depois de treinado, o método pode ser executado e testado frente a algumas métricas de avaliação, assim, o ParseNet é comparado ao SPFN e ao ERANSAC, onde tem resultados superiores aos dois. Além disso, é importante pontuar que todos os métodos são testados com e sem a entrada das normais dos pontos, visto que o SPFN não recebe esse dado como entrada. Porém, mais uma vez, a comparação é feita frente a um conjunto de dados específico voltado para nuvens de pontos de pequenas estruturas, o que torna esse modelo treinado do ParseNet superior nesse contexto, mas não de forma geral. Somado a isso, o fato de a rede de *embedding* buscar uma representação de 128 dimensões para cada ponto de entrada torna o processo pouco escalável, visto que com o aumento do número de pontos, a memória necessária para armazenar os dados desse espaço multi-dimensional pode acabar sendo impraticável.

Apesar do fato de que, no contexto de objetos individuais, o ParseNet é o método com os melhores resultados, isso não necessariamente é verdade no contexto de grandes

estruturas. Assim, as diferenças na estratégia adotada pelo SPFN, mesmo que tenha a mesma fundamentação teórica, ainda podem promover a obtenção de resultados superiores aos da ParseNet. Junto a isso, o ERANSAC, apesar de mais antigo, não tem um contexto de aplicação especificado e pode ter resultados ainda superiores a essas novas técnicas baseadas em aprendizado profundo. Contudo, essa comparação seria mais justa, caso existissem dados de grandes estruturas com *ground truth* para treinar os modelos baseados em aprendizado e comparar os métodos utilizando métricas mais confiáveis.

HPNet [63]. Recentemente outras soluções baseadas em aprendizado profundo surgiram buscando resolver o problema. A primeira delas é a HPNet [63], a qual também é para pequena-escala e é muito embasada na ParseNet, tendo limitações de consumo de memória e número de pontos muito parecidas. A grande contribuição desse método é aprender também os parâmetros das geometrias dentro do modelo da rede neural. Entretendo, esses parâmetros são aprendidos para cada ponto, e não para cada instância geométrica. Dessa forma, esses parâmetros são utilizados para garantir consistência entre os diferentes pontos de uma primitiva geométrica, verificando, para cada ponto, se seus vizinhos condizem com a geometria estimada por ele. Apesar disso, a HPNet é classificada como um método de segmentação de primitivas, e não de ajuste das mesmas, visto que esses parâmetros estimados são apenas utilizados para formar a representação de *embedding* a ser utilizada pelo método. Além dessa representação, o módulo de *spectal embedding* da HPNet conta com dois outros vetores de características, sendo eles um que garante suavidade, o qual é baseado nas normais dos pontos, e outro que é similar ao já adotado anteriormente pela ParseNet. Dessa forma, essas três representações são agregadas de forma ponderada para gerar uma representação final de 128 dimensões, a qual é utilizada como entrada para um algoritmo de *mean-shift* que agrupa esse os vetores desse espaço 128-D em instâncias de primitivas geométricas.

PrimitiveNet [55]. A PrimitiveNet é um método de segmentação de instâncias geométricas, assim como a HPNet, por outro lado, é a primeira abordagem baseada em aprendizado que busca encontrar geometrias em nuvens de pontos de larga escala. Apesar das limitações de consumo de memória dos métodos baseados em aprendizado já comentadas, a PrimitiveNet implementa uma abordagem localizada, na qual, em vez de todos os pontos entrarem de uma só vez na rede neural, eles entram de maneira fracionada. Para isso, cada ponto entra na rede acompanhado apenas de seus vizinhos mais próximos computados a partir de um algoritmo de K-Vizinhos mais Próximos (KNN) com $K = 16$, estratégia que já adotada previamente na literatura por Ben-Shabat and Gould [5]. Entretanto, como a entrada é fracionada, os resultados também são, logo, a PrimitiveNet propõe um pós-processamento baseado em crescimento de regiões que expande os resultados de

cada uma das porções para a nuvem completa. A partir dessa abordagem, experimentos foram realizados pelos autores utilizando ate 250.000 pontos por modelo do *dataset* utilizado, sendo bem superior, nesse aspecto, aos seus antecessores.

Além dos dados de entrada já comentados nos outros métodos, que são os pontos e as normais, a PrimitiveNet também recebe um conjunto de arestas na entrada, o qual conecta cada um dos pontos aos seus vizinhos. Nos experimentos feitos pelos autores, eles utilizam malhas triangulares em vezes de nuvens de pontos na entrada, dessa forma, os vértices da malha são os pontos, as normais dos vértices são as normais e as arestas dos triângulos são o conjunto de arestas. Contudo, para se utilizar somente nuvens de pontos, Huang et al. [24] sugere o uso do mesmo KNN da entrada como fonte de geração das arestas, onde cada ponto tem conexão com seus 16 vizinhos mais próximos. Ao contrário dos métodos anteriores, como a ParseNet e a HPNet, que calculam uma representação em *embbeding* com muitas dimensões e aplicam um *mean-shift* na mesma, a PrimitiveNet define o problema segmentação em instâncias de outra maneira. Assim, em vez agrupar com um algoritmo clássico, esse método busca classificar cada aresta do conjunto de arestas como sendo uma borda ou uma não-borda. Logo, o algoritmo de crescimento de regiões que agrega os resultados só se desloca sobre arestas de não-borda, ou seja, que não marcam a divisão entre geometrias diferentes. Ao fim do processo, o pós-processamento é capaz de gerar uma nuvem de pontos segmentada por instâncias de primitivas geométricas.

A arquitetura de rede neural desse método primeiro realiza o *Embedding* de Primitivas, com o objetivo de gerar as representações de *embedding* para os pontos. Assim, o conjunto local de pontos passa por duas camadas de PointNet [43] as quais geram uma representação F_1 de dimensão $[N \times 16]$ e, logo após, F_1 é passa por uma SparseConvNet[20], gerando F_2, também de dimensão $[N \times 16]$. Ao fim, as duas são concatenadas e o *embedding* final é passado por uma MLP que gera vetores de características a serem utilizados para computar o tipo de primitiva de cada ponto e para passar pelo módulo Discriminador de Primitivas, o qual é capaz de dizer, para cada aresta, se essas são arestas de borda ou não-borda.

Apesar de ter sido desenvolvida para nuvens de pontos e para contextos de larga escala, a PrimitiveNet não foi validada na segmentação de múltiplos tipos de geometrias em nuvens de pontos de larga escala devido a não existência de um *dataset* para tal.

Embora outros métodos da literatura tenham soluções orientadas ao problema que cheguem até mais perto da solução completa do processo de reconstrução de CAD, como a ComplexGen [21], seu processo de treinamento é extremamente pesado, mesmo utilizando apenas 10.000 pontos na entrada, o que impossibilita ainda mais a adaptação para contextos de larga escala. Por outro lado, a SED-Net [33], apesar de apresentar uma

solução mais escalável que a ComplexGen, o código desenvolvido pelos autores ainda não está disponível para uso.

Com base no que foi discutido, os métodos foram, de maneira geral, foram treinados e validados em contextos de pequena escala devido ao fato de que há *datasets* para tal nos mesmos. Porém, com o *dataset* a ser proposto no presente trabalho, se abrirá espaço para aplicar conhecimentos de técnicas aplicadas a outros problemas de larga escala, discutidas na Seção 2.2, em conjunto com as ideias utilizadas pelos métodos aqui detalhados, para se construir uma solução para o problema do trabalho.

3 GERAÇÃO DE *DATASETS* SINTÉTICOS PARA APRENDIZADO GEOMÉTRICO

Como discutido na Seção 2.1, grande parte dos *datasets* utilizados para aprendizado profundo em nuvens de pontos são formados por dados reais capturados em contextos específicos e anotados manualmente em *softwares* de anotação de nuvens de pontos. Apesar dos dados manualmente anotados serem muito comuns em *datasets* de segmentação, detecção e classificação, essa abordagem de anotação não pode ser facilmente transposta para o problema do ajuste de primitivas geométricas. Junto com o aumento da complexidade da anotação que a tridimensionalidade dos dados acrescenta, o que é comum a todos *datasets* de representações 3D, a tarefa de anotar manualmente parâmetros, como posições, eixos de revolução e comprimentos, para diversos tipos de geometria de múltiplas escalas não é uma tarefa simples para anotadores, mesmo que sejam especializados nisso. Anteriormente aos parâmetros, a própria habilidade de reconhecer que um conjunto de pontos configura uma superfície que pode ser descrita pela equação de um tipo de primitiva geométrica é uma tarefa que foge da capacidade de um profissional especialista em anotação de dados.

Voltando a atenção novamente para a motivação do trabalho, o processo de *As Built* automático aqui explorado visa sair de uma representação 3D não estruturada topologicamente, como as nuvens de pontos, para alcançar uma representação estruturada e com parametrização detalhada das geometrias existentes, como os formatos de CAD 3D, mais especificamente em representação B-Rep. Dessa forma, assumindo que se quer transformar nuvens de pontos em CADs, é possível guiar o processo de geração sintética desses dados avançando no sentido contrário, partindo de arquivos CAD para obter diretamente a lista de geometrias de sua topologia, bem como seus respectivos tipos e parâmetros. Contudo, a presença dos valores de referência não é o bastante para ter um *dataset*, visto que os arquivos de entrada também são necessários, nesse caso, as nuvens de pontos. O processo de geração sintética de pontos a partir de um modelo CAD será discutido na Seção 3.3. Contudo, para que os pontos possam ser produzidos na superfície dos elementos que compõe o modelo CAD, uma abordagem intermediária costuma ser necessária. Técnicas como amostragem de pontos, *ray casting* (ou lançamento de raio) e

simulação em *softwares* como Gazebo, Unreal e Unity demandam uma malha triangular como entrada. Embora essa representação também seja chamada de modelo CAD em alguns contextos, as malhas triangulares não contém as geometrias originais que compõe, por exemplo, um modelo B-Rep, ao contrário, elas são representações de superfície que descrevem todos os elementos a partir de uma série de triângulos interconectados. Dessa forma, anterior a geração de nuvem de pontos, um processo de transformação de CAD em formato STEP para malhas triangulares se faz necessário.

Com base no que foi discutido anteriormente, o processo de geração de dados geométricos de forma sintética deve cumprir os seguintes requisitos:

1. Explorar as geometrias contidas na topologia do arquivo CAD em formato B-Rep;
2. Anotar, para cada geometria, o tipo e os parâmetros associados;
3. Gerar uma malha triangular sob a superfície do modelo B-Rep;
4. Relacionar vértices e triângulos (ou faces) da malha triangular com cada geometria anotada;
5. Gerar uma nuvem de pontos em cima da malha triangular produzida;
6. Associar pontos com as geometrias anotadas com base na associação da malha com as mesmas.

Dessa forma, um *Software* em linha de comando foi desenvolvido para cumprir com os requisitos estabelecidos.

O processo de transformação e as conexões entre os formatos de dados que fluem pelo *Software* estão demonstrados na Figura 5. No diagrama é possível perceber que, como estabelecido no Requisito 1, a entrada do processo é um modelo CAD representado em padrão B-Rep. A partir da exploração topológica desse modelo, as geometrias são armazenadas em objetos internos do software para posterior serialização em YAML, JSON ou PKL, processo detalhado na Seção 3.1 e que cumpre com o Requisito 2. Paralelamente, o modelo de entrada passa por um algoritmo de triangulação para que uma malha triangular possa ser gerada como representação 3D parcial para geração dos pontos. Ao passo que o modelo é triangulado, os índices dos vértices e triângulos da malha que representam cada primitiva geométrica vão sendo anotados nos objetos que armazenam as geometrias para serem serializados conjuntamente com os outros parâmetros. A triangulação e armazenamento dos índices cumprem com os requisitos 3 e 4, respectivamente, ambos são detalhados na Seção 3.2. Ao fim, todas as geometrias e a malha são normalizadas para que a unidade de medida seja metros e para que o eixo z seja o eixo que aponte no sentido positivo da direção vertical do sistema de coordenadas. Logo após, a malha triangular é utilizada para se gerar pontos que sirvam como dado de entrada para

os modelos de aprendizado, processo que é detalhado na Seção 3.3. Ao fim, a Seção 3.4 detalha e justifica alguns métodos de pós-processamentos implementados que são úteis para a problemática do trabalho.

3.1 Exploração Topológica e Anotação de Geometrias

A partir de um modelo CAD em padrão B-Rep, a exploração topológica visa percorrer sua estrutura em árvore, anteriormente apresentada na Figura 2, coletando todas as suas curvas e superfícies geométricas em conjunto com o tipo e parâmetros associados. Para abrir o arquivo e realizar a exploração topológica, a *Open Cascade Technology* (OCCT) [1] foi utilizada, a partir de sua interface para a linguagem de programação *Python*, denominada PythonOCC[2]. Dessa forma, a interface de carregamento dos arquivos CAD pode ser utilizada e algumas funções de exploração topológica também. No presente trabalho, duas explorações topológicas foram propostas, utilizando a função principal de exploração topológica da biblioteca, são elas:

- **Exploração a partir da Dimensão mais Alta:** as únicas Faces a serem consideradas são aquelas que estão conectadas a alguma Casca Fechada. A mesma regra se aplica para as Arestas com relação as Faces. Dessa forma, a exploração parte das Cascas Fechadas presentes no CAD para alcançar todas Faces e Arestas topologicamente relacionadas;

- **Exploração Completa:** todas as Arestas e Faces, independente das conexões na topologia, são utilizadas. Nessa abordagem, uma superfície não precisa estar conectada necessariamente a um volume para ser considerada. Nem uma curva necessita da conexão com uma superfície para ser alcançada.

No início do trabalho, a principal função de exploração topológica da biblioteca começou a ser usada para realizar os dois tipos de exploração, onde se percebeu um problema grave de performance na versão mais atual disponibilizada até aquele momento, a v7.5.1, visto que, a cada nova entidade em exploração, a mesma tinha de ser comparada a todas as outras já exploradas para garantir a não repetição. Este processo é necessário porque, assim como uma Aresta (entidade de 1 dimensão) pode contornar múltiplas Faces (entidade de 2 dimensões), estas, por sua vez, também pode contornar múltiplas Cascas Fechadas (entidade de 3 dimensões). Dessa forma, o método de exploração topológica implementado na biblioteca pode ser aproximado a uma complexidade computacional de $O(n^2)$, onde n é o número total de entidades. Visando aumentar a eficiência da exploração, uma solução alternativa foi proposta para otimizar a exploração topológica,

[1] Site Oficial: `https://dev.opencascade.org`
[2] Repositório no Github: `https://github.com/tpaviot/pythonocc`

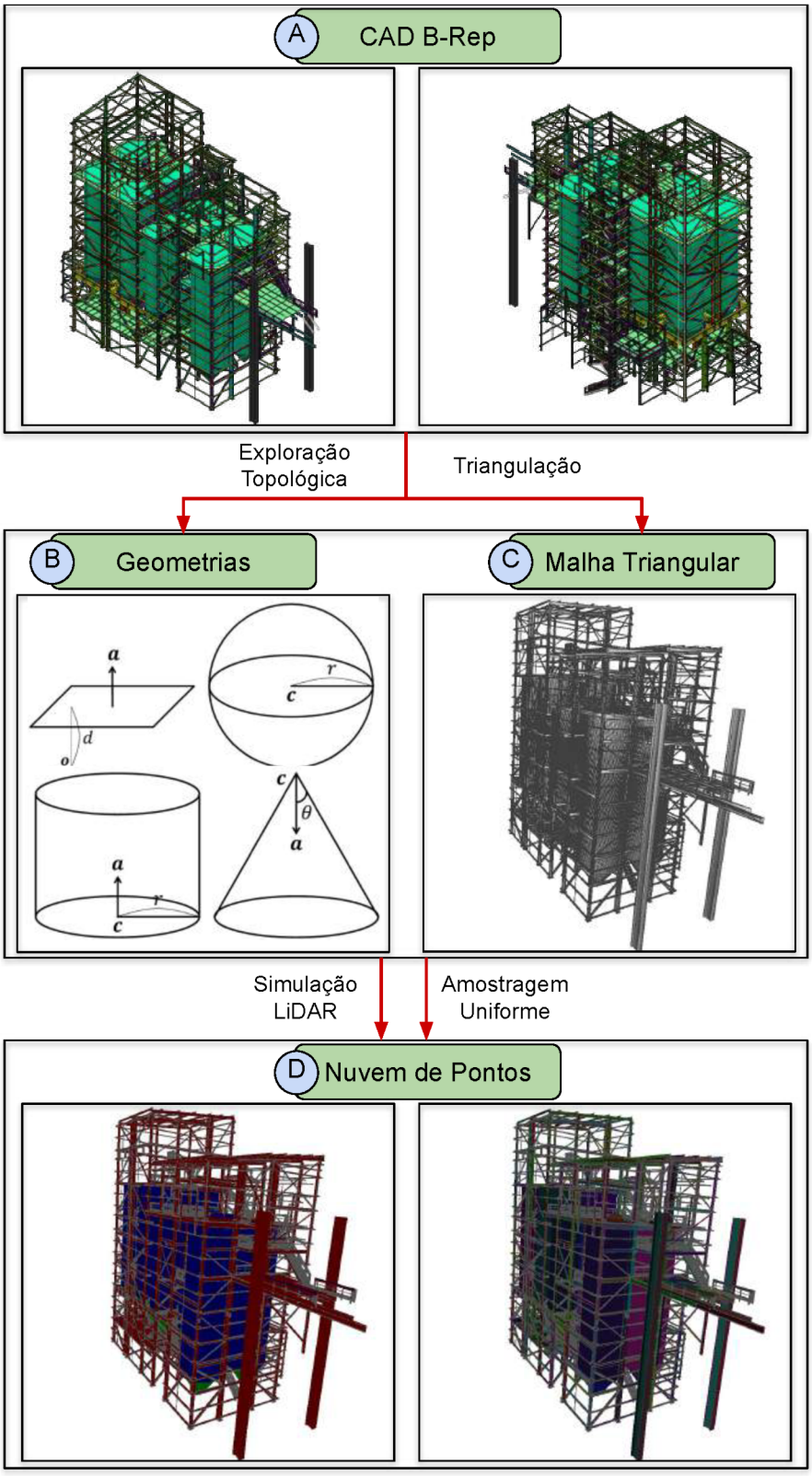


Figura 5: Diagrama dos diferentes formatos que fluem no *Software* de geração de dados. Em **A**, se tem um CAD B-Rep que serve como entrada para o sistema. Em **B**, se tem a lista de geometrias obtidas a partir da exploração topológica do arquivo CAD, as quais são armazenadas em uma estrutura chave-valor e serializadas em um arquivo. Em **C**, um processo de triangulação gera uma malha triangular como representação 3D parcial. Ao fim, em **D** as nuvens de ponto com *ground truth* são geradas utilizando duas abordagens possíveis, que são Simulação de LiDAR e Amostragem Uniforme.

utilizando o cálculo de um código *hash* para cada entidade e realizando a busca em complexidade $O(1)$ através de uma tabela *hash*. O cálculo de código *hash* é nativo da OCCT e leva em consideração os parâmetros e tipo das entidades para ser computado. Contudo, esse cálculo não garante a propriedade de unicidade, ou seja, entidades diferentes podem acabar tendo o mesmo código *hash*. Dessa forma, como apenas a busca na tabela *hash* não resolve o problema por completo, uma abordagem de Encadeamento Separado foi adotada, relacionando uma lista de entidades a cada código *hash* na tabela. Logo, sempre que um elemento recebe um código *hash* já presente na tabela, este elemento comparado com todos elementos da lista armazenada para confirmar se o processo de exploração topológica já passou por ele anteriormente ou não. A abordagem de busca de elementos geométricas proposta está demonstrada em pseudocódigo no Algoritmo 1, que pode ser usada para inserir novos elementos sem repetição através do procedimento demonstrado no Algoritmo 2.

Algoritmo 1 Busca usando de Elementos Geométricas usando Encadeamento Separado

```
1:  Função BUSCA(tabela, elemento)          ▷ Tabela(string→Lista), Objeto OCCT
2:      código ← FunçãoHash(elemento)
3:      se existe(tabela[código]) então
4:          i ← 1
5:          enquanto i ≤ tamanho(tabela[código]) faça
6:              elemento_consulta ← tabela[código][i]
7:              se elemento = elemento_consulta então          ▷ Chave encontrada
8:                  return código, i
9:              fim se
10:             i ← i + 1
11:         fim enquanto
12:     fim se
13:     retorna código, -1                                    ▷ Chave não encontrada
14: fim Função
```

Algoritmo 2 Inserção usando Encadeamento Separado

```
1:  Procedimento INSERIR(tabela, elemento)   ▷ Tabela(string→Lista), Objeto OCCT
2:      código, índice ← BUSCA(tabela, elemento)
3:      se índice = -1 então                                  ▷ Novo Elemento
4:          se existe tabela[código] então
5:              tabela[código] ← novo Lista()
6:          fim se
7:          tabela[índice].adicionarNoFinal(chave, valor)
8:      fim se
9:  fim Procedimento
```

O objetivo da exploração topológica é a anotação das geometrias presentes no modelo CAD. Baseado no que foi discutido com maiores detalhes no Capítulo 1, as entidades

que são descritas a partir da definição de primitivas geométricas são as Faces, representadas por superfícies geométricas, e as Arestas, representadas por curvas. Buscando que os *datasets* gerados pela ferramenta sejam completos quando se trata de representação geométrica, todas as superfícies e curvas são armazenados e levados para o arquivo final de *features* geométricas do modelo. Ao passo que as geometrias vão sendo exploradas, uma estrutura de dados com a lista das geometrias anotadas vai sendo gerada. Para formatar essa estrutura, se escolheu dispor as geometrias da mesma maneira que as *features* do *Dataset ABC* [27], dessa forma, técnicas que venham a ser ou que já foram propostas para processar esse *dataset* de pequena escala, serão mais facilmente transpostas para o uso nos *datasets* a serem gerados pela ferramenta aqui sendo descrita.

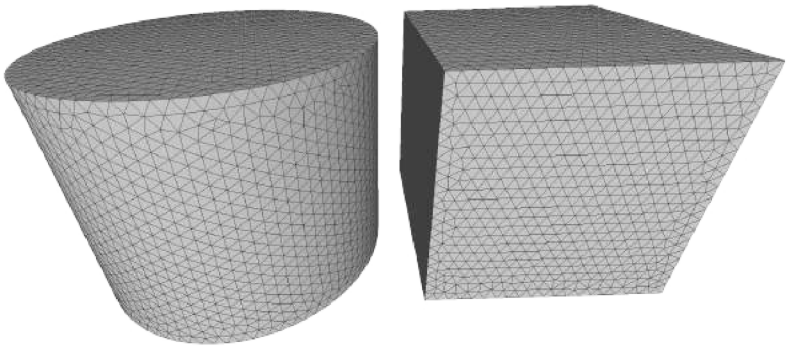
3.2 Geração de Malha Triangular e Relação com as Geometrias

Apesar da malha triangular ser uma representação parcial para geração sintética de pontos no presente trabalho, estas também podem ser usadas como representação final em outros métodos. Como comentado com mais detalhes na Seção 2.2, algoritmos também que podem ser treinados e validados usando malhas como entrada, ou até mesmo como valor de referência no treinamento, visto que métodos como os de reconstrução de superfície tem como objetivo final a geração de uma malha triangular que se ajuste aos dados de entrada [16, 57]. Ainda nesse contexto, duas bibliotecas são exploradas para realizar a geração de malha triangular, as quais diferem substancialmente em método de triangulação, características e casos de uso das malhas geradas, são elas:

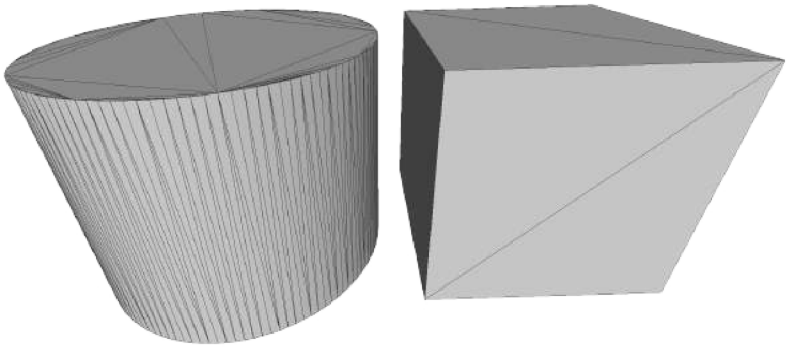
- **Gmsh API [18]:** a biblioteca Gmsh API é capaz de abrir arquivos CAD e fazer uma exploração topológica enquanto amostra malhas triangulares para cada curva e superfície do arquivo de entrada. O algoritmo utilizado por padrão para realizar a triangulação em cada geometria é o Frontal Delaunay [48], porém, em casos de erros, o algoritmo é trocado automaticamente para que a execução não seja interrompida. Entre outras características, o principal diferencial do processo de triangulação do Gmsh é a possibilidade definir o tamanho máximo de aresta que um triângulo pode ter. Dessa forma, é possível gerar malhas com triângulos mais uniformes.

- **OCCT [51]:** além de explorar topologicamente e realizar uma série de operações com o modelo CAD e suas partes, a biblioteca OCCT também possibilita realizar o processo de triangulação de forma incremental, ou seja, uma triangulação a cada entidade geométrica. Dessa forma, as malhas triangulares de cada entidade são agregadas a uma única malha que representa o modelo CAD por completo. Contudo, o processo de geração do OCCT não permite a definição do tamanho máximo de aresta dos triângulos, prejudicando a uniformidade das malhas triangulares ge-

radas.



(a) Malha gerade pela Gmsh API.



(b) Malha gerada pela OCCT.

Figura 6: Malhas triangulares geradas para um CAD simples de um cilindro e um cubo. Em **(a)** é possível ver a malha gerada pela Gmsh API, a qual tem triângulos mais regulares e vértices se distribuindo ao longo das faces. Já em **(b)**, é possível ver a malha gerada pela OCCT, com vértices apenas nas arestas e triângulos que se expandem entre eles ao logo de toda extensão das faces.

A Figura 6 demonstra o resultado da geração utilizando Gmsh API (Fig. 6a) e OCCT (Fig. 6b). As diferenciações de uniformidade das malhas que podem ser observadas acontecem porque no processo de tesselação[1] das superfícies do OCCT, os principais parâmetros estão mais conectados ao erro residual entre a malha e a superfície original do que relacionados as características individuais de cada triângulo. Na Figura 7 é possível identificar as variáveis da tesselação que se relacionam a esses parâmetros, nesse caso a distância d é o erro residual entre o polígono gerado em verde e a superfície contínua original em amarelo. Por outro lado, o ângulo entre as normais $N1$, $N2$ e $N3$ também é considerado no processo de geração. Dessa forma, o método limita a deflexão linear e a deflexão angular, que são, respectivamente, a maior distância d admitida e a maior diferença angular dentre duas normais de um mesmo polígono. Então, é possível entender a malha do cilindro da Figura 6b, já que ao mesmo tempo que as arestas das

[1] Tesselação é o processo de discretização de superfícies contínuas para que sejam recobertas por uma série de polígonos.

circunferências dos tampos do cilindro são muito pequenas, aquelas presentes no corpo dessa geometria se expandem ao longo de toda superfície, conectado diretamente o tampo inferior ao superior. Apesar de todas essas características também serem presentes no processo de geração da Gmsh API, a possibilidade de definir um tamanho máximo para as arestas dos triângulos tem maior precedência e atua regularizando a malha triangular para que tenha maior uniformidade.

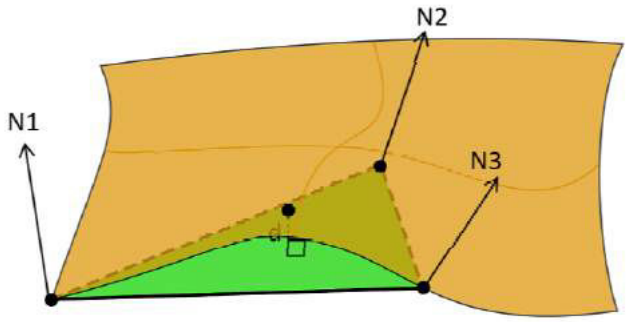


Figura 7: Processo de Tesselação de Superfícies feito pelo OCCT [51]. A superfície contínua em amarelo está sendo recoberta por polígonos como o triângulo em verde. A distância d é limitada no processo pelo parâmetro de deflexão linear. Da mesma forma, o maior desvio angular entre os pares dos vetores normais $N1$, $N2$ e $N3$ é limitado pelo parâmetro de deflexão angular. [52]

As malhas geradas pela OCCT tem pouca relevância para algoritmos que buscam trabalhar diretamente com essa representação 3D, seja como entrada ou como referência de saída. Isso se da pelo fato de que a finalidade do algoritmo de geração de malhas nesse caso é apenas possibilitar a visualização em múltiplos *softwares* que trabalham com malhas sem disponibilizar o arquivo CAD com as geometrias originais. Já o processo de geração de malhas através do Gmsh API, fornece mais parâmetros que possibilitam a geração de malhas com triângulos de arestas mais uniformes, os quais se parecem muito mais com as malhas triangulares geradas em contextos reais, tendo maior valor, seja como representação 3D a ser utilizada como entrada para algoritmos, quanto como referência de saída no treinamento de métodos de reconstrução de superfície. A semelhança maior entre as malhas da Gmsh API com as malhas reconstruídas a partir de dados reais pode ser observada analisando a cena do *dataset* Scannet [12] apresentada na Figura 8, onde os triângulos tendem a ser menos alongados e mais uniformes, similar ao padrão apresentado na Figura 6a. Nesse contexto, é possível comentar sobre a PrimitiveNet [24], que apesar de ser intitulada como um método de segmentação de instâncias geométricas em nuvens de pontos, a representação 3D que os autores realmente utilizam como entrada para o método são as malhas triangulares. Para isso, os vértices dos triângulos são tratados como pontos e as arestas entre os vértices são utilizadas para formar um grafo de conectividade entre os pontos, o qual é utilizado pelo algoritmo para identificar correspondências entre geometrias atribuídas a diferentes pontos conectados entre si. Nesse

contexto, as malhas triangulares geradas pela OCCT não podem ser utilizadas, visto que estruturas planas e cilíndricas não recebem nenhum ponto em suas superfícies (Fig. 6b), apenas nas extremidades, o que acabaria por gerar uma nuvem de pontos incapazes de representar superfícies geométricas.



Figura 8: Malha triangular de uma cena do *dataset* Scannet [12], na qual é possível observar qual o padrão de formação e distribuição dos triângulos em malhas geradas a partir de dados reais.

Apesar das vantagens das malhas geradas pela Gmsh API, sua maior quantidade de parâmetros configuráveis para a geração leva a necessidade de uma configuração específica para cada modelo CAD, na qual, caso não seja feita de maneira adequada, a triangulação apresenta um erro durante a execução e não finaliza o processo. Essa característica é ainda mais evidenciada em modelos de larga escala, os quais contam, ao mesmo tempo, com uma alta quantidade de estruturas e com um alto nível de detalhes em algumas delas. Dessa forma, o algoritmo acaba tendo dificuldade de triangular as regiões mais grosseiras e os detalhes mais finos usando o mesmo conjunto de parâmetros. Somado a isso, como as malhas do Gmsh API são mais detalhadas e contam com mais triângulos, o problema de se trabalhar com estruturas muito grandes mais uma vez aparece, visto que essas malhas acabam ficando muito densas e o consumo de memória durante o processo de geração acaba sendo inviável, mesmo para computadores modernos. Logo, as dificuldades aportas unidas ao fato de que no presente trabalho se planeja usar as malhas triangulares apenas como representação parcial conduz à conclusão de que a utilização da OCCT para a gerar as malhas é mais adequada. Todavia, a ferramenta desenvolvida conta com as duas formas de triangulação, portando a geração com Gmsh API está disponível para uso para que se possa gerar dados mais adequados para outros contextos que não sejam os mesmos do presente trabalho, como os já comentados anteriormente.

Durante a geração da malha, seus dados de vértices e triângulos são integrados com os

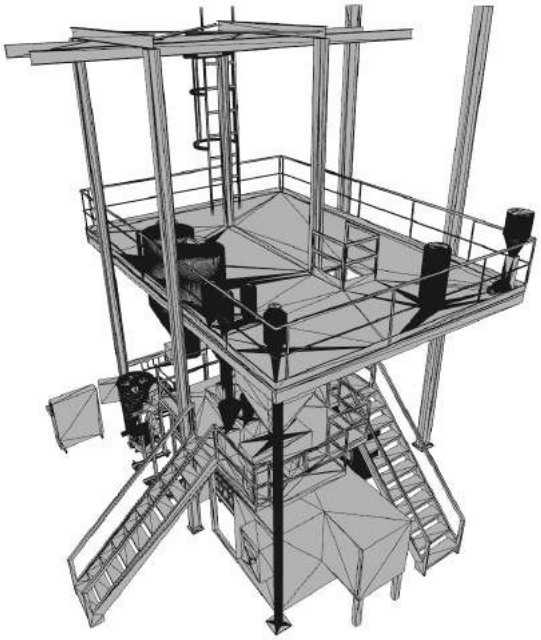
parâmetros das geometrias, vindos do exploração de anotação geométrica. Esse processo é claramente facilitado utilizando OCCT, visto que a geração de malha é incremental e se sabe quais vértices e triângulos descrevem cada entidade geométrica. Contudo, o processo também é direto nas malhas da Gmsh API, visto que o *kernel* utilizado para trabalhar com os arquivos STEP também é o OCCT, o que garante que a leitura do arquivo de entrada será feita da mesma forma. Finalmente, a estrutura chave-valor de curvas e superfícies recebe, para cada geometria, os parâmetros relacionados com a malha triangular. No caso das curvas, uma lista de índices dos vértices que representam aquela curva na malha são integrados a estrutura que armazena os parâmetros. Já nas superfícies, além da lista de vértices, uma lista de índices dos triângulos também é integrada a anotação das geometrias. Ao final do processo, a malha triangular é escrita em um arquivo em formato OBJ e a estrutura chave-valor é serializada em YAML, JSON ou PKL.

3.3 Geração de Nuvem de Pontos

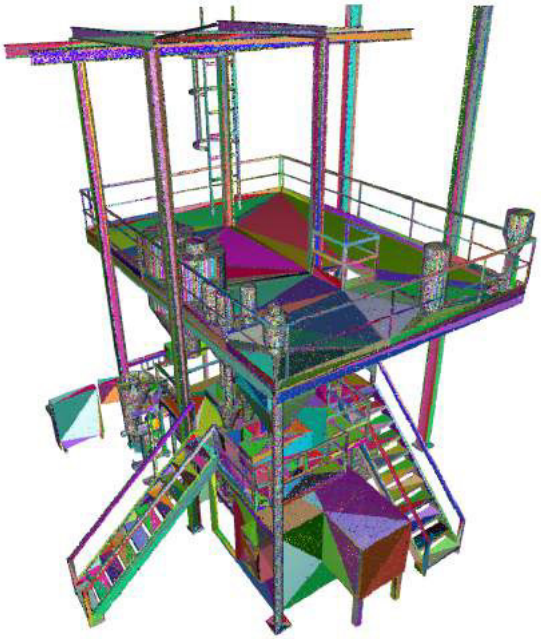
Utilizando a malha triangular como representação 3D intermediária, se faz possível a geração de nuvem de pontos. No presente trabalho, duas abordagens são abordadas, são elas: amostragem uniforme de pontos (Seção 3.3.1) e simulação de LiDAR através de *Ray Casting* (Seção 3.3.2). Além da geração de pontos, criar uma relação entre os pontos gerados e as geometrias anotadas é um requisito importante da ferramenta. Esse processo é feito armazenando, para cada ponto, o índice do triângulo da malha sobre o qual ele foi gerado, o que pode ser observado na Figura 9. A Figura 9a demonstra uma malha produzida pela ferramenta aqui proposta e a Figura 9b ilustra uma nuvem de pontos gerada na superfície da mesma, na qual a cor de cada ponto simboliza o índice do triângulo sobre o qual o ponto foi gerado, levando a nuvem de pontos a também adotar o mesmo padrão triangulado da malha. Dessa forma, é possível relacionar a nuvem de pontos com a malha triangular que, por sua vez, está relacionada com as geometrias.

3.3.1 Amostragem Uniforme de Pontos

A amostragem uniforme é uma das maneiras mais diretas de se gerar pontos em uma malha triangular. O processo se baseia em distribuir um conjunto de N pontos uniformemente ao longo da superfície da malha. Para isso, cada triângulo de índice k da malha recebe um peso w_k de acordo com a razão de sua área com relação a área total da malha, como demonstrado na Equação 1. Dessa forma, o número de pontos n_k a ser amostrado em cada triângulo é calculado a partir do produto entre o peso e o número total de pontos (N) escolhido inicialmente, seguindo a seguinte formulação matemática: $n_k = [w_k \times N]$. Logo após, cada triângulo recebe n_k pontos seguindo a tendência de uma distribuição aleatória uniforme em sua superfície.



(a) Malha triangular gerada na superfície de um modelo CAD utilizando OCCT [51].



(b) Nuvem de pontos gerada em cima da malha triangular. A cor dos pontos simboliza o índice único que identifica sobre qual triângulo cada ponto foi gerado.

Figura 9: Malha triangular (a) e nuvem de pontos (b) geradas a partir da ferramenta. A cor dos pontos demonstra a relação mantida entre as duas representações 3D, identificando em qual triângulo cada ponto foi gerado.

$$w_k = \frac{A_k}{\sum_{i=1}^{N} A_i} \tag{1}$$

As nuvens de pontos gerada a partir dessa abordagem tem a quantidade de pontos controlada diretamente pelo parâmetro N, o que pode ser útil em alguns cenários, mas também pode acabar gerando nuvens super densas, quando o modelo de entrada for pequeno, ou super esparsas, quando o modelo de entrada for grande, dificultando a escolha de apenas um valor para gerar o *dataset* inteiro. Contudo, outros dados podem ser usados em heurísticas de seleção automática do parâmetro N, como, por exemplo, a área total da malha. Dessa forma, é possível amostrar mais pontos malhas com maior área de superfície e menos pontos nas menores, atingindo uma maior equidade na resolução de pontos dos modelos.

Esta amostragem está implementada na biblioteca Open3D [64], contudo, é necessário manter para cada ponto a informação do índice do triângulo sobre o qual ele foi gerado. Dessa forma, a função da Open3D foi modificada para que também retorne essa informação e possibilite a conexão dos pontos com as malhas, gerando um resultado como o ilustrado na Figura 9. No contexto do ajuste de primitivas geométricas, essa abordagem é a mais utilizada, visto que os métodos da literatura como ParseNet [55] e HPNet [63] pegam as malhas do *dataset* ABC [27] e transformam em nuvens de pontos utilizando essa abordagem [55], todavia, em modelos de pequena escala.

3.3.2 Simulação de LiDAR através de *Ray Casting*

O *Ray Casting* [49] é uma técnica de renderização simples que permite determinar quais objetos de uma cena 3D estão visíveis a partir de um ponto de observação determinado. Para isso, normalmente se simula uma câmera disposta no ponto de observação com seu campo de visão apontando para o objeto sendo observado. Logo após, a partir do tamanho do campo de visão e da resolução da câmera simulada, feixes de luz sintéticos são emitidos para que possam ser refletidos pela superfície dos objetos da cena e retornar informações como cor, ponto de colisão, intensidade da luz refletida, normal da superfície ao redor, entre outras. No presente trabalho, as características como cor e intensidade de luz não estão sendo utilizadas, contudo, os pontos de colisão de diferentes feixes podem ser agregados para produzir uma nuvem de pontos e as normais da superfície de cada um podem ser incorporadas como uma característica útil para os métodos utilizarem.

Retomando a motivação do trabalho, o projeto AeroScan utiliza um *drone* comercial com um escâner 3D de tecnologia LiDAR embarcado. Esse sensor conta com um campo de visão vertical de 300° e um campo de visão horizontal de 360°, além de ter uma resolução de 0.09° em ambos eixos. Dessa forma, simular uma câmera no processo de *Ray Casting*, como usualmente é feito, não representaria adequadamente o sensor do projeto que motiva o trabalho. Logo, o processo de geração dos feixes de luz a serem

lançados segue o modelo e os parâmetros do sensor utilizado no projeto. Junto a isso, é importante considerar que o sensor LiDAR está embarcado em um *drone* que é capaz de voar ao redor da estrutura a ser reconstruída e capturar pontos a partir múltiplas vistas da mesma. Assim, todas nuvens de pontos parciais devem ser registradas em uma única nuvem no final do processo para que essa conte com mais detalhes de toda a estrutura sendo observada, da mesma forma que é feito no processo de captura do projeto.

Dada a malha triangular gerada no processo anterior, o *drone* simulado deve ser posto em múltiplos pontos de vista para que possa realizar a observação da estrutura por completo. O processo de decisão de quais são os melhores pontos de vista é um problema complexo e explorado através de diversas abordagens na literatura, contudo, esse não é o foco do presente trabalho. Dessa forma, a abordagem adotada foi a utilização das dimensões da malha triangular como referência para ajustar um semi-elipsoide ao redor da estrutura, sob o qual serão amostrados pontos de observação, como demonstrado na Figura 10. Como se pode observar, a geometria é um elipsoide completo que é cortado no $z = 0$, ou seja, no plano xy que representa o chão do espaço 3D. Logo, o ajuste da geometria ao redor da estrutura é feito através do processo de otimização dos parâmetros a, b e c da Equação 2, que é a equação geral do elipsoide. Para isso, se busca a solução de um problema de mínimos quadrados recebendo como entrada 3 vértices que não sejam coplanares nem colineares de uma caixa delimitadora que envolve a estrutura e é duplicada no eixo $-z$.

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1 \tag{2}$$

Com os parâmetros do elipsoide ajustados, pontos de vista para o *drone* são espaçados uniformemente ao longo de toda sua superfície, como ilustrado na Figura 11. Esse processo é feito utilizando o mapa *UV* da geometria, de forma que o número de pontos de vista diferentes dependa primeiramente, do perímetro da elipse formada pela seção transversal no plano zy e, logo após, do perímetro da elipse formada pela seção transversão no plano xy em cada altura em que pontos de vista para o *drone* são dispostos. Assim, através de um parâmetro que defina a distância entre as vistas e de outro que defina a distância miníma do *drone* ao plano xy, a solução é capaz de gerar as múltiplas vistas para que as coletas possam ser realizadas.

A partir das vistas do *drone*, o processo de *Ray Casting* pode finalmente ser executado em cada uma delas. Para isso, as equações das retas que regem cada um dos feixes das múltiplas vistas são geradas previamente utilizando os parâmetros do sensor que foram apresentados anteriormente com referência. Contudo, como o *drone* nunca se posiciona em uma região interna da estrutura, não é necessário fazer a varredura vertical e horizontal com campo de visão maior do que 180°. Dessa forma, considerando a resolução de 0.09°, se gera uma matriz de raios de dimensões ($h \times w$), onde $h = w = \frac{180}{0.09} = 2000$, para

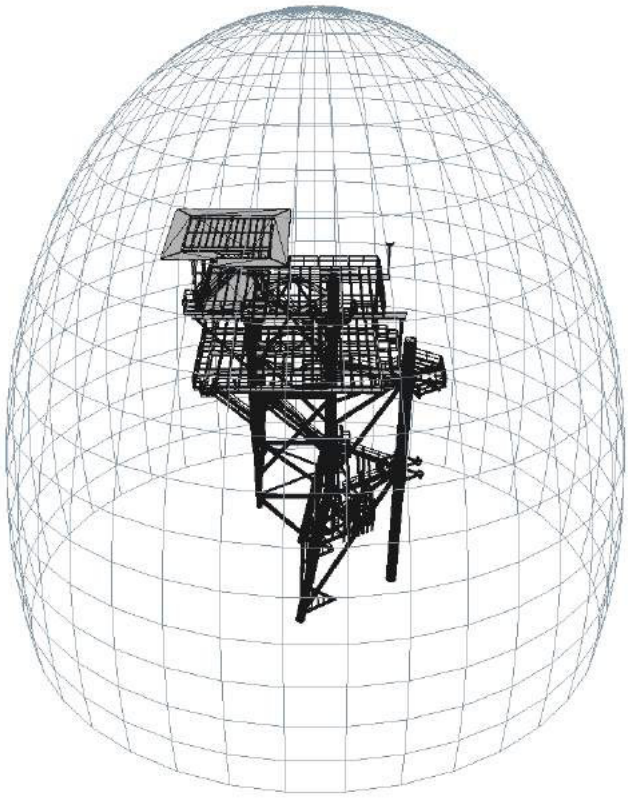


Figura 10: Semi-Elipsoide ajustado para cobrir toda a estrutura representada por uma malha triangular.

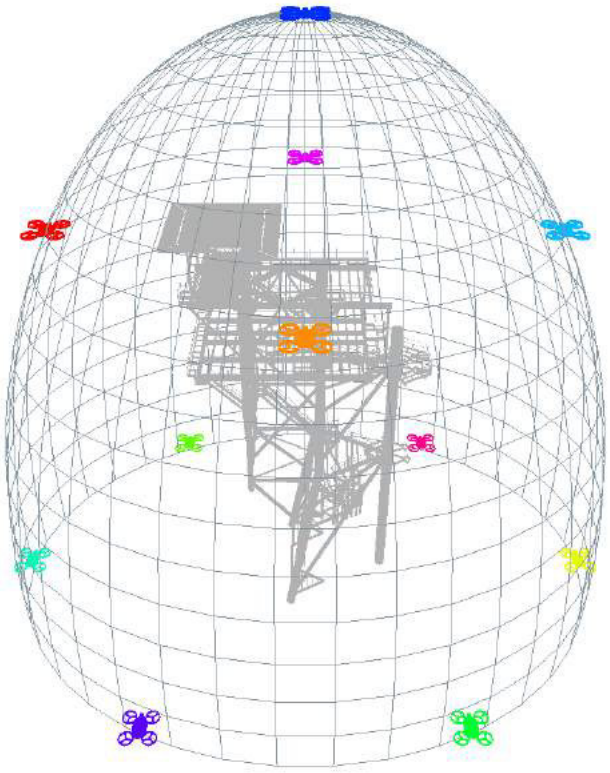
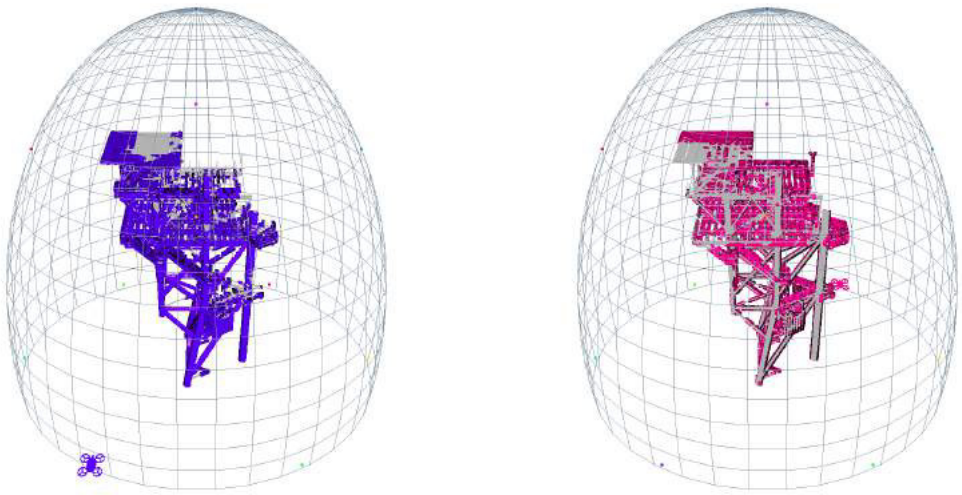


Figura 11: Pontos de Vista para o Drone dispostos ao longo da superfície do Semi-elipsoide.

cada ponto de vista. Logo, 4 milhões de raios são lançados em cada vista, onde cada um deles é capaz de gerar um ponto caso colida com algum triângulo da malha triangular. O processo está está ilustrado para duas vistas diferentes na Figura 12.



(a) Ponto de Vista Frontal (b) Ponto de Vista Traseiro

Figura 12: Nuvens de Pontos capturadas em diferentes pontos de vistas de observação do *Drone*. Em **(a)** é possível ver um ponto de vista frontal, indicado pelo *drone* em roxo na parte inferior esquerda da imagem. Já em **(b)** é possível observar um ponto de vista traseiro, indicado pelo *drone* em rosa atrás da estrutura.

Após o processo de captura a partir de todos os pontos de vista, as nuvens de pontos parciais são registradas acumulativamente em uma única nuvem de pontos final da estrutura por completo. A execução acontece em pares, onde a cada iteração a nuvem de uma vista é registrada na nuvem acumulada das vistas anteriores. Nesse contexto, o processo de registo em par consiste na aplicação de um limiar na distância fornecida por um algoritmo de busca dos K vizinhos mais próximos entre as duas nuvens. Dessa forma, caso haja $K = 1$ pontos na nuvem acumulada com distância até ele menor do que o limiar, o ponto é considerado repetido e não é adicionado.

Apesar dessa metodologia estar sendo proposta para buscar maior similaridade entre os dados sintéticos e os dados reais que são capturados pelo sistema do projeto, não são todos os aspectos físicos que estão sendo simulados. Variáveis como: ruído do sensor LiDAR, ruído de posição do *drone*, tempo de leitura completa do sensor e distorções de movimento da estrutura ocasionadas pelo vento, não estão sendo reproduzidas. Embora a simulação desses conceitos seja importante para reduzir a lacuna entre os dados reais e os sintéticos, o escopo do presente trabalho não se expande até esse ponto, abrindo possibilidades de trabalhos futuros.

3.3.3 Diminuição da Lacuna entre dados Reais e dados Sintéticos

Ainda que nem todos os aspectos físicos possam sem simulados, alguns deles traçam importante papel na busca de uma maior similaridade entre os dados sintéticos e os dados reais. Voltando a atenção para as nuvens de pontos, que são a principal representação 3D utilizada no trabalho, os sensores que são normalmente utilizados para gera-las costumam produzir efeitos característicos e importantes que sejam reproduzidos em contexto simulado. Conceitos importantes como oclusão e esparsidade local serão discutidos na Seção 3.3.3.1 e, logo após, na Seção 3.3.3.2, será feita uma comparação entre as duas abordagens de geração de pontos e a escolha de uma delas para ser aplicada nos dados a serem utilizados no presente trabalho de forma que se diminua ao máximo a lacuna entre os dados reais e os dados sintéticos será justificada.

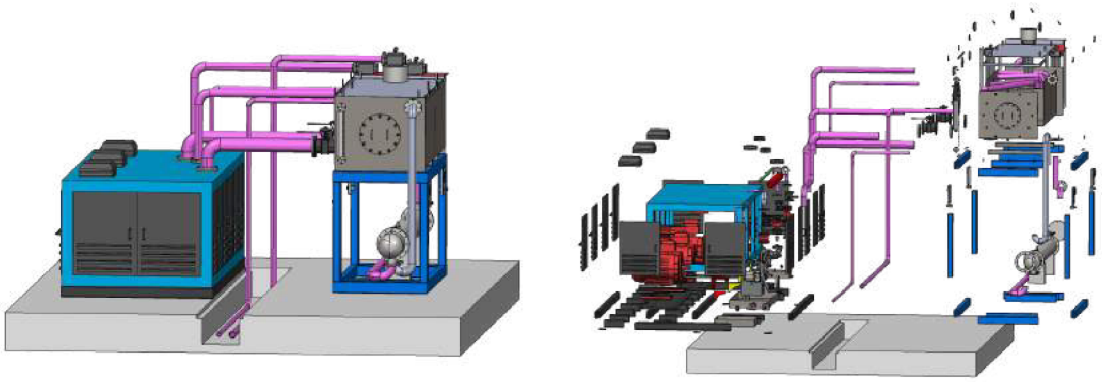
3.3.3.1 Oclusão e Esparsidade Local

Ao se observar uma estrutura real através de sensores, seja ela de pequena ou larga escala, a oclusão e a esparsidade local serão características sempre presentes na nuvem de pontos gerada. A oclusão acontece porque, independente do número de vistas a ser utilizado para que o sensor observe todo o ambiente, sempre haverão regiões que não podem ser observadas, seja por estarem completamente envolvidas por algum anteparo mecânico que reflita completamente o tipo de onda captada pelo sensor ou por acabarem ficando sempre atrás de alguma estrutura nas vistas realizadas, o que depende do tipo de sensor utilizado. O primeiro caso pode ser visto na Figura 13, onde através do processo de explosão das partes do modelo CAD é possível observar algumas estruturas que são inerentemente oclusas, independente do processo de aquisição. Já a segunda, pode ser observada nas regiões marcadas em vermelho na Figura 14, que demonstra algumas regiões oclusas na visão de uma vista de um LiDAR 360° de uma cena no *dataset* SemanticKITTI [4].

A esparsidade local, por sua vez, significa que a nuvem de pontos não terá, em ambiente real, a mesma resolução ao longo de toda estrutura. Dessa forma, o termo simboliza que haverão regiões mais esparsas do que outras na nuvem de pontos. Essa característica pode ser observada nas marcações em azul da Figura 14, onde a densidade da nuvem de pontos nessas regiões é reduzida. Esse fenômeno acontece por diversos motivos, dentre eles:

- **Distância ao Sensor:** quanto maior a distância da superfície medida ao sensor, menor será a sua resolução na nuvem de pontos, visto que menos feixes de onda vindas daquela parte chegarão ao sensor, gerando maior esparsidade na região.

- **Características da Superfície:** superfícies com maior curvatura, mais irregularidades ou altamente texturizadas tendem a produzir regiões mais esparsas na nuvem



(a) Modelo CAD normal.

(b) Modelo CAD com explosão de 40%, mostrando as partes oclusas mecanicamente.

Figura 13: Oclusão gerada por partes que são internas à outras. Em (a) pode-se ver o modelo CAD como entra no gerador de *datasets*. Já em (b), pode-se ver a visualização desse mesmo modelo com um grau de explosão aplicado, possibilitando a visualização de algumas partes internas.

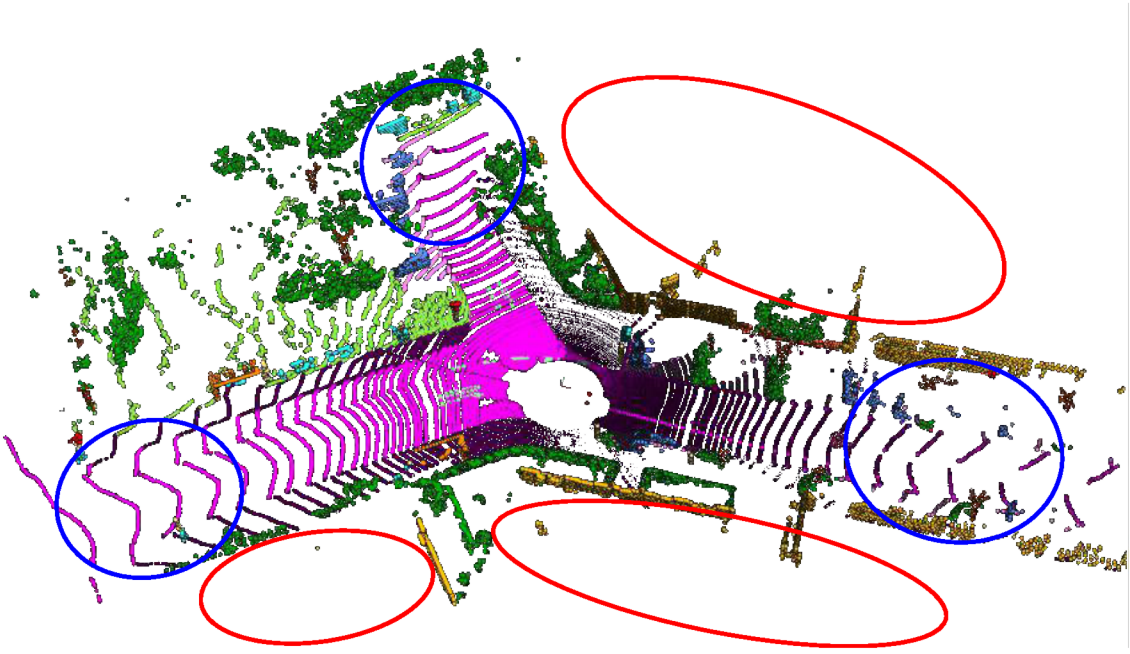


Figura 14: Cena do *Dataset* SemanticKITTI [4] de uma nuvem de pontos capturada por um LiDAR 360° a partir de um único ponto de vista em cenário urbano. As marcações em azul mostram as regiões de maior esparsidade, já as em vermelho mostram regiões oclusas que não puderam ser observadas pelo sensor.

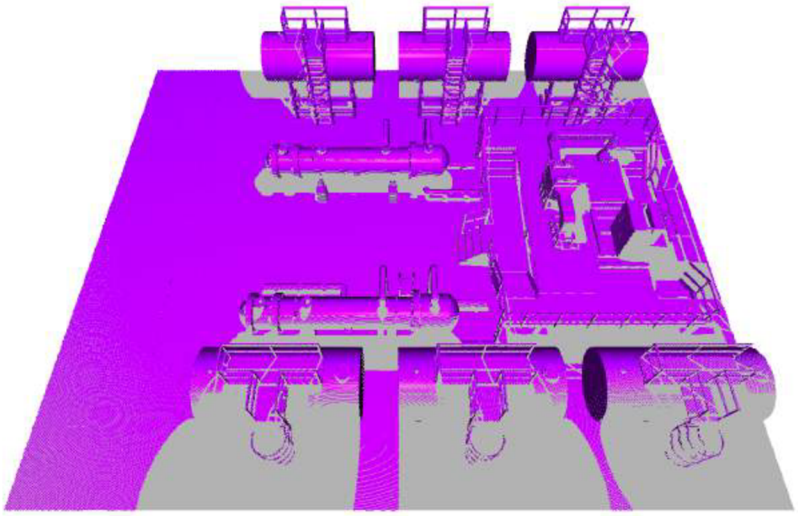
de pontos, visto que os feixes de onda não podem ser capturados ou refletidos corretamente.

- **Regiões Pouco Observadas:** como o processo de aquisição pode ser feito em múltiplas vistas, é lógico pensar que algumas regiões serão observadas por mais vistas do que outras. Dessa forma, dependendo dos parâmetros utilizados no processo de registro das múltiplas nuvens, aquelas regiões que foram observadas em mais vistas tendem a ser mais densas do que aquelas que foram observadas em menos.

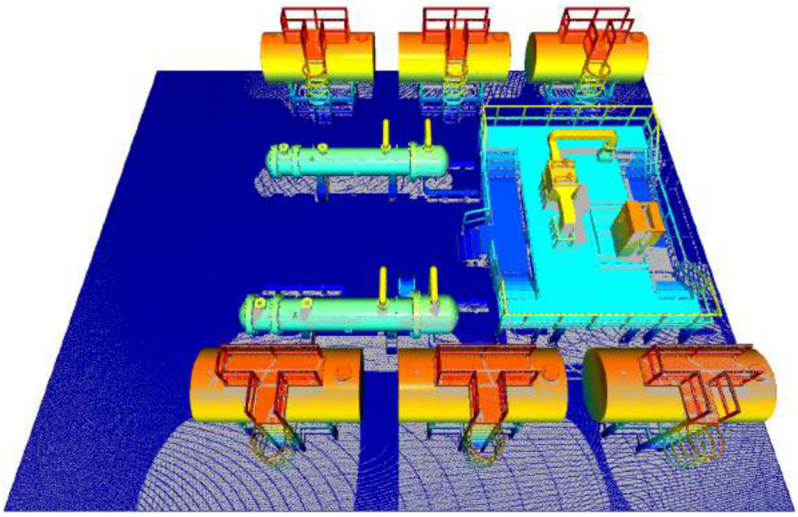
3.3.3.2 Comparação entre abordagens para Geração de Pontos

A geração sintética de *datasets* serve para que não seja necessária a anotação manual do *ground truth*. Esse processo se tornou ainda mais importante no contexto do trabalho pela extrema dificuldade de se anotar, com alta precisão nos parâmetros, primitivas geométricas de diversos tipos contidas em nuvens de pontos. Contudo, a busca pela maximização da semelhança entre os dados sintéticos e os dados reais deve ser um objetivo do processo de geração, visto que, ao aplicar o método treinado em problemas reais, os dados utilizados como entrada para o modelo não serão sintéticos. Dessa forma, dadas as características discutidas na Seção 3.3.3.1, o processo de geração de nuvem de pontos que simula o LiDAR através de *Ray Casting* é mais interessante. A Figura 15 demonstra com clareza a capacidade do processo de simulação de LiDAR de reproduzir o fenômeno da oclusão, observada na Figura 15a, e de reproduzir o fenômeno da esparsidade local, o qual pode ser visualizado na nuvem de pontos da Figura 15b.

Ao se amostrar pontos uniformemente em uma malha triangular, o algoritmo não exclui regiões oclusas, visto que não há um ponto de vista a ser considerado na geração para determinar a visibilidade de um triângulo qualquer da malha. Junto a isso, a esparsidade local acaba não sendo explorada, visto que o número de pontos por triângulo depende da razão da área do triângulo com relação a área total da malha. A comparação entre as duas abordagens está ilustrada na Figura 16, onde ambas as nuvens tem $2.878.465$ pontos e a baseada em simulação de LiDAR demonstra as características desejadas, como oclusão (regiões brancas) e esparsidade local (regiões de cor mais fraca). Além disso, tanto na Figura 15 quanto na Figura 16b é possível observar uma maior semelhança com um cenário real, exemplificado na Figura 14, até mesmo na maneira como os pontos estão dispostos, visto que em ambos os casos um padrão de múltiplas curvas de pontos acaba aparecendo, o que converge diretamente com as características do processo de aquisição de um sensor LiDAR.

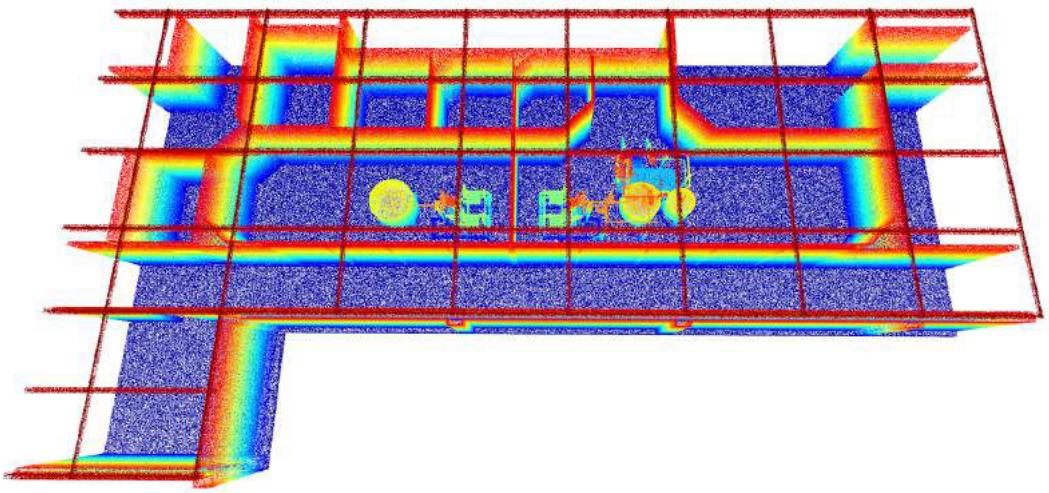


(a) Nuvem de pontos capturada a partir de uma vista superior do modelo CAD. Regiões em cinza demonstram o conceito de oclusão, já que essas puderam ser observadas pelo sensor somente a partir da vista superior.

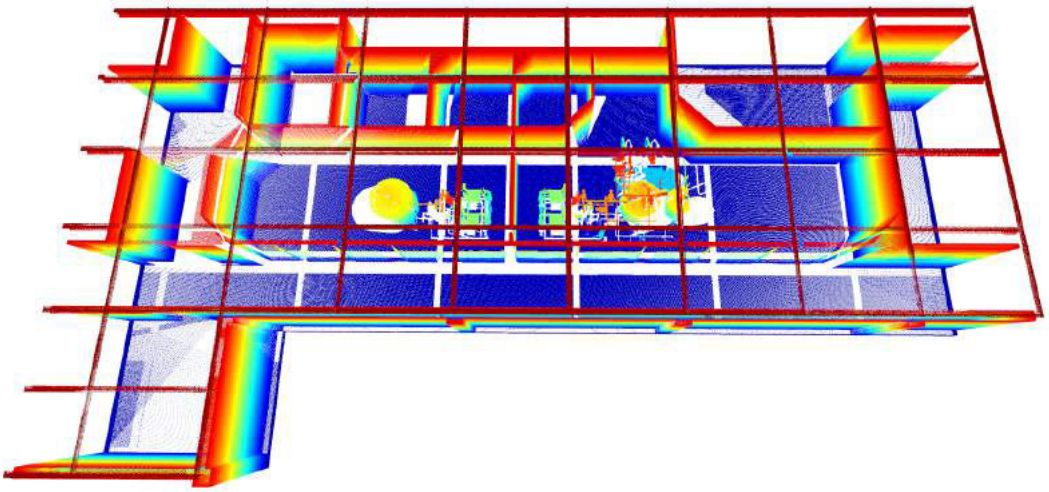


(b) Nuvem de pontos capturadas por múltiplas vistas do modelo CAD. Apesar da oclusão ser menos frequente, as regiões com maior área em cinza demonstram uma esparsidade maior do que as regiões mais coloridas.

Figura 15: Oclusão e Esparsidade Local exemplificados em nuvens de pontos de um modelo CAD geradas a partir da técnica de Simulação de LiDAR por *Ray Casting*.



(a) Nuvem de Pontos Gerada a partir de Amostragem Uniforme.

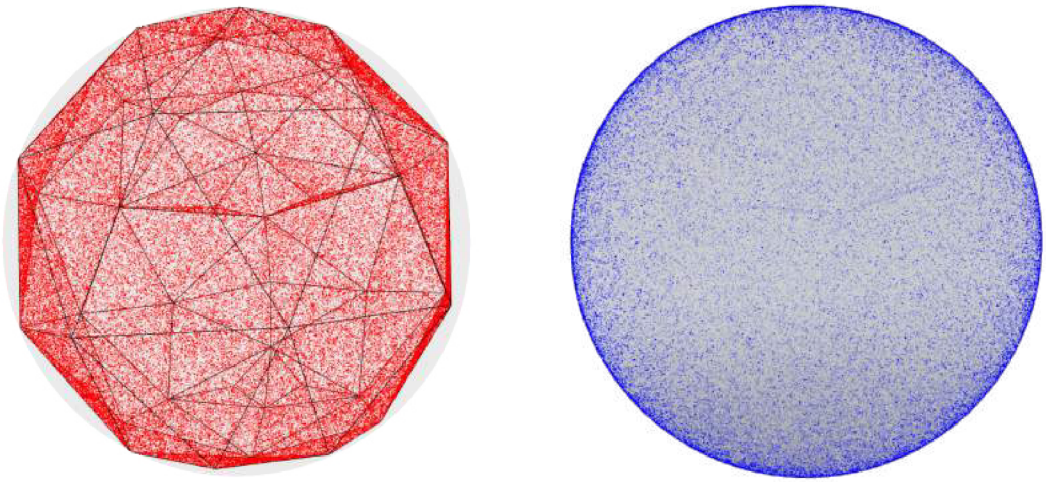


(b) Nuvem de Pontos Gerada a partir de Simulação de LiDAR através de *Ray Casting*.

Figura 16: Nuvens de pontos geradas sobre a malha triangular de um modelo CAD. Ambas nuvens tem $2.878.465$ pontos. Em (a) é possível ver uma nuvem gerada a partir do processo de amostragem uniforme, sem oclusão ou esparsidade local. Já em (b), é possível ver uma nuvem gerada a partir do processo de simulação de LiDAR através de *Ray Casting*, que conta com trechos mais esparsos (cor mais fraca) e com regiões oclusas (brancas).

3.4 Pós-processamentos

Independente do método de geração dos pontos, as malhas triangulares são a representação base que se utiliza para gera-los. Nesse contexto, como comentado na Seção 3.2, o processo de tesselação das superfícies em polígonos triangulares envolve parâmetros que limitam o erro residual de distância ente os triângulos da malha e as superfícies originais. Contudo, apesar de ser possível limitar esse erro, não é possível torná-lo inexistente, o que acaba por transferir esse pequeno erro aos pontos gerados. Não somente as coordenadas (x, y, z) dos pontos carregam esse erro, como também os vetores normais, que são ou obtidos diretamente pelo processo de *Ray Casting* ou são estimados com base nos vetores normais dos três vértices do triângulo de interesse. Esse problema é resolvido a partir de um processo *Points Healing*, onde cada ponto da nuvem é substituído pela projeção na superfície geométrica que faz parte. Logo após, as novas coordenadas (x, y, z) de cada ponto são utilizadas para descobrir as coordenadas UV no sistema de coordenadas local da geometria e, assim, o vetor normal mais adequado para aquela projeção. A solução proposta foi implementada utilizando a biblioteca OCCT, já comentada anteriormente, e sua efetividade pode ser ilustrada na Figura 17, onde uma uma malha triangular de baixa resolução e alto erro de uma esfera gera uma nuvem de pontos bem distante da superfície original (Fig. 17a). Logo após, na Figura 17b, o processo de *Points Healing* produz uma nuvem de pontos projetada na superfície esférica, representando de maneira mais realista a geometria original do arquivo CAD.



(a) Nuvem de pontos (●) gerada sob uma malha de baixa resolução, da qual se pode ver as arestas, de uma esfera (●.

(b) Nuvem de pontos (●) corrigida a partir da técnica de *Points Healing*, projetando todos os pontos da esfera em sua superfície.

Figura 17: Processo de *Points Healing* aplicado a uma nuvem de pontos de uma esfera amostrada uniformemente em uma malha triangular de baixa resolução da mesma.

Grande parte dos métodos baseados em aprendizado que processam nuvem de pontos

no contexto do problema de ajuste de primitivas geométricas, contam com algum processo de normalização prévio para promover maior generalização durante o treinamento. Nesse escopo, métodos como SPFN [31], ParseNet [55] e HPNet [63], aplicam as seguintes normalizações: centralização, alinhamento e reescala. Logo, como os métodos com melhores resultados na literatura de pequena escala utilizam esses procedimentos, se optou por adicioná-los ao *pipeline* de geração de dados aqui proposto. A implementação das três normalizações seguiu o mesmo padrão utilizado no código da ParseNet[2]. De forma mais detalhada, se tem:

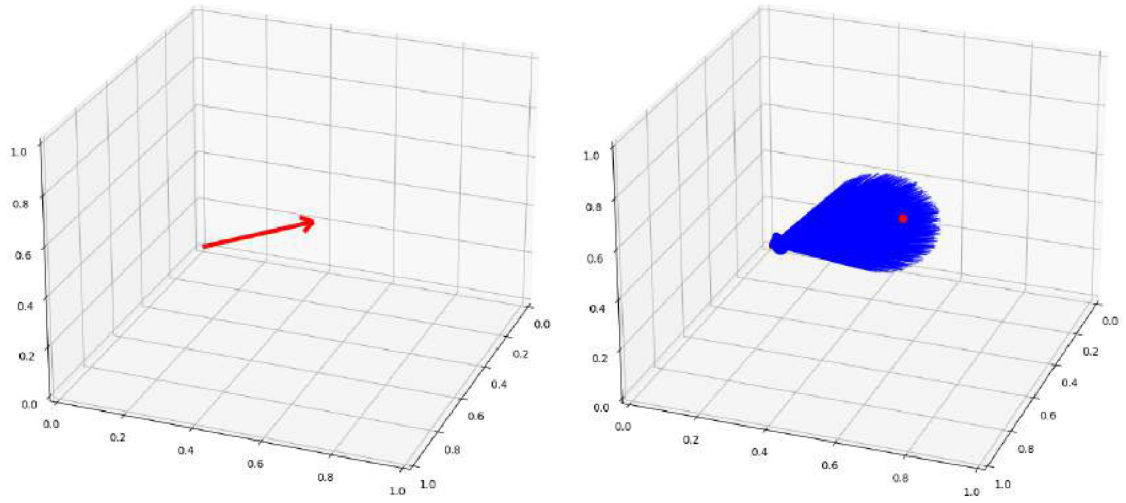
- **Centralização:** centraliza a origem do sistema de coordenadas da nuvem de pontos no seu centroide. Logo, é feita uma translação de todos os pontos, na qual o vetor que parametriza essa transformação é obtido pela média simples entre as coordenadas de todos os pontos.

- **Alinhamento:** rotaciona a nuvem de pontos para que seu eixo x esteja na direção de menor variação dos pontos. Para isso, uma análise das componentes principais (PCA) [62] é feita na nuvem de pontos, onde o autovetor correspondente ao menor autovalor é tratado como o eixo de menor variação. Logo após, uma matriz de rotação R é calculada entre o menor autovetor e o eixo $x = (1, 0, 0)$, a qual é aplicada em todos os pontos da nuvem.

- **Reescala:** multiplica todos os pontos da nuvem por um parâmetro, reduzindo ou aumentando sua escala. No caso específico do que é utilizado pela SPFN e ParseNet, o fator de escala aplicado é calculado para que toda a nuvem de pontos caiba em um cubo de $1 \times 1 \times 1$, com a ideia de abstrair a escala do modelo do processo de aprendizado. Dessa forma, essa abordagem foi implementada no *pipeline*.

Embora as normalizações desempenhem um papel importante no aprendizados dos métodos e até mesmo na avaliação deles, Li and Feng [31] também propõe a adição de ruído artificial para diminuir a lacuna entre os dados reais e os dados sintéticos. Com relação ao ruído nos pontos, o código da ParseNet[2] disponibiliza uma abordagem simples para adição de ruído uniforme em cada ponto da nuvem. Para isso, um valor aleatório entre um limite inferior e um superior é gerado para cada ponto, o qual é utilizado como módulo para uma translação na direção da normal de cada ponto. Somado a isso, também se faz necessária adição de ruído nas normais de cada ponto, nesse caso, a solução implementada tem resultado parecido com a anterior, na qual o vetor normal de cada ponto é rotacionado a partir de um parâmetro aleatório para que fique dentro de um limite de distância angular do vetor normal original. A Figura 18 ilustra a solução, na qual o vetor normal em vermelho da Figura 18a pode vir a ser qualquer vetor na região em azul da Figura 18b, onde foi usado um limite de 10° para ilustrar a solução.

[2]Link do Código da ParseNet: `https://github.com/Hippogriff/parsenet-codebase`



(a) Vetor normal no Espaço 3D.

(b) Possíveis vetores normais ruidosos (●) que podem ser gerados dentro de uma região de $\pm 10°$ ao redor do vetor normal original. (●)

Figura 18: Ilustação so processo de adição de ruído artificial de $\pm 10°$ nos vetores normais dos pontos.

4 MODELO DE AJUSTE DE PRIMITIVAS GEOMÉTRICAS EM NUVENS DE PONTOS DE LARGA ESCALA

O desenvolvimento de um modelo baseado em aprendizado para realizar ajuste de primitivas geométricas em nuvens de pontos de larga escala só é possível a partir do momento que há um *dataset* adequado para a tarefa. Logo, dado que a metodologia proposta no Capítulo 3 oferecerá como resultado a geração de um *dataset*, esse requisito passa a ser contemplado pelo trabalho. A partir disso, os diversos conceitos discutidos ao longo da Revisão Bibliográfica (Cap. 2) podem ser explorados para guiar as decisões do processo de construção de uma solução que consiga aprender e obter resultados de qualidade nos dados.

Como já comentado anteriormente, diversos métodos da literatura recente de aprendizado profundo buscam encontrar geometrias em nuvens de pontos. Contudo, via de regra, os métodos tem foco estabelecido em pequenas nuvens de pontos oriundas do processo de amostragem uniforme nas malhas triangulares dos modelos do *Dataset* ABC [27]. Dessa forma, a metodologia aqui proposta busca adaptar os principais métodos da literatura para que possam ser aplicados em modelos de larga escala. Contudo, cada método é construído a partir de diferentes princípios, os quais podem facilitar ou impossibilitar sua aplicação em nuvens de larga escala. Além disso, o presente trabalho apresenta um contexto mais desafiador, visto que a partir da ferramenta de geração de dados proposta, há a possibilidade de se usar Simulação de LiDAR por *Ray Casting* para gerar as nuvens de pontos, a qual insere padrões não uniformes nos dados, ao contrário das técnicas de amostragem normalmente utilizadas.

Em contextos larga escala, dificilmente as nuvens de pontos podem ser diretamente utilizadas como entrada para os modelos de aprendizado. Como discutido na Seção 2.3, o maior limitante nesse caso é a memória das GPUs, as quais, mesmo que modernas, não contam com memória VRAM suficiente para que todos os pontos sejam processados o mesmo tempo durante o processo de treinamento. Dessa forma, grande parte das técnicas abordam alguma metodologia de divisão prévia dos dados caso as nuvens do *da-*

taset tenham muitos pontos [30, 8, 24, 54], seja por divisão simples em grade ou por uma pré-segmentação baseada em eurísticas isentas de aprendizado. Contudo, se os dados vão ser divididos em partes menores no antes de serem processados, os resultados também serão oferecidos de maneira separada, os quais devem ser, obrigatoriamente, ser agregados seguindo o mesmo padrão em que os dados de entrada foram divididos para produzir o resultado final no modelo completo, o que é o objetivo principal do trabalho.

A metodologia proposta pelo presente trabalho para realizar ajuste de primitivas geométricas em nuvens de pontos de larga escala é intitulada de *"Split and Merge Geometry Fitting"* (SMGF) ou, em português, "Ajuste de Geometrias por Divisão e Agregação". Esta metodologia, que está ilustrada na Figura 19, parte de uma nuvem de pontos e do *ground truth* (GT) de geometrias (**1**), para realizar a divisão dos mesmos em partes menores, processo ilustrado por **(A)** Figura 19 e detalhado na Seção 4.2. Logo após, com base nas várias nuvens das partes divididas (**2**) e de seus respectivos *ground truths* de geometria (**3**), é possível treinar e validar modelos capazes de realizar ajuste de primitivas geométricas nesse conjunto limitado de pontos, processos que são ilustrados por **(B)** e **(C)** na Figura 19 e discutidos na Seção 4.3. Ao fim, os resultados obtidos nas partes divididas de cada modelo completo são agregados para gerar os resultados finais (**5**), os quais podem ser validados frente ao *ground truth* (**1**), processos que são ilustrados por **(D)** e **(E)** na Figura 19 e são discutidos na Seção 4.4. Contudo, antes de discutir cada parte da metodologia, será feita uma análise, na Seção 4.1, com maior detalhamento sobre o problema de ajuste de primitivas geométricas na larga escala, passando pelas principais técnicas da literatura e evidenciando as limitações mais latentes.

4.1 Ajuste de Primitivas Geométrica em Modelos de larga escala

Como já discutido, a categorização de "larga escala" no presente trabalho se refere a dimensão e contexto das estruturas que se pretende processar. Ainda, como a representação dos dados de entrada a ser utilizada são as nuvens de pontos, essa categorização se traduz, de maneira mais explícita, em uma quantidade maior de pontos. Apesar da quantidade de pontos presentes nas nuvens ser um limitante de alto impacto para os métodos existentes, a larga escala também se traduz implicitamente em uma maior quantidade de geometrias, fator que também deve ser considerado ao se propor uma solução para o problema.

Embora estruturas de larga escala normalmente tenham mais pontos e mais geometrias, a geração sintética de dados proposta no presente trabalho permite que se controle a resolução das nuvens geradas e, por consequência, a quantidade de pontos (Sec. 3.3). Ainda, diminuir intencionalmente a quantidade de pontos para facilitar o desenvolvimento de um método aplicável na larga escala traz consigo problemas óbvios, como por exemplo, a representação insuficiente de geometrias menores, podendo resultar até mesmo no

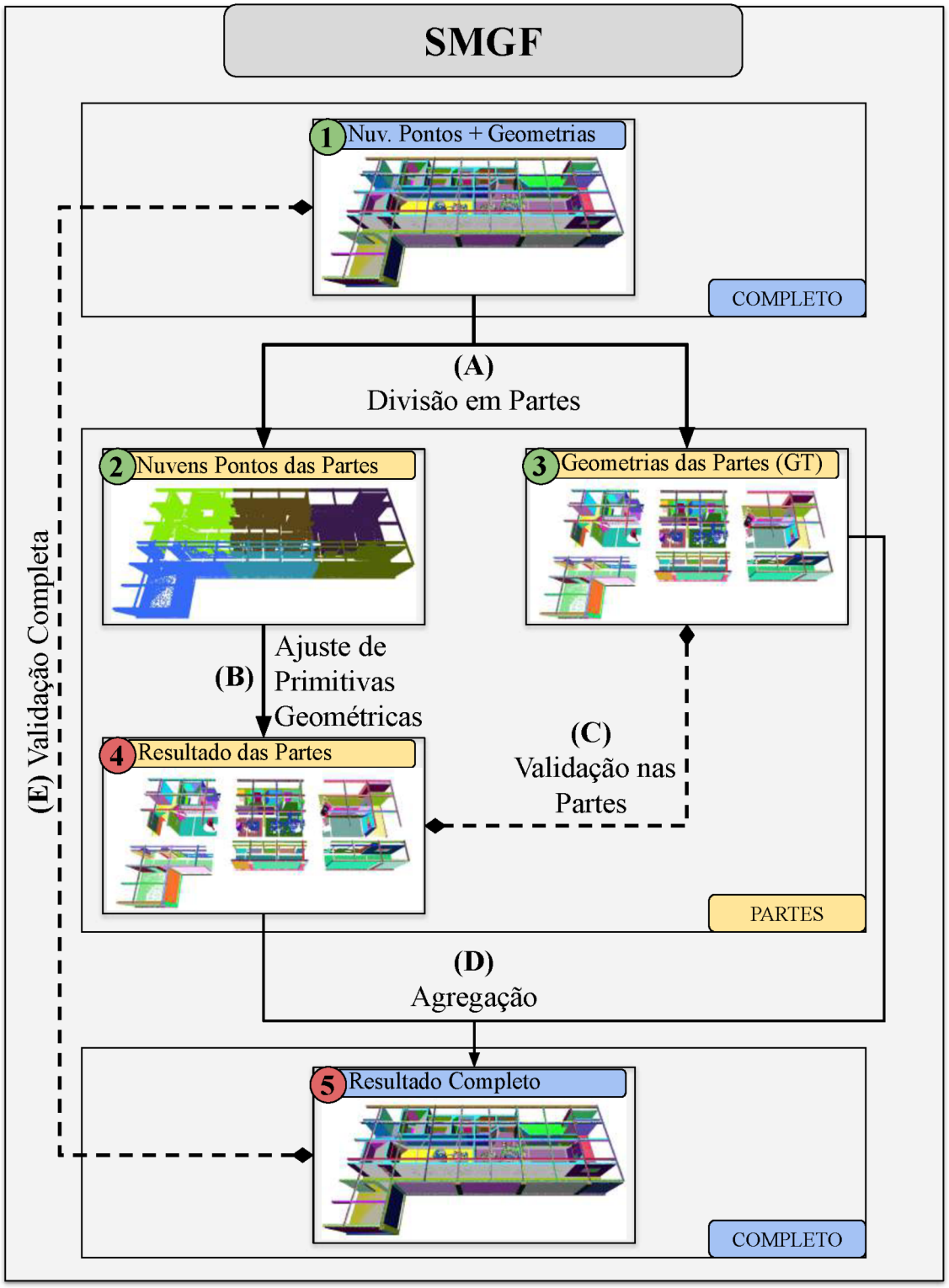


Figura 19: Proposta *"Split and Merge Geometry Fitting"(SMGF)* para resolver o problema de ajuste de primitivas geométricas à nuvens de pontos de larga escala. O processo **(A)** de divisão das nuvens de pontos gera duas abstrações para os dados, os quais podem estar em partes (●) ou completos (●). O método de ajuste de primitivas geométricas é treinado e validado nas partes nos processos **(B)** e **(C)**. Logo após, os resultados gerados, representados pelo número **4**, podem ser agregados por **(D)** e validados frente ao modelo completo original em **(E)**. Os números identificam o estado da informação em cada parte da metodologia e as cores simbolizam se o número se refere a um dado de *ground truth* (●) ou a um resultado (●).

desaparecimento das mesmas. Nesse contexto, a Figura 20 exemplifica o impacto da diminuição de pontos na resolução de amostragem das estruturas e na quantidade de geometrias presentes, visto que a mesma estrutura quando amostrada por 10 milhões de pontos conta com 28.153 geometrias e quando sub-amostrada por 10.000 pontos conta com apenas 156 geometrias restantes. Analisando a figura, é possível perceber que a diminuição progressiva da quantidade de pontos prejudica até mesmo o entendimento visual de que tipo de estrutura o dado está representando. Junto a isso, desconsiderando todas as geometrias com menos de 10 pontos, foi possível observar uma redução da quantidade de geometrias representadas na nuvem de $41,5\%$ do total de geometrias do modelo CAD, na Figura 20a, até $0,23\%$ na Figura 20d.

A diminuição intencional da quantidade de pontos não é somente uma abordagem que leva a perda de geometrias importantes, como também pode acarretar no aumento da distância entre os dados sintéticos, aqui usados para treinamento, e os dados reais, que serão aplicados nos métodos depois de treinados, impactando negativamente nos resultados. Dessa forma, ao se lidar com modelos de larga escala, diminuir progressivamente a quantidade de pontos para possibilitar que os métodos possam processa-los inteiramente não é uma abordagem viável, já que leva a perda de informação e aumento da diferença entre o que é real e o que é sintético.

Visto que a diminuição da quantidade de pontos na entrada leva a problemas indesejados, outra abordagem que emerge dessa dificuldade é possibilitar que os métodos a serem utilizados recebam mais pontos na entrada. Nesse contexto, métodos clássicos não costumam ser limitados com relação a quantidade de pontos que processam, como é o caso do ERANSAC [53]. Contudo, quando se fala em métodos de aprendizado, a limitação existe e é rigorosa, vistos que esses são treinados em GPUs e tanto os modelos, quanto diversas informações relacionadas a eles no processo de treinamento devem estar simultaneamente na memória das placas de vídeo, as quais normalmente são limitadas a valores próximos à $24\ GB$, mesmo nas mais modernas.

Como comentado na Seção 2.3, grande parte dos métodos existente são treinados e validados com nuvens de pontos da ordem de 10000 pontos, como por exemplo: SPFN [31], ParseNet [55], HPNet [63], ComplexGen [21] e SED-Net [33]. Apesar de não serem estudados a fundo com relação ao limite de pontos que podem receber na entrada, esses métodos são propostos para contextos de pequena escala, nos quais altas quantidades de pontos não são normalmente necessárias, dessa forma, o requisito de suportar maiores quantidades de pontos não foi prioritário no processo de idealização dos mesmos. Ao contrário dos métodos anteriormente comentados, a PrimitiveNet [24] foi idealizada para larga escala e alta resolução, a qual se baseia em uma abordagem local que faz com que que o modelo não precise ver todos os pontos de uma única vez, permitindo que os autores treinem e validem o método usando nuvens da ordem de 250.000 pontos.



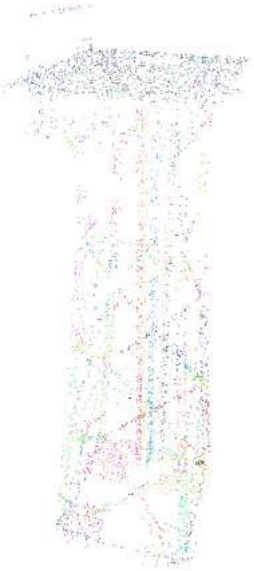
(a) Estrutura representada por 10 milhões de pontos, representando 28.153 geometrias, $41,5\%$ do total.



(b) Estrutura representada por 1 milhão de pontos, representando 9.197 geometrias, $13,5\%$ do total.



(c) Estrutura representada por 100 mil de pontos, representando 1.297 geometrias, $1,91\%$ do total.



(d) Estrutura representada por 10 mil de pontos, representando 156 geometrias, $0,23\%$ do total.

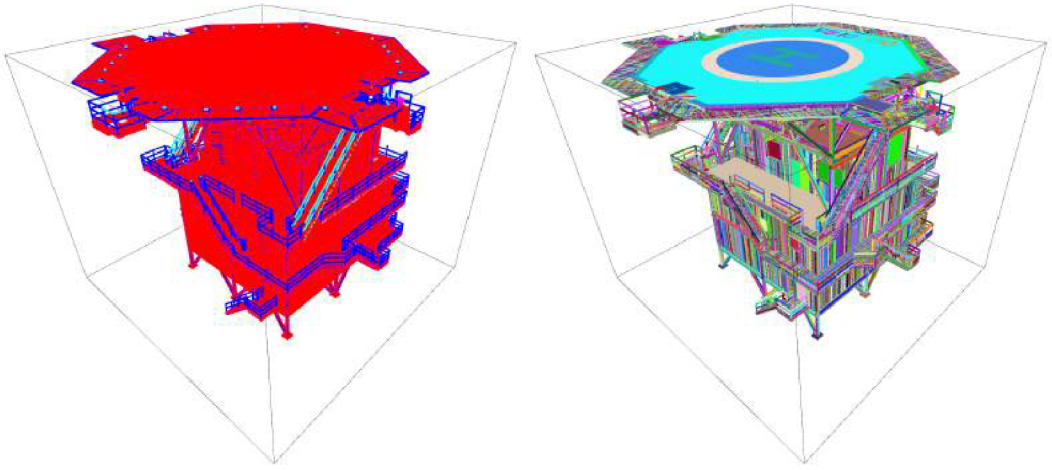
Figura 20: Nuvens de pontos amostradas uniformemente na malha triangular de uma estrutura de larga escala. As nuvens estão segmentadas por instâncias de geometria de uma estrutura com 67.899 superfícies geométricas. Partindo de uma quantidade de 10 milhões de pontos em (a), até 10 mil em (d), é possível notar uma redução da capacidade de representar geometrias, iniciando em 28.153 geometrias e finalizando com apenas 156 restantes.

4.2 Divisão dos Modelos em Partes Menores

Na problemática de encontrar geometrias em representações 3D, apesar da PrimitiveNet [24] possibilitar o uso de uma quantidade maior de pontos na entrada, ainda assim esse método utiliza uma abordagem suplementar de divisão do modelo em partes menores, o que permite que se trabalhe com estruturas de larga escala mantendo um nível de detalhamento alto. Além disso, métodos desenvolvidos para outras tarefas em contextos de larga escala também utilizam algum processo de divisão como base. Landrieu and Simonovsky [30] propôs a divisão da nuvem de pontos de entrada em grupos de características geométricas semelhantes, os quais foram denominados "super-pontos"e são processados separadamente pelo modelo, aliviando a carga de consumo de memória. Por outro lado, a ConvPoint [8] e a Mask3D [54] são dois exemplos de inúmeros métodos que também dividem as nuvens de pontos de entrada, porém, utilizando uma abordagem mais simples, a qual se baseia na utilização de múltiplos cuboides dispostos em forma de grade para delimitar a região de cada nuvem de pontos parcial a ser gerada.

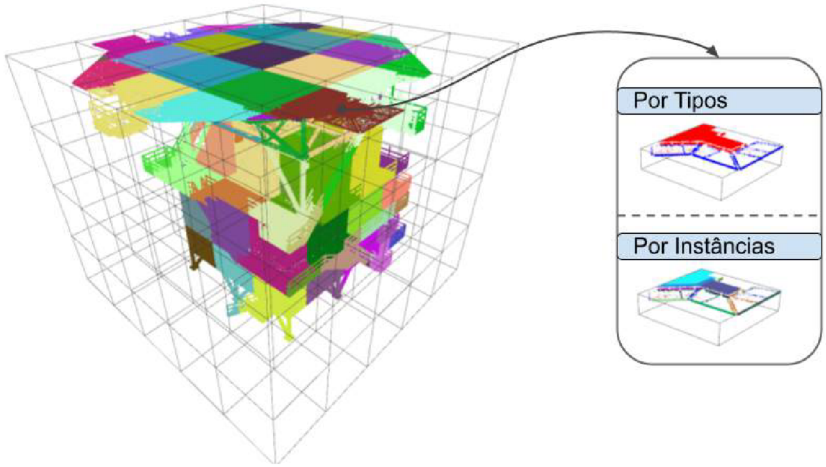
Apesar da abordagem proposta por Landrieu and Simonovsky [30] realizar uma divisão mais complexa e embasada da nuvem de pontos, esse método também estabelece arestas entre os grupos gerados, produzindo um grafo de super-pontos que necessita de uma solução a parte para agregar os resultados. Dessa forma, se optou por uma abordagem mais simples que pudesse ser aplicada de forma mais robusta e direta no problema de ajuste de primitivas geométricas a nuvens de pontos de larga escala. Logo, a divisão em partes menores é feita utilizando uma grade de cuboides de tamanho fixo como caixas delimitadoras para selecionar quais pontos pertencerão a nuvem de cada fragmento a ser gerado da estrutura completa. A Figura 21 ilustra o processo de divisão, o qual parte de uma nuvem de pontos completa da estrutura (Fig. 21a e Fig. 21b) e realiza a divisão em cuboides de tamanho fixo gerando diversas nuvens menores enquanto ainda mantém o *ground truth* geométrico proveniente das nuvens completas dos *datasets* gerados pela metodologia proposta no Capítulo 3. Apesar do processo de divisão em grade ser funcional e amplamente adotado na literatura, métodos como ConvPoint buscam diversificar mais essa abordagem no conjunto de treino com o objetivo de gerar maior generalização durante o aprendizado dos modelos. Para isso, em vez de uma divisão em cuboides topologicamente estruturados em formato de grade, a estratégia adotada nos dados de treino foi a geração de cuboides aleatórios, os quais, apesar de ainda terem tamanho fixo, tem seu centroide selecionado aleatoriamente entre todos os pontos presentes na nuvem de pontos original.

Utilizando como referência o processo ilustrado na Figura 21, o modelo original representado pelas nuvens de pontos completas conta com um volume total de aproximadamente $5.320\ m^3$ e a nuvem de pontos gerada por simulação de LiDAR conta com cerca de $4,4$ milhões de pontos. Dessa forma, como o tamanho dos cuboides utilizado nessa



(a) Nuvem de Pontos completa segmentada por tipo de geometria.

(b) Nuvem de Pontos completa segmentada por instância de geometria.



(c) Nuvem de Pontos completa dividida em 84 partes iguais a partir de uma grade de cuboides com $4 \times 4 \times 4$ metros de dimensão.

Figura 21: Processo de divisão ilustrado utilizando uma nuvem de pontos gerada pelo trabalho em cima de uma estrutura de um heliponto. As nuvens completas (a) e (b) contam com $4,4$ milhões de pontos.

demonstração foi de $4 \times 4 \times 4$, é possível calcular um volume de $64\ m^3$ por cuboide, logo, se espera que o processo de divisão gere cerca de $\frac{5.320}{64} \approx 83$ novas nuvens de pontos de menor dimensão. Contudo, apesar do processo tentar produzir todos os 83 novos modelos nesse exemplo, apenas 74 foram gerados, visto que 9 cuboides acabaram delimitando regiões sem nenhum ponto, as quais não tem serventia para os processos futuros. Nesse contexto, a partir do número de pontos da nuvem completa e do número de partes divididas é possível concluir que as nuvens de pontos geradas tem em média $\frac{4{,}4M}{74} \approx 59,460$ pontos, quantidade que estaria dentro do limítrofe testado pela PrimitiveNet, por outro lado muito acima da quantidade utilizada para testar e validar métodos focados em modelos de pequena escala como ParseNet [55] e HPNet [63]. Dessa forma, além de selecionar o tamanho dos cuboides, o que impacta diretamente na quantidade de pontos de cada parte, também é possível selecionar a quantidade máxima de pontos das nuvens geradas, o que permite que mais experimentos possam ser feitos para que se encontra uma solução mais completa para o problema sem perder tanta informação quanto se perderia reduzindo a quantidade de pontos sem nenhum processo de divisão prévio.

4.3 Ajuste de Primitivas Geométricas nas Partes

Após o processo de divisão, cada parte pode ser tratada como uma estrutura de pequena escala, visto que menos pontos podem ser usados para representar, com resolução adequada, todas as geometrias existentes em cada parte dividida. Logo, os métodos presentes na literatura podem ser adaptados para que sejam treinados e aplicados nesse contexto pra realizar o ajuste de primitivas geométricas nas partes para que posteriormente os resultados sejam agregados e se gere uma solução completa para as estruturas de larga escala. Contudo, cada método traz consigo características únicas que precisam ser consideradas na hora de propor novas versões que se adaptem a metodologia SMGF. Junto a isso, compreendendo todos os detalhes da metodologia aqui concebida e os principais detalhes de cada método, é possível classificá-los em não elegíveis e elegíveis para serem modificados e aplicados em contextos de larga escala.

Não Elegíveis. Os métodos desconsiderados para a aplicação da metodologia SMGF foram classificados dessa forma por razões altamente relacionadas com os aspectos construtivos dos mesmos, impossibilitando ou tornando inviável a adaptação. Os métodos são:

- **SPFN [31]** tem uma limitação muito grande associada ao parâmetro K, que é o número máximo de geometrias que podem ser encontradas em uma nuvem de entrada. Os autores utilizam $K = 24$, contudo, esse valor é muito baixo para modelos industriais com altíssima densidade de geometrias, mesmo que o método seja aplicado apenas nas partes geradas pelo processo de divisão. Além disso, o aumento progressivo do valor K leva a um altíssimo custo computacional e uso de memória.

Para além do problema de quantidade máxima de geometrias, o formato dos dados de entrada da SPFN assume que todas as entidades tem a mesma quantidade de pontos, o que não é verdade nos dados gerados por esse trabalho, visto que os pontos são gerados por modelo completo e não por geometria individual. Logo, com base nessas características, somadas ao fato de que a SPFN foi o primeiro método proposto para o problema com base em aprendizado e, dessa forma, é aquele que tem resultados menos expressivos em contextos de pequena escala, o método foi considerado como não-elegível.

- **ComplexGen [21]** é um método completo e robusto de reconstrução de arquivos CAD. Contudo, por se tratar de um método baseado em redes *Transformers* [60], o custo computacional do processo de treinamento é extremamente alto, requisitando 3 dias de treino em um *cluster* com 8 GPUs Nvidia V100 nos experimentos conduzidos pelos autores. Logo, propor uma adaptação desse modelo para contextos de larga escala sem ter essa infraestrutura para realizar o treinamento é inviável.

  Não somente a complexidade e o custo computacional são problemas, a aplicação da ComplexGen nas partes geradas pela metodologia SMGF também é problemática, visto que apesar das nuvens reduzidas geradas pelo processo de divisão serem de menor tamanho, estas não são construídas da mesma forma que as estruturas de pequena escala. Logo, para um método como a ComplexGen, que tenta reconstruir também o *wireframe* de uma estrutura de pequena escala completa, lidar com pedaços pequenos cortados aleatoriamente de uma estrutura maior foge completamente do escopo de elaboração do método, não sendo elegível para a aplicação da metodologia SMGF.

- **SED-Net [33]** apesar de ser o último dos métodos a ser publicado e atingir o melhor resultado em diversas métricas do problema de segmentação de geometrias em nuvens de pontos, o mesmo ainda, até o momento da escrita desse trabalho, não tem código divulgado para que possa ser adaptado a partir da metodologia SMGF.

Elegíveis. Os métodos elegíveis para adaptação e aplicação nas partes são:

- **ParseNet [55]** é um método capaz de realizar o ajuste de superfícies paramétricas a nuvens de pontos. Os tipos de superfícies suportados são: planos, esferas, cilindros, cones e b-splines. Apesar desse método ser focado na larga escala e explorado pelos autores com nuvens de 10.000 pontos, a metodologia SMGF é capaz de adapta-lo para que possa ser utilizado na larga escala.

- **HPNet [63]** é um método baseado na ParseNet, porém sem a parte de ajuste de parâmetros das primitivas geométricas, apenas com a segmentação das mesmas. Apesar disso, o método implementa algumas teorias diferentes que o permi-

tem ter resultados superiores ao anterior em comparação feita utilizando o *dataset* ABC [27].

Nesse método, assim como seu módulo de *clustering*, o qual efetivamente segmenta os pontos em instâncias de geometria, é executado como um pós processamento e não é baseado em aprendizado, o módulo de ajuste de parâmetros também será. Para isso, o módulo da ParseNet, que é o mesmo da SPFN, será empregado ao final do processamento da HPNet para que se possa obter os parâmetros de cada geometria.

- **PrimitiveNet [24]** já explora os desafios de resolver esse problema na larga escala, propondo um método de abordagem local que é complementado por uma estratégia de crescimento de regiões para produzir os resultados para todo o modelo de entrada. Contudo, a falta de um *dataset* como que está sendo proposto no presente trabalho impediu os autores de treinar o método para realizar segmentação de múltiplas primitivas geométricas em um contexto de larga escala.

  Mesmo que o método já seja mais pronto para contextos de larga escala, a PrimitiveNet não realiza o ajuste dos parâmetros das primitivas geométricas, assim como a HPNet. Logo, se faz necessária a agregação do módulo de ajuste da ParseNet como um pós processamento do processo de segmentação de instâncias geométricas.

  Embora o método seja trazido como uma solução para encontrar geometrias em nuvens de pontos, os principais experimentos conduzidos pelos autores são feitos em malhas triangulares, nas quais os vértices são tratados como pontos e as areatas como um grafo de conexão entre os mesmos. Dessa forma, o método precisou ser modificado para que funcione também em nuvens de pontos sem arestas, utilizando um KNN com $K = 16$ para produzir o grafo de conexão entre cada ponto e seus 16 vizinhos mais próximos, como sugerido pelos autores.

  Os experimentos conduzidos pela PrimitiveNet usam cerca de 250.000 pontos na entrada do método, o qual consegue ainda assim ter um consumo de memória baixo devido a sua abordagem local. Logo, a possibilidade de usar uma quantidade alta de pontos em conjunto com a divisão em partes menores proposta pela metodologia SMGF permite que se perda o mínimo de informação entre todos os métodos ao se processar as nuvens de pontos.

Para além dos métodos baseados em aprendizado, a solução considerada como estado-da-arte clássico é o ERANSAC [53], o qual pode ser aplicado tanto em contextos de larga escala quanto de pequena escala. Dessa forma, essa abordagem será utilizada como base de comparação para os outros métodos que aplicam a metodologia aqui proposta. Por outro lado, esse método exige o ajuste de cinco parâmetros que impactam diretamente na qualidade dos resultados obtidos, como foi demonstrado por Maurell et al. [37] através

da aplicação do mesmo em nuvens de pontos de larga escala. Todavia, com o objetivo de realizar uma comparação agnóstica entre os métodos, o processo de parametrização automático da implementação do ERANSAC no pacote *Shape Detection* [40] da biblioteca CGAL [58] será utilizado, como é adotado nas comparações feitas em contextos de pequena escala, inicialmente proposto por Li and Feng [31].

Apesar do ERANSAC ser capaz de processar os modelos completos inteiramente, também será proposta uma versão do mesmo integrado a metodologia SMGF para mensurar o impacto da abordagem de Divisão e Agregação na qualidade dos resultados. Dessa forma, o **ERANSAC** padrão será comparado com quatro soluções propostas, são elas: **SMGF-ERANSAC**, **SMGF-ParseNet**, **SMGF-HPNet** e **SMGF-PrimitiveNet**. Com isso posto, todos todos os cinco métodos, incluindo o ERANSAC, utilizarão não somente os pontos na entrada como também as normais de cada ponto. Junto a isso, dado que os métodos serão comparados entre si de forma visual e quantitativa, é necessário que os tipos de geometria possíveis de serem detectados e ajustados pelos diferentes métodos seja o mesmo. Dessa forma, como o ERANSAC não trabalha com B-Splines, se optou por não considerá-las nem no treinamento, nem na validação de nenhum dos métodos para ter uma comparação mais justa e significativa.

Como o *dataset*, que será apresentado na Seção 5.2, conta com *ground truth* de geometrias e o processo de divisão é capaz de manter essa referência nas partes menores, as três soluções baseadas em aprendizado serão treinadas no conjunto de treino desse *dataset*. Já com relação a validação nas partes, processo representado por **(C)** na Figura 19, todos os 5 métodos serão executados no conjunto de validação do *dataset*, o qual não é utilizado no processo de treino, para que se possa calcular as métricas que avaliam quantitativamente as soluções e também gerar os resultados visuais que ilustram as qualidades e deficiências de cada uma.

4.4 Agregação dos Resultados Obtidos nas Partes

A partir da aplicação de todos os passos esclarecidos até aqui, os métodos modificados pela metodologia são capazes de dividir os modelos de larga escala em pequenas partes, as quais podem ser processadas pelos algoritmos de ajuste de primitivas geométricas para gerar os resultados parciais. Entretanto, o objetivo final da metodologia é a obtenção de resultados para os modelos completos, o que demanda um método de agregação desses resultados. Todavia, este é um problema complexo que inicia nas geometrias identificadas localmente em cada parte, passa pela união dos resultados de todas as partes de cada modelo completo, e segue para a identificação global de cada geometria encontrada pelo método de ajuste. Como base do processo de agregação, se tem a identificação de correspondências entre geometrias encontradas nas múltiplas partes do modelo completo, dessa forma, a agregação dos resultados é a resolução exaustiva de diversos problemas de

correspondência.

Apesar de complexa, a solução pode ser muito facilitada a partir de uma referência que guie o processo de agregação. Nesse contexto, o próprio *ground truth* da partes, ilustrado como **3** na Figura 19, pode contribuir como um mapa que guie esse processo, simplificando substancialmente o problema, dado que a divisão anteriormente aplicada mantém os identificadores globais de cada geometria do *ground truth* das partes. Dessa forma, assumindo um exemplo de modelo completo com 1000 geometrias, todas elas identificadas por um índice único dentro do intervalo de $[0, 1000)$, ao se dividir esse modelo em K partes, em vez das geometrias assumirem identificadores locais em cada parte, elas mantém os índices originais que tinham no modelo completo, aqueles mesmos no intervalo de $[0, 1000)$. Logo, para cada resultado parcial, sabendo qual a correspondente no *ground truth* de cada geometria identificada nos resultados, é possível se inferir também qual geometria ela representa no modelo completo, resolvendo o problema de agregação.

O grande desafio da abordagem de agregação proposta é, ainda nas partes, encontrar as correspondências entre as geometrias presentes nos resultados e aquelas anotadas no *ground truth*, o que também e requisito do processo de validação nas partes, representado por **(C)** na Figura 19. Esse desafio é tratado no contexto de segmentação de instância como um ”problema de atribuição”, o qual foi definido e solucionado por Kuhn [29] no ano de [1955], ou, como ”reordenação de primitivas” no campo do ajuste de primitivas geométricas [31]. A solução proposta por Kuhn [29] é chamada de *”Hungarian Matching”* e consiste na formação de uma matriz de custo de atribuição de um elemento a outro, a qual é utilizada como entrada para um método de resolução de sistema linear, resultando na melhor atribuição possível entre os elementos das linhas e das colunas.

O *Hungarian Matching* é utilizado até hoje, a partir de diferentes implementações, como é o exemplo da SPFN [31] e da ParseNet [31], as quais montam uma matriz de custo de dimensões $K \times \hat{K}$, sendo K o número de geometrias preditas e $\hat{K}$ o número de geometrias do *ground truth*, utilizando o complemento da métrica RIoU [28] como custo de cada posição da matriz. Porém, de forma anterior a montagem da matriz de custo, os métodos utilizam a representação *one-hot* para transformar em vetor o índice da instância de geometria de cada ponto da nuvem, gerando uma representação $W = N \times K$ para a predição e $\hat{W} = N \times \hat{K}$ para o *ground truth*, sendo N o número de pontos da nuvem de entrada. Dessa forma, ao se aplicar a Equação 3 [31], que calcula a métrica RIoU, será produzida uma representação parcial de dimensão $N \times K \times \hat{K}$, o que não é um problema no contexto de pequena escala em que essa formulação é aplicada. Entretanto, como comentado anteriormente, os índices globais são mantidos nas geometrias do *ground truth*, o que faz com que o valor de $\hat{K}$ seja muito elevado, podendo ser exemplificado pelas 67.899 geometrias presentes no modelo da Figura 20. Junto a isso, a validação completa, representada por **(E)** no diagrama da Figura 19, também é um processo que precisa ser executado e conta com a reordenação de primitivas para isso, contudo, nesse caso são utilizados todos

os pontos e todas as geometrias do modelo completo após a agregação. Exemplificando esse problema a partir do exemplo da Figura 20a, e considerando, de maneira cautelosa, que apenas metade de todas as primitivas presentes nos pontos foram encontradas pelo método de ajuste, teria-se uma matriz de dimensão $10.000.000 \times 14.076 \times 67.899$, ou seja, $9,55 \times 10^{15}$ números em ponto flutuante na matriz, o que resultaria em um consumo de cerca de 38.230 TB de memória, impossibilitando o uso dessa formulação.

$$RIoU(W, \hat{W}) = \frac{W^T \hat{W}}{\|W\|_1 + \|\hat{W}\|_1 - W^T \hat{W}} \tag{3}$$

A solução eficiente em memória que substitui a Equação 3 está apresentada no Algoritmo 3. Com o objetivo de consumir menos memória, se optou por não utilizar a representação *one-hot*, visto que esta, apesar de útil e necessária na utilização de algoritmos de aprendizado, tende a aumentar muito o consumo desse recurso. Dessa forma, os índices que identificam as instâncias geométricas de cada ponto foram armazenados em uma matriz de dimensão $[N \times 1]$ onde, para cada um dos N pontos da nuvem, se armazena um número inteiro que identifica a geometria que este ponto faz parte. Dessa forma, as matrizes $\hat{L}$ e L entram na função com essa representação sendo elas, respectivamente, os índices preditos e os índices do *ground truth*. Após isso, uma contagem de bins[1] é utilizada para contar o número de ocorrências de cada elemento existente nas matrizes de entrada, gerando a matriz $\hat{C}$ de dimensão $[s \times 1]$ e a matriz C de dimensão $[1 \times s]$. Seguindo, o processo para obter as intersecções ($M^{\cap}$) entre todos os pares de geometrias preditas e geometrias do *ground truth* passa pela codificação das duas matrizes de índices da entrada em uma única matriz denominada L^C. A partir disso, a contagem de bins dessa matriz de codificação já produz automaticamente a matriz de intersecções ($M^{\cap}$) entre todos os pares possíveis. Finalizando, a união ($M^{\cup}$) é calculada somando as contagens individuais e subtraindo da intersecção ($M^{\cap}$), onde a segunda é dividida pela primeira para concluir o processo.

Algoritmo 3 RIoU Eficiente em Memória

Função RIOUEFICIENTE($\hat{L}$, L) ▷ $[N \times 1]$
  s ← Max(2*Max($\hat{L}$), Max(L))
  $\hat{C}$ ← ContagemDeBins($\hat{L}$, s) ▷ $[s \times 1]$
  C ← ContagemDeBins(L, s).T ▷ Transposta, $[1 \times s]$
  L^C ← $s \times \hat{L} + L$ ▷ Índices Combinados, $[s \times 1]$
  $M^{\cap}$ ← ContagemDeBins(L^C).reshape(s, s) ▷ $[s \times s]$
  $M^{\cup}$ ← $\hat{C} + C - M^{\cap}$ ▷ $[s \times s]$
  retorna $M^{\cap}/M^{\cup}$
fim Função

Embora a busca dos índices globais das primitivas geométricas seja o problema prin-

[1] Contagem de Bins: `https://numpy.org/doc/stable/reference/generated/numpy.bincount.html`

cipal resolvido pela agregação proposta pelo trabalho, também há a necessidade de se agregar os parâmetros das geometrias, quando as mesmas acabaram sendo divididas em fragmentos pelo processo de divisão. Nesses casos, cada fragmento da geometria completa terá seu próprio conjunto de parâmetros únicos para descreve-la. Logo, os mesmos devem ser combinados de maneira que se obtenha apenas um conjunto para a geometria agregada. Visando simplificar o problema, se escolheu manter os parâmetros do fragmento com maior quantidade de pontos, assumindo que esse tem mais capacidade de gerar parâmetros adequados.

5 RESULTADOS

Com base em tudo o que foi discutido no trabalho e que circundam o problema do ajuste de primitivas geométricas a nuvens de pontos de larga-escala, alguns experimentos foram elaborados para que se possa melhor entender, comprovar e comparar o funcionamento das metodologias propostas nos Capítulos 3 e 4. Inicialmente, será feita uma demonstração do processo de geração sintética de dados proposta no Capítulo 3. Para isso, a Seção 5.1 tem como objetivo demonstrar o passo a passo do funcionamento da metodologia proposta através de um estudo de caso simplificado, comprovando seu funcionamento e demonstrando características importantes do processo. Logo após, com a metodologia de geração demonstrada e validada, uma aplicação mais complexa da mesma será demonstrada na Seção 5.2, na qual a metodologia será utilizada para produzir o *Dataset* LS3DS, que é um objetivo do trabalho e requisito fundamental para que se possa aplicar técnicas baseadas em aprendizado para resolver o problema principal do trabalho. Logo após, na Seção 5.3, a metodologia SMGF (Cap. 4), será validada e suas variações serão comparadas visualmente e com base em métricas largamente discutidas na literatura para demonstrar o funcionamento das soluções propostas. Junto a isso, esta seção também demonstra a capacidade do *Dataset* LS3DS de servir como base de treinamento e comparação para futuros métodos de ajuste de primitivas geométricas a nuvens de pontos de larga-escala.

5.1 Validação da Geração de *Datasets* Sintéticos em Estudo de Caso Simplificado

Visando simplificar a demonstração da validação da metodologia para geração de *datasets* proposta, um modelo simplista com apenas primitivas geométricas básicas será utilizado, o qual pode ser visualizado na Figura 22. Como pode ser observado, o modelo conta com:

- **8 Planos:** o plano de base é um sólido prismático extremamente fino, dessa forma, conta com 6 faces. Ao contrário, os dois planos laterais são apenas superfícies soltas que não participam de nenhuma Casca Fechada no arquivo STEP;

- **2 Cilindros:** um completo, mais ao fundo, e outro incompleto, mais a frente;
- **2 Cones:** um completo, na esquerda, e outro incompleto, ao lado do cilindro incompleto, mais a frente;
- **2 Esferas:** uma completa, na direita, e outra incompleta, acima do cilindro e do cone incompletos, mais a frente.

A partir dessa composição, a malha triangular e o arquivo de geometrias foram gerados. A malha ficou com 7.598 vértices e 14.252 triângulos, o que é aceitável dada a quantidade de superfícies curvas do modelo CAD. A informação de quantidade de triângulos e a justificativa relacionada ao número de superfícies curvas bate com o que está demonstrado na Figura 23, que relaciona cada tipo de primitiva com a quantidade de triângulos envolvidos na representação das entidades daquele tipo. Além disso, o gráfico da Figura 23 também confirma que todos os triângulos da malha estão relacionados a alguma primitiva geométrica, visto que $12.044 + 16 + 316 + 1.876 = 14.252$.

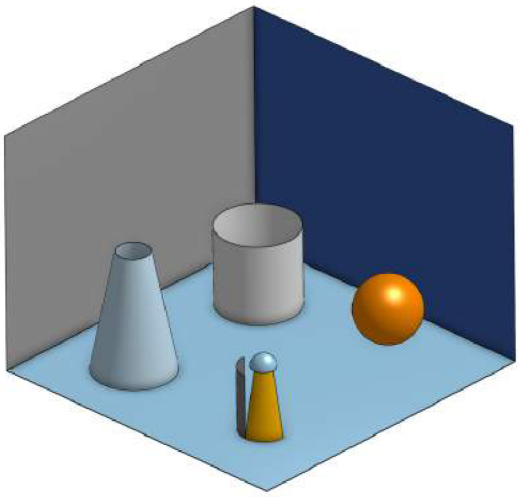


Figura 22: Modelo CAD Utilizado no Estudo de Caso Simplificado.

O arquivo de geometrias é muito extenso para ser trazido por completo ao texto, contudo, a análise cuidadosa desse arquivo mostrou que tanto a quantidade de geometrias, quanto os parâmetros individuais de cada uma delas conferem com o que foi desenhado. Com relação as quantidades de cada tipo, o gráfico da Figura 24, que foi produzido automaticamente no processo de geração dos dados e relaciona a quantidade de entidades geométricas com cada tipo de primitiva demonstra conformidade com o que foi desenhado. Com relação a parametrização das geometrias e relação com a malha triangular, o processo de geração de pontos produz uma validação quantitativa e qualitativa que ajuda a entender se tudo está de acordo.

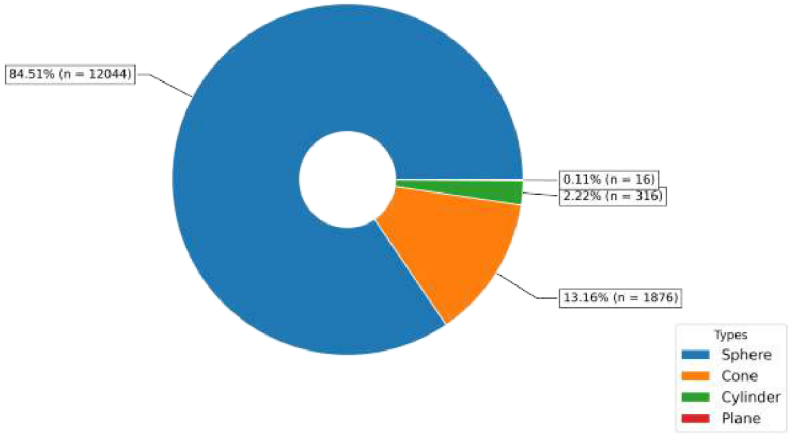


Figura 23: Gráfico do Número de Triângulos por Tipo de Primitiva no Estudo de Caso Simplificado.

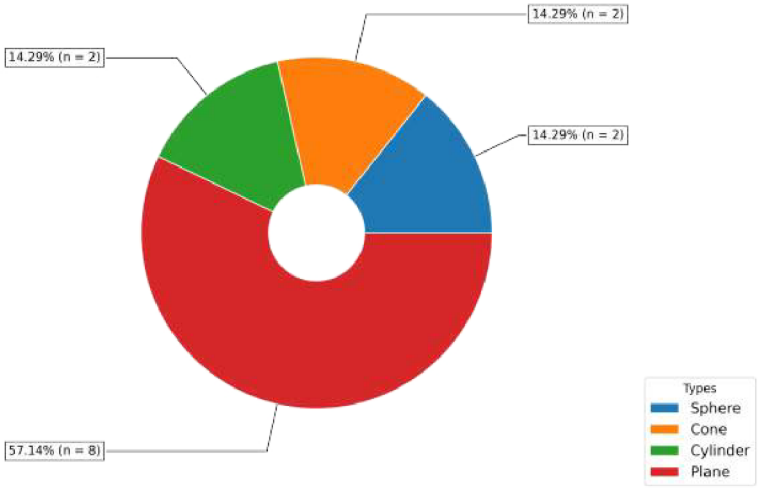


Figura 24: Gráfico do Número de Instâncias por Tipo de Primitiva no Estudo de Caso Simplificado.

Com base na malha triangular, apresentada na Figura 25a, o processo de simulação de LiDAR por *Ray Casting* pode ser aplicado para gerar a nuvem de pontos da estrutura. Entretanto, a relação dos pontos com o triângulo sobre o qual cada um foi gerado é necessária e é mantida no processo de geração de pontos. Dessa forma, essa relação está representada na Figura 25b em conjunto com a nuvem de pontos gerada, onde o identificador do triângulo é representado por uma cor aleatória, dividindo a nuvem de pontos em segmentos triangulares. Com base nessa relação e na informação de correspondência entre os triângulos da malha e as entidades geométricas, vinda do arquivo de geometrias, é possível relacionar cada ponto a uma entidade geométrica, o que pode ser visto na Figura 25c, que atribui uma cor aleatória para cada entidade de geometria do modelo original. Finalizando, o arquivo de geometrias armazena o tipo de primitiva para cada entidade, o que está representado na Figura 25d.

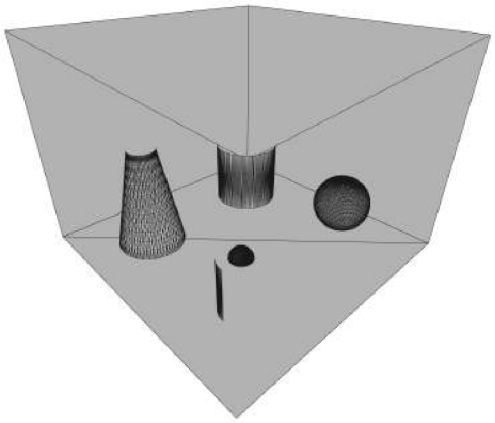
Validando quantitativamente a conformidade entre os pontos e os parâmetros das geometrias, a Figura 26 apresenta três gráficos *Box Plot* que demonstram duas informações para cada tipo de primitiva, são elas:

- **Erro de Distância:** demonstra a distribuição de distâncias por geometria entre os seus pontos e a sua superfície, em metros.

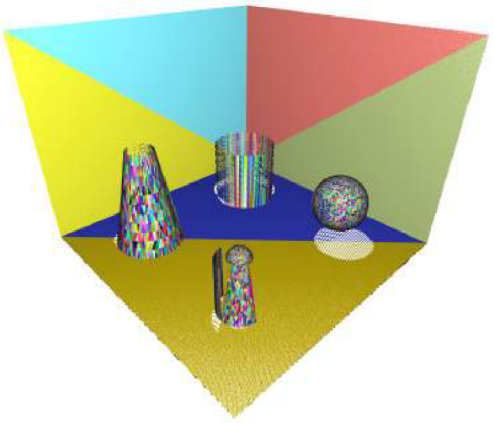
- **Erro de Normal:** demonstra uma distribuição de desvio angular por geometria entre o vetor normal de seus pontos e o vetor normal da sua superfície naquele ponto, em graus.

Então, é possível também observar as diferenças entre as três versões. Na primeira (Fig. 26a) está a versão sem o uso do método de *Points Healing* proposto (Sec. 3.4), na qual o erro na de distâncias pode chegar a 3 cm e o de normal a 1,25° sem que se adicione nenhum ruído, apenas por erro associado a tesselação da malha triangular. Logo após, o segundo gráfico (Fig. 26b) demonstra a efetividade da aplicação do *Points Healing*, visto que o erro de distância diminui para a escala de 1×10^{-8} e o de normal para 0,12° nas superfícies de maior curvatura. Ao fim, o terceiro gráfico (Fig. 26b) apresenta os dados de erro após a adição artificial de ruído uniforme, onde se utiliza $\pm$0,1 m como limite do ruído nas coordenadas dos pontos (erro médio de 0,05 m por instância) e se utiliza $\pm 10°$ como limite de ruído angular para as normais (erro médio de 5° por instância).

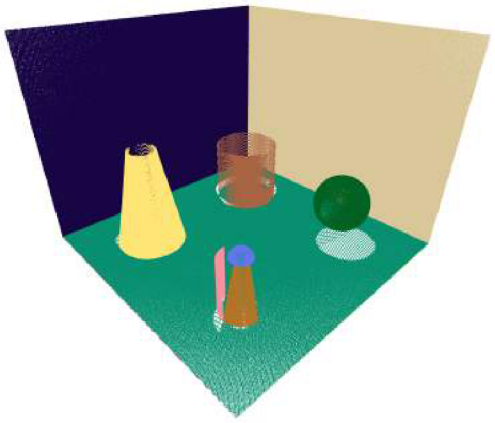
Com base em tudo o que foi demonstrado e discutido, é seguro afirmar que a metodologia proposta é capaz de gerar dados funcionais para treinamento e validação de modelos não somente para a tarefa de Ajuste de Primitivas Geométricas a Nuvens de Pontos, como outras no mesmo contexto. Essa afirmação é embasada na qualidade e verosimilhança entre as nuvens de pontos do *dataset* com nuvens de pontos reais e na correlação comprovadamente existente e ajustadas ente os pontos e as entidades geométricas do modelo CAD original.



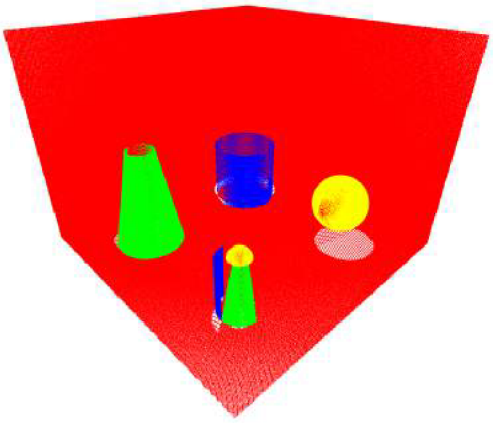
(a) Malha Triangular a ser usada como representação parcial para se gerar nuvens de pontos.



(b) Nuvem de Pontos Gerada com uma Cor Aleatória Representando cada Triângulo da Malha.

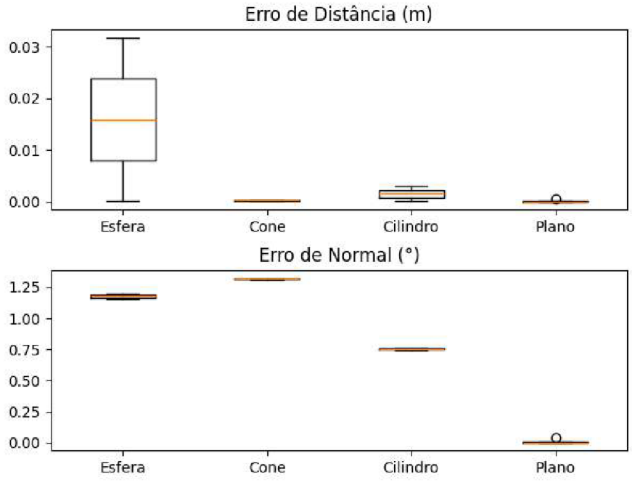


(c) Nuvem de Pontos Gerada com uma Cor Aleatória Representando cada Instância Geométrica.

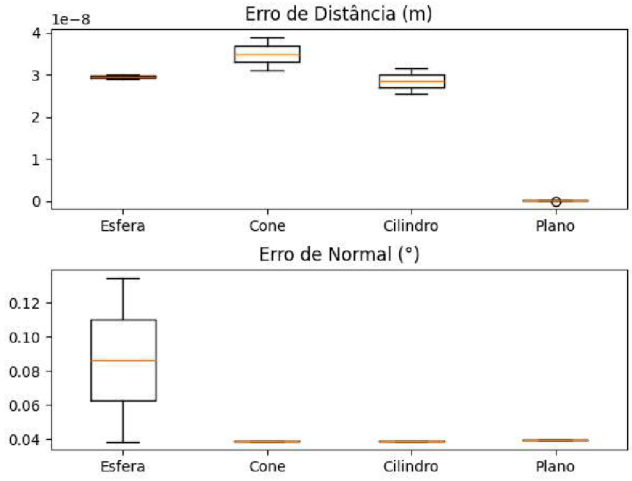


(d) Nuvem de Pontos Gerada com uma Cor Aleatória Representando cada Tipo de Primitiva Geométrica. Os tipos possíveis são: Plano (●), Cilindro (●), Cone (●) e Esfera (●).

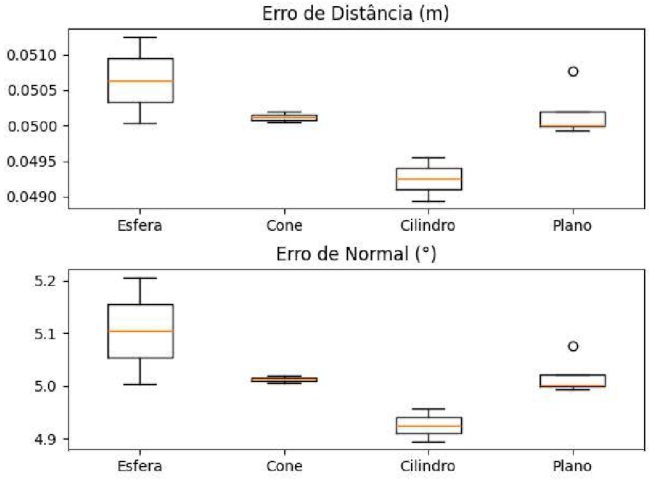
Figura 25: Nuvens de Ponto Gerada no Estudo de Caso Simplificado com Cores Representando Diferentes Informações do *Dataset*. Em **(a)**, apenas uma cor baseada na altura de cada ponto, em **(b)**, a cor representa o identificador do triângulo, em **(c)**, representa a instânica geométrica e em **(d)**, o tipo de primitiva geométrica.



(a) Erros associados a não aplicação do *Points Healing*.



(b) Erros após a aplicação do *Points Healing*.



(c) Erros após a adição de ruído uniforme de $\pm 0{,}1$ m nas coordenadas dos pontos e de $\pm 10°$ em suas normais.

Figura 26: Gráficos *Box Plot* da distribuição dos erros de distância e normal ponto-primitiva. Os erros são mostrados por instância geométrica.

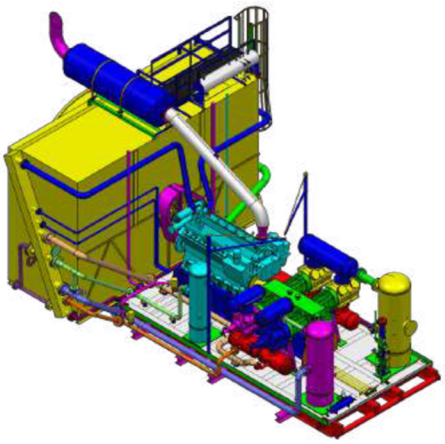
5.2 *Dataset* de Larga-escala de Modelos CAD 3D (LS3DS *Dataset*)

O *Dataset* LS3DS foi gerado aplicando a metodologia de geração de dados do trabalho (Capítulo 3). Os modelos CAD coletados são de contexto industrial, buscando maior relação com o projeto AeroScan, e foram divididos em privados e públicos. Os privados são aqueles fornecidos pela empresa associada ao projeto AeroScan ou de outras fontes que não permitem que o arquivo seja compartilhado. Já os públicos, são aqueles coletados em plataformas abertas da *web*, como o GrabCAD[1], *blogspot Library 3D Engineer* [2] e *CGTrader* [3]. Para a montagem do *dataset*, cerca de 300 modelos CAD foram catalogados e avaliados com relação a quantidade de geometrias, volume em m^3 e análise visual. Nesse contexto, modelos com volume inferior a $20\ m^3$ e com poucas geometrias presentes foram removidos. Além disso, a análise visual permitiu ponderar sobre a qualidade e nível de detalhamento de cada desenho 3D, removendo aqueles com baixo realismo e aqueles que passaram por muitas conversões de formato e acabaram perdendo as geometrias originais no processo. Ao fim do processo de seleção, 86 modelos restaram para formar o *Dataset* LS3DS, dos quais, 80 são públicos e 6 são privados.

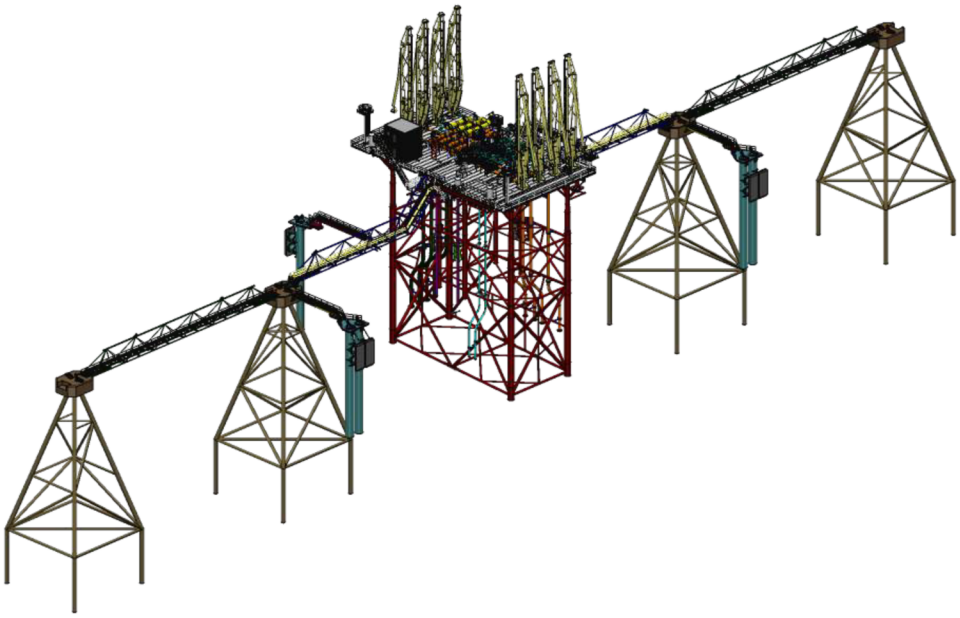
Apesar da filtragem dos modelos, a escala dos arquivos mantido no *dataset* varia substancialmente. É complexo definir de maneira exata qual modelo é de larga-escala e qual não é, contudo, visto que a metodologia de ajuste de primitivas geométricas do trabalho (Cap. 4) prevê a divisão dos modelos grandes em partes menores, a tolerância com relação ao tamanho do modelo pode ser maior, visto que os modelos serão divididos em caixas de tamanhos fixos de qualquer forma. Usando a área do plano de base dos modelos para ilustrar a diversidade de escala, o *dataset* conta com modelos que partem de $5\ m^2$ de área de base até modelos com cerca de $2000\ m^2$, esses dois extremos podem ser visualizados na Figura 27. Apesar da pequena escala, é possível observar alto nível de detalhamento do modelo apresentado na Figura 27a, no qual há uma coerência da disposição e interconexão das diferentes partes mecânicas e uma alta densidade de geometrias de múltiplos tipos. Porém, se modelos menores não forem de interesse à trabalhos futuros que forem usar o LS3DS, as dimensões da caixa delimitadora do modelo, estão disponíveis em um arquivo de metadados. Dessa forma, há a possibilidade de calcular o volume total ou a área do plano de base para passar mais uma camada de filtro e remover os modelos de menor escala, caso seja necessário.

Os modelos são diversos com relação ao tipo de geometrias que os compõe. Apesar de o presente trabalho se propor a identificar apenas superfícies geométricas, o *dataset* conta também com as curvas, visando oferecer maior extensibilidade para trabalhos futuros. Analisando a distribuição das geometrias por tipo, mais especificamente as superfícies, que são apresentadas na Figura 28b, é possível observar que os planos e cilindros são, por

[1] GrabCAD *Link*: https://grabcad.com/
[2] Library 3D Engineer Link: http://library3dengineer.blogspot.com
[3] CGTrader Link: https://www.cgtrader.com



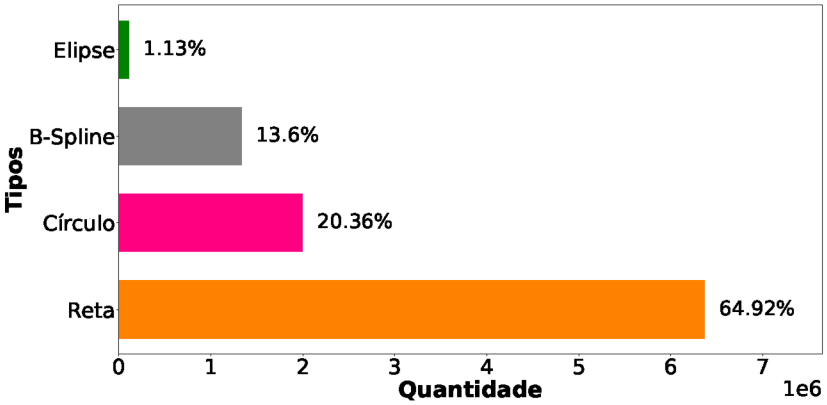
(a) Um dos modelos com menor área de base.



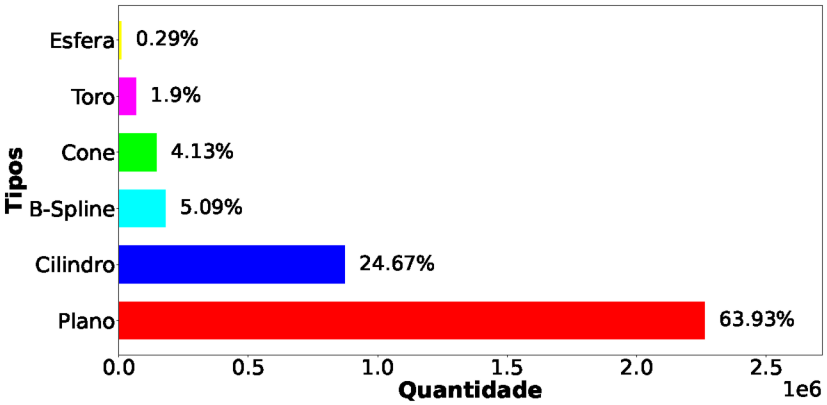
(b) Um dos modelos com maior área de base.

Figura 27: Multiplicidade de escalas dos modelos do *Dataset LS3DS*. Em **(a)** está demonstrado o modelo com menor área de base, cerca de $5\ m^2$. Já em **(b)** é possível observar o modelo com maior área de base, cerca de $2000\ m^2$.

uma larga margem, os tipos mais frequentes no *dataset*, totalizando $88,6\%$. Seguindo a ordem decrescente de frequência, as B-Splines aparecem em terceiro, logo após, se tem os cones, toros e esferas, fechando os $11,4\%$ faltantes de todas as entidades geométricas do *dataset*. Apesar da alta frequência das B-Splines, estas são um caso a parte, visto que as mesmas não são formas geométricas básicas, elas são uma formulação proposta por De Boor [13] que define a superfície a partir de uma série de pontos de controle, o que é diferente das outras formas comentadas, que podem ser definidas por um conjunto de poucos parâmetros previamente estabelecidos. Focando apenas nas primitivas geométricas, que é o foco do trabalho, se tem cerca de 95% das superfícies geométricas que compõe uma estrutura de larga-escala sendo consideradas. Já com relação as curvas, a disposição segue o mesmo padrão estabelecido pelas superfícies, com as Linhas e Círculos sendo as geometrias mais frequentes, seguido pelas BSplines e Elipses, como apresentado na Figura 28a.



(a) Distribuição do número de curvas por tipo ao longo dos modelos do *Dataset*.



(b) Distribuição do número de superfícies por tipo ao longo dos modelos do *Dataset*.

Figura 28: Quantidade de entidades Geométricas de cada tipo no *Dataset* LS3DS.

Apesar do número de entidades geométricas e da área de base serem parâmetros im-

portantes para embasar a discussão sobre as características do *dataset*, somente esses dois valores não são suficientes para descreves os dados. Visto que, seja por amostragem ou por simulação com *Ray Casting*, os pontos sempre são gerados na superfície dos modelos CAD. Dessa forma, medir a área da superfície por tipo demonstra uma distribuição dos dados que valoriza o tamanho das geometrias e, dessa forma, a quantidade de pontos que ela deve receber no processo de geração. Na Figura 29, o gráfico detalha a área média de cada tipo de superfície ao longo dos 86 modelos do *Dataset* LS3DS. Como pode ser observado, os planos ainda são a superfície geométrica mais frequentes e os cilindros tiveram um aumento de aproximadamente 5%. Outra característica interessante é expressividade menor das B-Splines no gráfico por área. Além disso, os toros passam a ser mais frequentes, o que pode ser explicado pelos joelhos presentes nas longas linhas de tubulação presentes nos modelos industriais. Em geral, as primitivas geométricas representam $99,2\%$ da área de superfície dos modelos CAD do *Dataset* LS3DS, o que é ainda superior aos 95% anteriormente levantados, fortalecendo o embasamento na metodologia adotada.

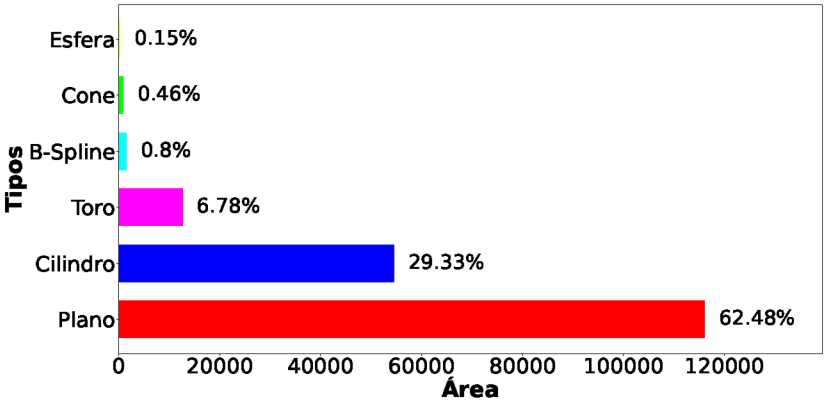


Figura 29: Distribuição da área das superfícies por tipo ao longo dos modelos do *Dataset*.

5.2.1 Nuvens de Pontos e *Ground Truth* para Treinamento e Validação

Os 86 modelos do *Dataset* LS3DS foram divididos em conjunto de treino e validação, onde, o primeiro é utilizado apenas para treinar os métodos baseados em aprendizado e o segundo, que nunca é visto durante o treinamento, é utilizado para calcular métricas e comparar diferentes abordagens que buscam resolver o problema do trabalho. Essa divisão seguiu uma regra de 70/30, respectivamente, onde 61 modelos integram o conjunto de treinamento e os 25 restantes integram o de validação.

Como já justificado no Capítulo 3, o método de triangulação de pontos utilizado no presente trabalho é o que está disponível na biblioteca OCCT [51]. Contudo, principalmente com a utilização do processo de *Points Healing*, a qualidade da nuvem de pontos gerada pouco depende da malha triangular utilizada como base. Nesse contexto, a abor-

dagem de Simulação e LiDAR através de *Ray Casting* foi utilizada para gerar os pontos, a qual permite a produção de nuvens mais realistas e que reduzam a lacuna entre os dados sintéticos e os dados reais. Embora haja a possibilidade de se introduzir ruído artificialmente, as nuvens do *dataset* não contam com nenhum tipo perturbação, permitindo ao usuário das nuvens decidir qual a intensidade de ruído se deseja, seja nas coordenadas dos pontos ou nas normais.

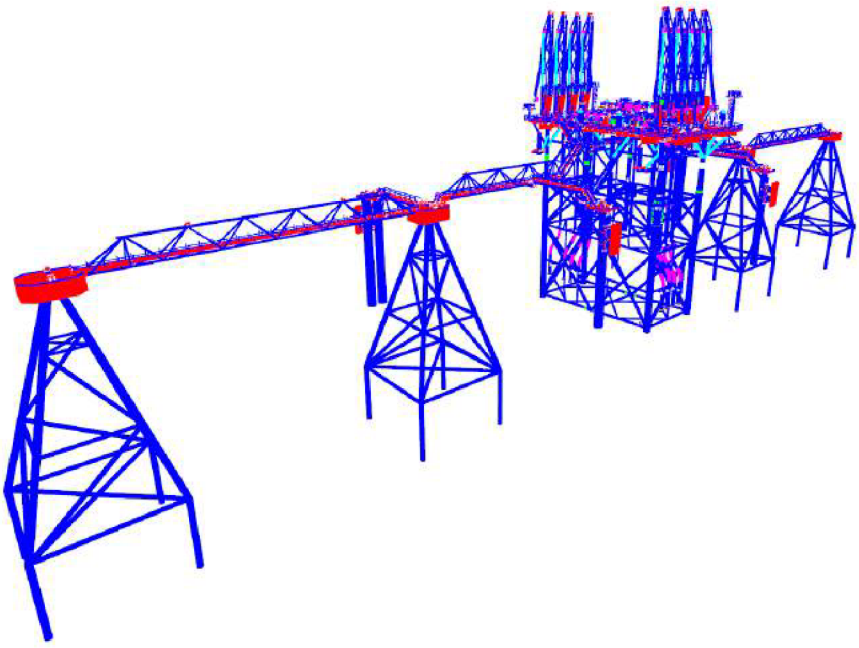
Com relação aos tipos de geometria, apenas as superfícies geométricas estão relacionadas aos pontos, visto que o reconhecimento destas é parte fundamental do objetivo do trabalho. Nesse escopo, todos as geometrias de todos os tipos de superfícies presentes no *dataset* foram mantidas no *ground truth* e na relação com a nuvem, são eles: plano, esfera, cilindro, cone, toro e b-spline. Todavia, os códigos de geração do *dataset* permitem a filtragem das geometrias para manter apenas os tipos desejados, processo que será feito para os experimentos da Seção 5.3, que vão utilizar apenas os quatro primeiros da lista.

Por serem modelos de larga-escala, o *Dataset* LS3DS tem uma média de 4,35 milhões de pontos por modelo, porém, há uma variabilidade considerável, visto que há modelos com cerca de 1 milhão de pontos (Fig. 27a) até modelos com aproximadamente 70 milhões (27b). A Figura 30 ilustra as nuvens do *Dataset* LS3DS apresentando a nuvem de pontos com *ground truth* para segmentação de instâncias geométricas do maior modelo do mesmo'.

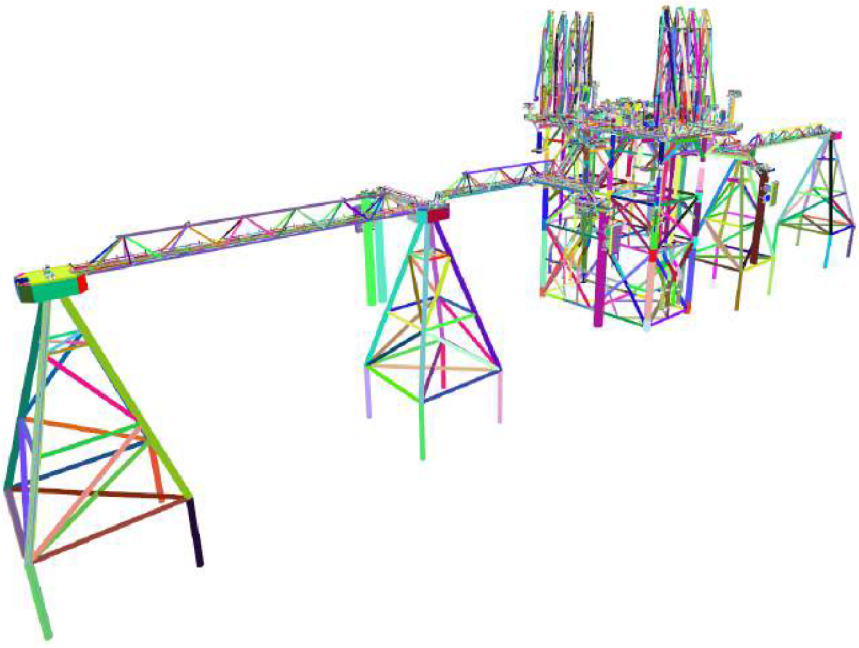
5.3 *Benchmark* de Ajuste de Primitivas Geométricas a Nuvens de Pontos de Larga-Escala

Os experimentos conduzidos nesta seção tem como objetivo explorar detalhes e validar o funcionamento de metodologia SMGF, a qual foi proposta no Capítulo 4. A principal base que fundamenta essa análise é o *Dataset* LS3DS, o qual já conta com as nuvens de pontos e *ground truth* de primitivas geométricas, permitindo o treinamento das diferentes variações da metodologia SMGF e a comparação, a partir de métricas quantitativas, das abordagens propostas. Dentro desse escopo, como já estabelecido no Capítulo 4, cinco soluções serão validadas nesses experimentos, as quais estão definidos na Tabela 1. Por se tratar do único método não baseado em aprendizado e de já ser aplicado em nuvens de pontos de larga-escala na literatura, o método ERANSAC [53] será a base de referência para avaliar as soluções que utilizam a metodologia SMGF. Junto a isso, é importante ressaltar que os métodos serão treinados e avaliados para ajustarem apenas planos, esferas, cilindros e cones, dado que esse é o conjunto que intersecciona todas as abordagens, como já foi justificado na Seção 4.3.

Remetendo novamente ao processo de divisão em partes da SMGF, é importante detalhar que, ao se aplicar a divisão em partes em todos os 86 modelos do *Dataset* LS3DS, novas versões do *dataset* serão criadas, as quais, terão modelos de tamanho controlado



(a) Nuvem de pontos segmentada por tipo de geometria. Os tipos possíveis são: Plano (●), Cilindro (●), Cone (●), Esfera (●), Toro (●) e B-Spline (●).



(b) Nuvem de pontos segmentada por instâncias geométricas, a cor de cada ponto identifica a geometria ao qual ele pertence.

Figura 30: Nuvem de pontos do maior modelo do *Dataset* LS3DS com o *ground truth* para segmentação de tipos geométricos e de instâncias Geométricas.

Método	SMGF	N° Pontos das Partes	Baseado em Aprendizado	Referência
ERANSAC	×	$\approx \infty$	×	[53]
SMGF-ERANSAC	✓	$\approx \infty$	×	[53]
SMGF-ParseNet	✓	$\approx 10^4$	✓	[55]
SMGF-HPNet	✓	$\approx 10^4$	✓	[63]
SMGF-PrimitiveNet	✓	$\approx 2,5 \times 10^5$	✓	[24]

Tabela 1: Descrição dos Métodos a serem comparados nos experimentos.

pela definição da dimensão dos cuboides da divisão. Para além disso, os métodos baseados em aprendizado que foram adaptados para realizar o ajuste de geometrias nas partes divididas tem diferentes escalabilidades com relação a quantidade de pontos, o que justifica a utilização de um processo de controle da quantidade de pontos de cada parte gerada. Logo, a partir desses dois parâmetros (dimensão dos cuboides e número de pontos de cada parte), algumas configurações diferentes de *dataset* pós-divisão foram propostas, as quais estão nomeadas e detalhadas na Tabela 2. A necessidade de cada um desses *datasets* ficará mais esclarecida no decorrer dos experimentos, onde todas as análises feitas justificarão a existência dessas diferentes configurações.

Dataset	Dimensão dos Cuboides	N° Pontos das Partes	N° Modelos de Treino	N° Modelos de Validação
LS3DS-2-7K-Div	2×2×2	7K	69.626	6.908
LS3DS-4-7K-Div	4×4×4	7K	8.731	2.753
LS3DS-4-14K-Div	4×4×4	14K	8.731	2.396
LS3DS-4-28K-Div	4×4×4	28K	8.731	1.764
LS3DS-8-7K-Div	8×8×8	7K	1.124	755

Tabela 2: Características dos *datasets* gerados utilizando a metodologia SMGF com diferentes parâmetros no *Dataset* LS3DS.

Como se pode observar na Tabela 2, o nome de cada *dataset* é composto por 4 partes separadas por traço, são elas: nome do *dataset* base (LS3DS), dimensão dos cuboides, número de pontos e o estado dos dados (”Div” para divididos em partes e ”Com” para completos). Além disso, é possível perceber que o número de pontos sempre parte de 7K, o que é justificado pelo fato de que essa é a quantidade de pontos utilizada pela HPNet nos experimentos conduzidos pelos autores, sendo então, a quantidade base aqui utilizada. Junto a isso, outra característica que chama atenção é a diminuição do número de modelos de validação nos vários *datasets* divididos por cuboides de tamanho 4, o que pode ser explicado pelo fato de que apenas as partes com número de pontos maior ou igual ao utilizado no *dataset* são mantidas, visto que a ParSeNet e a HPNet utilizam sempre

nuvens com uma mesma quantidade de pontos em seus processo de treinamento. Já no número de modelos de treino, essa peculiaridade não se repete, visto que os cuboides tem centroide aleatório e, caso caiam em uma região com quantidade de pontos menor que o mínimo, um novo centroide é sorteado até que se ache uma região adequada.

Ainda com relação aos *datasets*, vale ressaltar que a versão original do LS3DS foi filtrada com o objetivo de manter apenas as geometrias dos tipos a serem estudados nos experimentos desta seção. Somado a isso, alguns pós-processamentos adicionais do processo de geração de dados foram aplicados nos *datasets* listados na Tabela 2. Contudo, fora do processo de *Points Healing*, que já é aplicado por padrão nas nuvens do LS3DS, todos os outros pós-processamentos dos dados foram aplicados diretamente nos modelos das partes e não nos modelos completos antes de serem divididos. Dessa forma, não somente as nuvens de cada um dos milhares de modelos dos *datasets* já comentados sofreram centralização, alinhamento e reescala, como também os parâmetros das geometrias do *ground truth*. Dentre esses processos, o mais relevante de ser comentado é a reescala, visto que a partir dela, cada modelo passa a enquadrar-se dentro de um cubo de dimensão $1 \times 1 \times 1$, fazendo com que a aplicação de ruído artificial possa ser dada em uma escala de 0 à 1, a qual é relativa ao tamanho do modelo. Logo, ao se aplicar um ruído nas coordenadas dos pontos de 0.01, estará se aplicando um ruído de 1% com relação ao tamanho dos modelos, o que facilita algumas análises a serem feitas. Ainda nesse escopo, se optou por inserir ruído de ± 0.01 nas coordenadas e de ± 3 nas normais, assim como proposto por Li and Feng [31].

Diversas validações serão realizadas com base nos *datasets* propostos, logo o diagrama da Figura 19, apresentado no Capítulo 4, já evidenciava duas formas diferentes de validar os resultados. A primeira, denominada ”Validação nas Partes”, realiza a comparação entre os métodos ao rodarem nos conjuntos menores de pontos das partes divididas. Assim, apenas os métodos que implementam algum processo de divisão, ou a metodologia SMGF nesse contexto, podem participar dessa análise. Já a segunda e mais importante para o contexto do presente trabalho, é denominada de ”Validação Completa”, a qual é feita sob os modelos completos. Entrentato, é lógico afirmar que ao se dividir e reduzir a quantidade de pontos de cada parte há uma perda não somente na quantidade de pontos total, como também uma perda na quantidade de geometrias. Logo, ao se executar a Validação Completa de um método que utiliza a metodologia SMGF frente ao ERANSAC, que é executado diretamente nos modelos completos, o mesmo conjunto de pontos e de geometrias deve ser utilizado para que a comparação possa ser feita de maneira absoluta. Para isso, foi feita a agregação de cada um dos *datasets* da Tabela 2, produzindo suas versões completas que contam com a mesma perda de pontos e geometrias que suas partes e podem ser utilizadas para executar o ERANSAC diretamente. Logo, nas análises a seguir não será utilizado o modificador ”Div”ou ”Com”para nomear os *datasets*, já que, caso se esteja fazendo Validação nas Partes, o *dataset* ”Div”será utilizado, já caso for

Validação Completa, o *dataset* "Com"será a referência.

Ainda no escopo das validações, algumas métricas serão utilizadas para comparar quantitativamente as soluções propostas. Essas métricas foram majoritariamente propostas por Li and Feng [31] e buscam avaliar tanto a qualidade da segmentação quanto a qualidade do ajuste de parâmetros das geometrias. Junto a isso, cada configuração de *dataset* também será avaliada por métricas simples que demonstrem a perda de informação total do processo completo feito pela metodologia SMGF, visto que nos processos de dividir em partes e controlar a quantidade de pontos de cada parte, se perde pontos e geometrias. As métricas a serem utilizadas são:

Métricas dos *Datasets*. As métricas aqui propostas são extremamente simples e buscam avaliar a taxa de perda de geometrias e a taxa de perda de pontos das configurações propostas com relação aos dados originais e completos do *dataset* LS3DS. São elas:

- **Percentual de Perda de Pontos (PLR):** mede o percentual de perda de pontos do processo divisão e controle da quantidade de pontos em cada parte. Pode ser calculada a partir do complemento da divisão do número total de pontos de um *dataset* pelo número total de pontos do *dataset* LS3DS original.
- **Percentual de Perda de Geometrias (GLR):** mede o percentual de perda de geometrias do processo divisão e controle da quantidade de pontos em cada parte. Pode ser calculada a partir do complemento da divisão do número total de geometrias de um *dataset* pelo número total de geometrias do *dataset* LS3DS original.

Métricas dos Métodos. Para avaliar as diferentes soluções com relação à base de referência, métricas largamente adotadas na literatura de pequena escala serão aplicadas [31, 55, 63]. São elas:

- **Intersecção da União de Segmentação (SIoU):** mede o quanto, na média, o algoritmo foi capaz de segmentar adequadamente as instâncias geométricas no *dataset*. Para isso, é feito uma média entre todos os modelos, sejam eles partes ou completos, do mIoU de cada um. A fórmula do SIoU está disposta na Equação 4 em conjunto com as equações parciais que são usadas para calcular a métrica.

$$
\begin{aligned}
SIoU &= \frac{1}{M}\sum_{i=1}^{M} mIoU_i \\
mIoU_i &= \frac{1}{K}\sum_{j=1}^{K} IoU_{i,j} \\
IoU_{i,j} &= \frac{\hat{L}_{i,j} \cap L_{i,j}}{\hat{L}_{i,j} \cup L_{i,j}}
\end{aligned}
\tag{4}
$$

A equação mais básica da métrica calcula o $IoU_{i,j}$ para cada instância (j) de cada modelo do *dataset* (i). Esse valor pode ser calculado a partir da divisão entre

a intersecção e a união dos índices dos pontos que fazem parte de determinada instância nos resultados preditos ($\hat{L}$) e os índices da mesma no *ground truth* (L). Logo após, a $mIoU$ faz a média do IoU entre todas as K geometrias de um modelo do *dataset*. Finalizando, a $SIoU$ é calculada fazendo a média geral do $mIoU$ de todos os M modelos o *dataset*.

Essa métrica é a mais importante no sub-problema de segmentação de geometrias, onde a mesma se dá no intervalo de 0 à 1, sendo, no menor valor, uma segmentação totalmente errada e, no maior, uma segmentação perfeita.

- **Acurácia de Tipos (TypeAcc):** estima a qualidade do método em atribuir um tipo de geometria para as instâncias preditas. Nesse contexto, a fórmula que calcula essa métrica está definida na Equação 5, na qual o tipo predito para uma geometria ($\hat{t}_{i,j}$) é comparado ao tipo de geometria dessa instância no *ground truth* ($t_{i,j}$). Se forem iguais a função indicadora $\mathbb{1}(\cdot)$ retorna valor 1, porém, caso sejam diferentes, o valor retornado é 0. Logo após, é feita a média entre as K geometrias de cada modelo e, em seguida, entre os M modelos do *dataset*.

$$TypeAcc = \frac{1}{M}\sum_{i=1}^{M}\frac{1}{K}\sum_{j=1}^{K}\mathbb{1}(t_{i,j} = \hat{t}_{i,j}) \tag{5}$$

A métrica tem intervalo definido definido de 0 à 1, onde quanto mais próximo do maior valor o método se posiciona, maior é seu acerto com relação ao tipo das instâncias geométricas preditas.

- **Erro Residual (Res):** calcula o erro de distância residual entre as geometrias ajustadas e os pontos da nuvem. Apesar de também estar relacionado indiretamente com a qualidade da segmentação de instâncias, o Erro Residual é uma métrica muito importante para mensurar o quão bem estimados foram os parâmetros da primitiva geométrica predita. Para isso, em cada geometria predita, cada ponto que a integra tem a sua distância medida até ela. Logo após, a média de distâncias entre todos os pontos de uma geometria é feita, para que assim se possa concluir o cálculo da métrica realizando a média entre todas as geometrias de um modelo e finalizar com a média entre todos os modelos, o que está formulado na Equação 6.

$$Res = \frac{1}{M}\sum_{i=1}^{M}\frac{1}{K}\sum_{j=1}^{K}D_{\hat{t}_{i,j}}(P_{i,j}, \hat{A}_{i,j}) \tag{6}$$

Seguindo o mesmo padrão das outras métricas, é feita uma média entre as K geometrias de cada modelo e logo após, uma média entre todos os M modelos. Entretanto, previamente, a função de distância $D(\cdot)$, associada ao tipo de geometria predito $\hat{t}$,

é utilizada para calcular o erro residual entre o conjunto de pontos pertencentes a aquela geometria $P_{i,j}$ e a geometria de fato. Para isso, a função de distância $D(\cdot)$ é parametrizada pelo conjunto de parâmetros que foram ajustados pelo método para cada uma das geometrias ($\hat{A}_{i,j}$). Nesse escopo, é importante chamar atenção ao fato de que o conjunto de pontos sendo utilizado não é vinculado ao que foi predito, dessa forma, os pontos a serem utilizados nesse cálculo são os do *ground truth*, ou seja, os pontos que realmente deveriam ser da geometria em questão.

A métrica mede um valor de distância, dessa forma, a unidade da nuvem de pontos é a referência para a unidade da métrica. Posto isso, é importante salientar que a métrica tem uma relação intrínseca com a normalização de reescala, visto que, como esta será utilizada, toda a nuvem de pontos irá preencher um cubo de $1 \times 1 \times 1$ e a métrica se dará principalmente no intervalo de 0 à 1, a tornando relativa ao tamanho do modelo, fazendo com que o erro residual também possa ser visto como um percentual.

- **Taxa de Cobertura de Pontos (PCov):** determina o percentual médio de pontos que estão menos distantes de suas geometrias que um limiar ϵ determinado. Dessa forma, a métrica, apresentada na Equação 7, é muito similar ao Erro Residual, porém, com a adição da função indicadora $\mathbb{1}(\cdot)$ e da operação de comparação ($<$) com o ϵ. Assim, apenas aqueles pontos que tem distância abaixo do limiar serão contabilizados positivamente na métrica. Outro fator importante é que essa métrica tem como objetivo compreender o quão bem as geometrias encontradas cobrem os pontos, sem que nem o *ground truth* seja necessário para calcula-la, visto que todas as variáveis utilizadas são preditas. inclusive os pontos $\hat{P_{i,j}}$, diferentemente do Erro Residual.

$$PCov = \frac{1}{M}\sum_{i=1}^{M}\frac{1}{K}\sum_{j=1}^{K}\mathbb{1}(D_{\hat{t}_{i,j}}(\hat{P_{i,j}}, \hat{A}_{i,j}) < \epsilon) \tag{7}$$

A métrica mede um valor de 0 à 1, indicando o percentual de pontos cobertos pelas geometrias estimadas dentro de uma tolerância parametrizável.

Tendo as diferentes versões dos *datasets* propostas (Tab. 2) e as métricas para avaliar as perdas associadas ao mesmos definidas, a Tabela 3 detalha a Percentual de Perda de Pontos (PLR) e a Percentual de Perda de Geometrias (GLR) no conjunto de validação de cada uma das configurações. É importante pontuar que, cada configuração tem um propósito específico de avaliação dentro dos experimentos, contudo, ter um alto PLR e GLR é, intrinsecamente, um fator ruim associado ao *dataset* e ao método que for treinado nele. Outro fator relevante a ser comentado, é que em todas as configurações, inclusive no *dataset* completo original, geometrias com menos do que 100 pontos são removidas

e seus pontos são tratados como não pertencentes a nenhuma geometria, estratégia que é adotada por Yan et al. [63] e seguida no presente trabalho.

Dataset	↓ PLR	↓ GLR
LS3DS-2-7K	64,57%	61,45%
LS3DS-4-7K	85,88%	79,83%
LS3DS-4-14K	75,43%	70,30%
LS3DS-4-28K	**63,82%**	**60.18%**
LS3DS-8-7K	96,13%	91,20%

Tabela 3: Percentual de Perda de Pontos (PLR) e Percentual de Perda de Geometrias (GLR) de cada um dos *datasets* gerados para os experimentos.

Independente do *dataset* a ser usado, os métodos baseados em aprendizado precisam ser treinados, logo, anterior aos experimentos. Posto isso, a configuração de *dataset* base é a LS3DS-4-7K, assim, o experimento da Seção 5.3.1 comprovará o funcionamento da metodologia SMGF e deixará clara a capacidade do *dataset* LS3DS de servir como base para solução do problema do trabalho. Logo após, com o funcionamento comprovado, se fará a exploração dos detalhes da metodologia proposta. Primeiramente, no experimento da Seção 5.3.2, as consequências da diminuição ou aumento da dimensão das partes dos modelos completos serão discutidas. Seguindo, o experimento da Seção 5.3.3 demonstra as limitações associadas ao aumento da quantidade de pontos de cada parte, enfatizando o impacto nos resultados dos métodos de ajuste de primitivas e o crescimento acelerado do consumo de memória da GPU associado a esse processo em alguns casos. Ao fim, a Seção 5.3.4 faz um apanhado geral das conclusões obtidas a partir dos experimentos.

5.3.1 Experimento 1: validação de funcionamento

O presente experimento tem como objetivo demonstrar o funcionamento das soluções propostas de maneira quantitativa. Para isso, o *dataset* LS3DS-4-7K será utilizado, visto que a dimensão dos cuboides é a mediana entre as outras duas (2 e 8) e a quantidade de pontos é suportada diretamente por todas as abordagens de ajuste de primitivas geométricas nas partes.

Como primeiro passo, os métodos SMGF-ParseNet, SMGF-HPNet e SMGF-PrimitiveNet devem ser treinados nos 8.731 modelos do *dataset*. Após o fim do processo de treino, os métodos foram executados nos 2.753 modelos de validação do *dataset*, gerando tanto a segmentação em instâncias geométricas de cada modelo como também os parâmetros das primitivas. Logo, a partir desses resultados, as métricas foram calculadas e seus valores estão apresentados na Tabela 4.

O resultado de Validação nas Partes desse experimento apenas demonstra a eficiência dos métodos com relação as métricas, alcançando bons resultados de segmentação (SIoU e TypeAcc) e bons resultados de ajuste de parâmetros (Res e PCov). Além disso, algu-

Validação nas Partes - LS3DS-4-7K

Método	↑ SIoU	↑ TypeAcc	↓ Res	↑ PCov
SMGF-ERANSAC	**0.8146**	0.7539	**0.0132**	**0.9780**
SMGF-ParseNet	0.7174	0.9676	0.0254	0.9037
SMGF-HPNet	0.7981	**0.9678**	0.0257	0.9132
SMGF-PrimitiveNet	0.5540	0.9102	0.0865	0.6605

Tabela 4: Validação nas Partes no *Dataset* LS3DS-4-7K.

mas características que vão ser mostradas nos próximos experimentos também aparecem neste, que são:

- Alta qualidade de Segmentação de Instâncias do ERANSAC, sendo observada na sua aplicação em conjunto com a metodologia SMGF;
- Métodos baseados em aprendizado determinam melhor os tipos das instâncias.

Apesar da métrica de erro residual dar a entender, esse caso, que o ajuste de parâmetros do ERANSAC é melhor do que a abordagem diferenciável proposta por Li and Feng [31], e que é aplicada nos 3 métodos baseados em aprendizado, os resultados dessa métrica tem alta influência da qualidade da segmentação de instâncias, logo, ainda se tem poucos dados para se constatar isso.

Embora a Validação nas Partes seja importante para algumas análises, é na Validação Completa que realmente se vê como cada variação da metodologia SMGF é capaz de realizar o ajuste de primitivas geométricas a nuvens de pontos de larga-escala. Para isso, os resultados das 2.753 partes são agregados para remontar os 25 modelos de validação originais, processo descrito com mais detalhes na Seção 4.4. Dessa forma, a Tabela 5 apresenta os resultados no *dataset* LS3DS-4-7K para as 4 abordagens baseadas na SMGF e comparadas ao ERANSAC.

Validação Completa - LS3DS-4-7K

Método	↑ SIoU	↑ TypeAcc	↓ Res	↑ PCov
ERANSAC	0.2474	0.4003	0.0412	0.7252
SMGF-ERANSAC	**0.5563**	0.4888	**0.0154**	0.9408
SMGF-ParseNet	0.4636	**0.9115**	0.0190	0.9035
SMGF-HPNet	0.5251	0.9043	0.0174	**0.9592**
SMGF-PrimitiveNet	0.1132	0.7667	0.0973	0.6252

Tabela 5: Validação Completa no *Dataset* LS3DS-4-7K.

Com base na Tabela 5, pode ser observado que o ERANSAC é superado em todas as métricas pelos métodos que usam SMGF, com exceção do SMGF-PrimitiveNet. Para além disso, apesar do SMGF-ERANSAC ainda ser o melhor em segmentação de

instâncias, seus resultados ficam muito próximos do SMGF-HPNet, o qual ainda o supera em PCov, se aproxima muito na métrica Res e é largamente superior na acurácia de tipos. Dessa forma, em geral, o SMGF-HPNet é o métodos que conta com resultados mais relevantes neste experimento.

Embora as abordagens baseadas em aprendizado tenham apresentado bons resultados, o SMGF-PrimitiveNet aparenta ter uma dificuldade adicional de lidar com nuvens de quantidade de pontos muito baixa, como as de 7.000 pontos deste experimento. Esse comportamento era esperado, visto que o método base, a PrimitiveNet, foi proposto para lidar com maiores quantidades de pontos. Portanto, se voltara a estudar essa solução apenas no experimento da Seção 5.3.3, no qual os *datasets* com maior limite de quantidade de pontos serão explorados.

Como conclusões adicionais, o experimento valida mais uma vez não somente a solução para geração de *datasets* (Cap. 3), como também o *Dataset* LS3DS, que são duas contribuições importantes so trabalho. Apesar disso, é importante pontuar que o *dataset* LS3DS-4-7K conta com apenas 20.17% (GLR = 79.83%) das geometrias do LS3DS original. Logo, apesar dos resultados já superarem a base de referência, as problemáticas estabelecidas pelo *dataset* gerado por esse trabalho estão distantes de serem totalmente resolvidas.

5.3.2 Experimento 2: dimensão das partes

Anterior à mudança da quantidade de pontos para diminuir o GLR do *dataset*, o aumento ou diminuição dos cuboides do processo de divisão pode promover o mesmo efeito sem sobrecarregar os métodos de pequena-escala baseados em aprendizado. Dessa forma, os *datasets* LS3DS-2-7K e LS3DS-8-7K foram propostos com esse intuito. Além de aumentar o GLR (Tab. 3), a diminuição do tamanho das partes feita no LS3DS-2-7K também promove um contexto mais reduzido para o algoritmo de ajuste funcionar e também uma resolução maior na nuvem, visto que a mesma quantidade de pontos (7.000) está agora representando superfícies dentro de uma região de menor dimensão, os quais são fatores que podem impactar diretamente na qualidade dos resultados obtidos. Já com relação ao LS3DS-8-7K, o efeito é o oposto, onde ocorre uma diminuição da resolução e um aumento da janela de contexto.

Os resultados da Validação nas Partes dos diferentes métodos avaliados nesse experimento estão expostos na Tabela 6. Na tabela, é possível perceber que os resultados do SMGF-RANSAC pouco mudam nos diferentes *datasets*, demonstrando uma alta robustez do método com relação a nuvens de pontos com baixas resoluções, o que faz sentido, visto que este é um método de essência estocástica e abordagem local. Todavia, os métodos baseados em aprendizado já apresentam um comportamento oposto, onde há significativa progressão em todas as métricas conforme o tamanho das partes diminuem. Logo, este fato aponta na direção de que a utilização de nuvens de pontos mais densas gera uma

maior compreensão dos métodos de aprendizado e, dessa forma, melhores resultados.

Validação nas Partes - 7K Pontos

Método	↑ SIoU	↑ TypeAcc	↓ Res	↑ PCov
LS3DS - 8 - 7K				
SMGF-ERANSAC	**0.7785**	0.7630	**0.0161**	**0.9690**
SMGF-ParseNet	0.6168	0.9508	0.0347	0.8591
SMGF-HPNet	0.6843	**0.9538**	0.0357	0.8815
LS3DS - 4 - 7K				
SMGF-ERANSAC	**0.8146**	0.7539	**0.0132**	**0.9780**
SMGF-ParseNet	0.7174	0.9676	0.0254	0.9037
SMGF-HPNet	0.7981	**0.9678**	0.0257	0.9132
LS3DS - 2 - 7K				
SMGF-ERANSAC	0.7977	0.6835	**0.0135**	**0.9771**
SMGF-ParseNet	0.7699	**0.9752**	0.0179	0.9368
SMGF-HPNet	**0.8349**	0.9729	0.0164	0.9468

Tabela 6: Validação nas Partes dos métodos nos diferentes *datasets* que contam com 7.000 pontos em cada parte.

As mesmas inferências feitas na Validação das Partes também se aplicam ao processo de Validação Completa, que tem seus resultados expostos na Tabela 7. Nesse caso, é possível observar que as soluções utilizando a metodologia SMGF superam a base de referência, que é o ERANSAC, em praticamente todas as métricas de todos os *datasets*. Além disso, os resultados do SMGF-ERANSAC pouco mudam nas diferentes configurações, sendo superiores no LS3DS-8-7K, inferiores em algumas métricas no LS3DS-4-7K e inferiores em todas no LS3DS-2-7K. Com relação a acurácia de tipos (TypeAcc), os métodos baseados em aprendizado superam as soluções baseadas no ERANSAC por uma larga margem, o que já foi demonstrando em todos os resultados até aqui. Contudo, ao ser se fornecer nuvens de maior resolução na entrada dos métodos de aprendizado, eles acabam superando o ERANSAC e o SMGF-ERANSAC até mesmo na segmentação de instâncias (SIoU) e, no caso do SMGF-HPNet, até mesmo no erro residual e na taxa de cobertura de pontos.

Com base nos resultados demonstrados na Tabela 7, é intuitivo afirmar que a melhor configuração nesse contexto é a LS3DS-2-7K, visto que, além de ter os melhores resultados na larga-escala, também é a configuração que perde menos pontos e menos geometrias. Nesse escopo, o melhor método de maneira absoluta em todas as métricas é a SMGF-HPNet, porém, a SMGF-ParseNet tem uma ótima progressão de resultados com o aumento da resolução, alcançando valores muito próximos aos da anterior em praticamente todas as métricas.

Validação Completa - 7K Pontos				
LS3DS - 8 - 7K				
Método	↑ SIoU	↑ TypeAcc	↓ Res	↑ PCov
ERANSAC	0.2789	0.3525	0.0350	0.6419
SMGF-ERANSAC	**0.5045**	0.4362	**0.0284**	**0.9284**
SMGF-ParseNet	0.3736	**0.9074**	0.0360	0.7955
SMGF-HPNet	0.4423	0.8838	0.0312	0.9085
LS3DS - 4 - 7K				
ERANSAC	0.2474	0.4003	0.0412	0.7252
SMGF-ERANSAC	**0.5563**	0.4888	**0.0154**	0.9408
SMGF-ParseNet	0.4636	**0.9115**	0.0190	0.9035
SMGF-HPNet	0.5251	0.9043	0.0174	**0.9592**
LS3DS - 2 - 7K				
ERANSAC	0.2150	0.3627	0.0424	0.6915
SMGF-ERANSAC	0.5528	0.4625	0.0111	0.9614
SMGF-ParseNet	0.5633	0.9229	0.0127	0.9497
SMGF-HPNet	**0.5774**	**0.9233**	**0.0070**	**0.9757**

Tabela 7: Validação Completa dos métodos nos diferentes *datasets* que contam com 7.000 pontos em cada parte.

Com base no que foi discutido, pode-se concluir que o aumento de resolução promove resultados melhores. Nesse contexto, é lógico afirmar que quanto mais se divide a nuvem, maior resolução se obtém nas partes e, assim, melhor se saem os métodos baseados em aprendizado. Contudo, dividir mais as nuvens de entradas também provoca efeitos negativos, como:

- **Redução da Janela de Contexto:** quanto menores são as regiões das partes, menor é janela de contexto que o algoritmo utiliza para aprender quais pontos fazem parte de qual geometria. Um exemplo da influência do contexto nos resultados, nesse caso, poderia ser as linhas de tubulação, onde se ter uma janela de contexto maior permite ao método compreender como os cilindros normalmente se interconectam nelas e produzir resultados melhores devido a isso.

- **Divisão das Geometrias ao longo de muitas Partes:** ao se super dividir a nuvem de pontos, primitivas geométricas podem acabar tendo um fragmento em cada parte ao longo de múltiplas partes. Dessa forma, apesar de seu fragmento continuar sempre sendo uma superfície de mesmo tipo que a geometria completa, este pode acabar sendo ignorado por ser muito pequeno, prejudicando a qualidade dos resultados.

- **Maior Dificuldade no Processo de Agregação:** de forma intrínseca ao processo

de dividir e agregar, quanto mais se divide, mas difícil é a agregação dos resultados depois. Nesse escopo, mesmo que o algoritmo de agregação aqui utilizado utilize o *ground truth* para facilitar o processo e produzir resultados mais consistentes nos modelos completos, ainda assim, a divisão excessiva pode gerar consequências negativas. Ademais, caso um trabalho futuro aplique algum método de agregação que não use o *ground truth* como apoio, o erro desse método seria amplificado, visto que cada divisão adicional geraria uma necessidade de mais uma agregação deste método hipotético.

Dados os problemas associados a dividir demais as nuvens de entradas associados à melhora nos resultados promovida pelo aumento da resolução da nuvem, a possibilidade de aumentar a quantidade de ponto de cada parte emerge como uma possível saída para essa questão. Alguns resultados visuais para o experimento 2 estão apresentados no Anexo A.

5.3.3 Experimento 3: limitações na quantidade de pontos

Com o objetivo de diminuir o GLR e aumentar a resolução das nuvens sem precisar dividir os modelos originais em mais partes, este experimento discute o aumento do número de pontos por parte como uma possibilidade de promover essas melhorias. Para isso, três *datasets* foram propostos, são eles: LS3DS-4-7K, LS3DS-4-14K e LS3DS-4-28K. Logo, os métodos foram treinados e validados em cada uma dessas configurações, gerando resultados importantes a serem discutidas no presente trabalho. Primeiramente, com relação à Validação nas Partes, que tem seus resultados expostos na Tabela 8. Neste escopo, assim como observado no experimento da Seção 5.3.2, o SMGF-ERANSAC não tem uma melhora de seus resultados com o aumento da resolução da nuvem, neste caso, promovida pelo aumento dos pontos, se mantendo com pouca variação em todas as métricas e até piorando seus resultados no *dataset* com maior quantidade de pontos.

Os métodos baseados em aprendizado tem um comportamento um pouco diferente neste experimento, o que pode ser visto pelas métricas apresentadas na Tabela 8. Com respeito aos métodos que tendem a consumir mais memória e receber menos pontos na entrada, SMGF-Parsenet e SMGF-HPNet, houve uma diminuição significativa do tamanho de *batch* conforme se aumentou a quantidade de pontos. Como o *setup* de treinamento envolve a utilização de duas GPUs, o tamanho mínimo de *batch* possível é 2, ou seja, um modelo em cada GPU por vez. Neste contexto, mesmo com cada uma tendo 24 GB de memória de vídeo, utilizando 7.000 pontos um tamanho de *batch* de 12 foi utilizado, já no *dataset* com 14.000 pontos por modelo, o tamanho máximo de *batch* possível foi 6 e no último, com 28.000 pontos, o valor 2 foi utilizando. Apesar disso, a SMGF-HPNet não pode ser treinada na configuração com 28.000 pontos, visto que mesmo com o tamanho de *batch* mínimo o método deu erro de falta de memória disponível no começo do treinamento, o que justifica a não declaração de suas métricas nesse *dataset.* Logo,

Validação nas Partes - Partes de Dimensão $4 \times 4 \times 4$

Método	↑ SIoU	↑ TypeAcc	↓ Res	↑ PCov
LS3DS - 4 - 7K				
SMGF-ERANSAC	**0.8146**	0.7539	**0.0132**	**0.9780**
SMGF-ParseNet	0.7174	0.9676	0.0254	0.9037
SMGF-HPNet	0.7981	**0.9678**	0.0257	0.9132
SMGF-PrimitiveNet	0.5540	0.9102	0.0865	0.6605
LS3DS - 4 - 14K				
SMGF-ERANSAC	**0.8175**	0.7460	**0.0100**	**0.9877**
SMGF-ParseNet	0.7006	0.9606	0.0311	0.8803
SMGF-HPNet	0.2189	**0.9820**	0.0158	0.9385
SMGF-PrimitiveNet	0.5544	0.9187	0.0907	0.6365
LS3DS - 4 - 28K				
SMGF-ERANSAC	**0.7655**	0.6976	**0.0151**	**0.9798**
SMGF-ParseNet	0.6427	**0.9526**	0.0449	0.8287
SMGF-HPNet	×	×	×	×
SMGF-PrimitiveNet	0.4257	0.8557	0.1174	0.5477

Tabela 8: Validação nas Partes dos métodos nos diferentes *datasets* que foram divididos usando cuboides de dimensão $4 \times 4 \times 4$.

o limite superior para a quantidade de pontos por parte nas abordagens SMGF-ParseNet e SMGF-HPNet é algo em torno de 28.000 pontos, o que é um fator altamente limitante desses métodos.

Com relação aos resultados dos métodos baseados em aprendizado, é possível observar que, na Validação nas Partes (Tab. 8) não houve melhora dos resultados com o aumento da quantidade de pontos. No caso específico da SMGF-HPNet, a métrica SIoU desceu a um valor muito baixo no *dataset* LS3DS-4-14K, o que mostra que o método é extremamente sensível ao aumento da quantidade de pontos na entrada, sendo indicado que se mantenha nos 7.000 inicialmente utilizados pelos autores da HPNet [63]. Já com relação a SMGF-ParseNet, os resultados das métricas também pioraram, contudo, num ritmo muito mais lento do que a anterior. Nesse caso, a piora do método está atrelada ao aumento do nível de dificuldade da tarefa, visto que em cada parte há uma quantidade maior de pontos para segmentar em instâncias geométricas e ajustar parâmetros para elas. Finalizando, para o método SMGF-PrimitiveNet se esperava um ganho significativo de desempenho conforme a quantidade de pontos aumentasse, entretanto, isso não aconteceu, já que as métricas se mantiveram estáveis nos dos primeiros *datasets* e tiveram uma diminuição significativa no último.

Na Validação Completa, que tem seus dados expostos na Tabela 9, é possível observar um comportamento parecido onde nenhum dos métodos tem progressão significa-

Validação Completa - Partes de Dimensão $4 \times 4 \times 4$

Método	↑ SIoU	↑ TypeAcc	↓ Res	↑ PCov
LS3DS - 4 - 7K				
ERANSAC	0.2474	0.4003	0.0412	0.7252
SMGF-ERANSAC	**0.5563**	0.4888	**0.0154**	0.9408
SMGF-ParseNet	0.4636	**0.9115**	0.0190	0.9035
SMGF-HPNet	0.5251	0.9043	0.0174	**0.9592**
SMGF-PrimitiveNet	0.1132	0.7667	0.0973	0.6252
LS3DS - 4 - 14K				
ERANSAC	0.2282	0.3790	0.0458	0.6977
SMGF-ERANSAC	**0.5205**	0.4839	**0.0153**	**0.9578**
SMGF-ParseNet	0.4646	**0.9035**	0.0169	0.9342
SMGF-HPNet	×	×	×	×
SMGF-PrimitiveNet	0.1226	0.7217	0.0809	0.6978
LS3DS - 4 - 28K				
ERANSAC	0.2036	0.3681	0.0458	0.7490
SMGF-ERANSAC	**0.4485**	0.4370	0.0247	**0.9242**
SMGF-ParseNet	0.4092	**0.9000**	**0.0234**	0.8527
SMGF-HPNet	×	×	×	×
SMGF-PrimitiveNet	0.0520	0.6821	0.0976	0.6301

Tabela 9: Validação Completa dos métodos nos diferentes *datasets* que foram divididos usando cuboides de dimensão $4 \times 4 \times 4$.

tiva da qualidade dos resultados a partir do aumento do número de pontos. Entretanto, com exceção da SMGF-PrimitiveNet, todas as soluções propostas superam o ERANSAC padrão em todas as métricas de todos os *datasets*. Apesar dos resultados não melhorarem com o aumento de pontos, é importante pontuar que o *dataset* LS3DS-4-28K tem 39,82% da quantidade de geometrias dos modelos originais do LS3DS, enquanto o LS3DS-4-7K tem apenas 14,2% (Tab. 3). Assim, os resultados das métricas são calculados sob um denominador maior nas configurações com mais pontos, o que pode ser entendido como uma estabilidade na qualidade dos resultados, ao invés de uma regressão nos mesmos.

Apesar da limitação apresentada pelos métodos baseados em aprendizado com relação ao número de pontos presentes na entrada, a SMGF-PrimitiveNet não apresentou essa restrição, podendo ser treinada com tamnhos de *batch* superiores a 64 em todos os contextos do experimento. Contudo, apesar do aprendizado convergir e o método aprender de forma superior aos métodos clássicos como atribuir tipos de geometrias para os pontos (TypeAcc alto), o mesmo não obteve resultados de qualidade na métrica de segmentação. Esse SIoU baixo pode ter sido ocasionado por três motivos principais, são eles:

- **Poucos Pontos:** como o método consegue trabalhar com quantidades de 250.000

pontos sem problemas, uma das possíveis explicações pode ser que 28.000 ainda seja uma quantidade baixa para o método conseguir obter resultados expressivos.

- **Baixa Robustez para Adaptações:** foram feitas modificações para adequar todos os métodos ao *Dataset* LS3DS, nesse contexto, a PrimitiveNet original pode ter parâmetros determinados de forma específica para cada conjunto de dados que utilizam, dificultando a transposição para um novo *dataset*.

- **Não uso de Malhas Triangulares:** como já comentado, a PrimitiveNet original utiliza malhas triangulares como entrada, logo, a adaptação que foi feita para que o método seja capaz de processar nuvens de pontos pode ter impactado de maneira severa os resultados produzidos.

Com base no que foi discutido, os métodos com resultados mais expressivos no presente experimento foram o SMFG-ERANSAC e SMGF-ParseNet. Nesse contexto, apesar do SMGF-ERANSAC ter melhores resultados com relação ao PCov e ao SIoU, o SMFG-ParseNet tem maior acertividade na classificação dos tipos de cada instância, quase dobrando o valor do TypeAcc do outro SMGF-RANSAC. Logo, a escolha do uso de cada uma com em cenários com maiores quantidades de pontos depende diretamente da prioridade para o uso do método. Nesse caso, se os tipos tiverem alta relevância na aplicação para qual se usará o método de ajuste de primitivas, o SMGF-ParseNet é a melhor escolha, por outro lado, caso os tipos não sejam tão impactantes, deve-se priorizar o uso do SMGF-ERANSAC. Alguns resultados visuais para o experimento 3 estão apresentados no Anexo B.

5.3.4 Conclusões Obtidas a partir dos Experimentos

Os resultados apresentados e discutidos nos experimentos demonstram que os métodos adaptados pela metodologia SMGF superam por uma larga margem o estado-da-arte no problema em contextos de larga escala, que é o ERANSAC [53]. Junto a isso, também comprovam o funcionamento do *Dataset* LS3DS como base de treinamento, validação e comparação de métodos baseados em aprendizado.

Com relação as diferentes configurações propostas, a SMGF-HPNet utilizando o *Dataset* LS3DS-2-7K obteve os melhores resultados, onde nesse caso, a mesma supera o ERANSAC por **0.36** pontos em segmentação de instâncias (SIoU), **0.58** pontos em acurácia de tipos (TypeAcc), **0.03** pontos em erro residual (Res.) e **0.33** pontos em percentual de cobertura de pontos (PCoV).

Ademais, outros métodos se destacaram, como a SMGF-ParseNet, pelos seus bons resultados e maior tolerência ao aumento de número de pontos que a SMGF-HPNet ou até mesmo o SMGF-RANSAC, que lidera com o melhor SIoU em diversos contextos, apesar de ter um baixo TypeAcc na maioria dos cenários. Ao contrário, a SMGF-PrimitiveNet

não conseguiu realizar boas segmentações de primitivas geométricas das nuvens de pontos do LS3DS, apesar de ter alcançado bons índices em acurácia de atribuição de tipos.

6 CONCLUSÃO

Motivado pelo *As Built* automático de estruturas industriais de larga escala, o trabalho desenvolvido explorou e contribuiu com o estado-da-arte em ajuste de primitivas geométricas a nuvens de pontos. Nesse contexto, com o objetivo de se utilizar técnicas baseadas em aprendizado, foi desenvolvido uma ferramenta de geração sintética de dados que produz nuvens de pontos realistas e anotadas geometricamente. Assim, o *Dataset* LS3DS foi apresentado, o qual conta com cerca de 86 modelos CAD industriais de larga escala com suas respectivas nuvens de pontos com geometrias anotadas.

A partir do uso do *Dataset* LS3DS, se apresentou uma metodologia para se obter melhores resultados na tarefa central do trabalho, que é o ajuste de primitivas geométricas a nuvens de pontos de larga escala. Essa abordagem, que foi nomeada de SMGF, se baseia na divisão das nuvens de pontos em partes menores para que possam ser processadas individualmente e, ao fim, terem seus respectivos resultados agregados para que se possa gerar a primitivação do modelo por completo. Nesse contexto, os algoritmos que fazem o ajuste geométrico nas partes foram adaptações dos principais métodos da literatura.

De maneira quantitativa e qualitativa, todas as contribuições foram validadas individualmente e em conjunto para comprovar o funcionamento e utilidade de cada uma. Nesse contexto, se demonstrou o funcionamento passo-a-passo da ferramenta de geração em um caso de uso simplificado, demonstrando gráficos e imagens dos resultados gerados. Logo após, o *Dataset* LS3DS foi detalhado através de gráficos de distribuição de diferentes tipos de geometrias e imagens dos modelos que o integram. Ao fim, a metodologia SMGF adaptou os métodos ERANSAC [53], ParseNet [55], HPNet [63] e PrimitiveNet [24], onde se obteve resultados superiores aos do estado-da-arte da literatura, que é a versão não adaptada do ERANSAC.

Por fim, conclui-se que o trabalho tem contribuição científica importante para o campo de processamento, baseados em aprendizado, de dados tridimensionais. Junto a isso, a disponibilização do *Dataset* LS3DS e dos métodos adaptados pela metodologia SMGF abre espaço para que novos trabalhos possam ser treinados, validados e comparados com os discutidos no presente trabalho, propiciando a continuidade das pesquisas na área. Nesse contexto, alguns possíveis trabalhos futuros podem ser identificados, são eles:

- Adição de Ruído na Simulação do Processo de aquisição de Nuvens de pontos por LiDAR;
- Utilização de técnicas de Estimativa de Melhor Ponto de Vista para escolher melhores posições para o drone na geração de pontos por simulação;
- Introduzir anotação semântica nas nuvens de pontos do *dataset*;
- Criar *dataset* híbrido, utilizando nuvens de pontos reais coletadas de estruturas industriais e geometrias oriundas do modelo CAD dessa estrutura;
- Técnica de agregação que não necessite do *ground truth* como referência para guiar o processo;
- Método Baseado em Aprendizado com maior tolerância ao aumento do número de pontos.

REFERÊNCIAS

[1] Ahmed, E., Saint, A., Shabayek, A., Cherenkova, K., Das, R., Gusev, G., Aouada, D., and Ottersten, B. (2018). Deep learning advances on different 3d data representations: A survey. *arXiv preprint arXiv:1808.01462*, 1.

[2] Al-wswasi, M. and Ivanov, A. (2019). A novel and smart interactive feature recognition system for rotational parts using a step file. *The International Journal of Advanced Manufacturing Technology*, 104(1):261–284.

[3] Armeni, I., Sener, O., Zamir, A. R., Jiang, H., Brilakis, I., Fischer, M., and Savarese, S. (2016). 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*.

[4] Behley, J., Garbade, M., Milioto, A., Quenzel, J., Behnke, S., Stachniss, C., and Gall, J. (2019). SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*.

[5] Ben-Shabat, Y. and Gould, S. (2020). Deepfit: 3d surface fitting via neural network weighted least squares. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*, pages 20–34. Springer.

[6] Bénière, R., Subsol, G., Gesquière, G., Le Breton, F., and Puech, W. (2011). Recovering primitives in 3d cad meshes. In *Three-Dimensional Imaging, Interaction, and Measurement*, volume 7864, page 78640R. International Society for Optics and Photonics.

[7] Bing, W., Chen, L., and Yang, B. (2022). Dm-nerf: 3d scene geometry decomposition and manipulation from 2d images. *arXiv preprint arXiv:2208.07227*.

[8] Boulch, A. (2020). Convpoint: Continuous convolutions for point cloud processing. *Computers & Graphics*, 88:24–34.

[9] Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. (2017). Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42.

[10] Chen, M., Hu, Q., Yu, Z., THOMAS, H., Feng, A., Hou, Y., McCullough, K., Ren, F., and Soibelman, L. (2022). Stpls3d: A large-scale synthetic and real aerial photogrammetry 3d point cloud dataset. In *33rd British Machine Vision Conference 2022, BMVC 2022, London, UK, November 21-24, 2022*. BMVA Press.

[11] Choy, C., Gwak, J., and Savarese, S. (2019). 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3075–3084.

[12] Dai, A., Chang, A. X., Savva, M., Halber, M., Funkhouser, T., and Nießner, M. (2017). Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*.

[13] De Boor, C. (1972). On calculating with b-splines. *Journal of Approximation theory*, 6(1):50–62.

[14] Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395.

[15] Fu, H., Cai, B., Gao, L., Zhang, L.-X., Wang, J., Li, C., Zeng, Q., Sun, C., Jia, R., Zhao, B., et al. (2021). 3d-front: 3d furnished rooms with layouts and semantics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10933–10942.

[16] Gao, K., Gao, Y., He, H., Lu, D., Xu, L., and Li, J. (2022). Nerf: Neural radiance field in 3d vision, a comprehensive review. *arXiv preprint arXiv:2210.00379*.

[17] Geuzaine, C. and Remacle, J.-F. (2009a). Gmsh: A 3-d finite element mesh generator with built-in pre-and post-processing facilities. *International journal for numerical methods in engineering*, 79(11):1309–1331.

[18] Geuzaine, C. and Remacle, J.-F. (2009b). Gmsh: A 3-d finite element mesh generator with built-in pre-and post-processing facilities. *International journal for numerical methods in engineering*, 79(11):1309–1331.

[19] Graham, B. (2014). Spatially-sparse convolutional neural networks. *arXiv preprint arXiv:1409.6070*.

[20] Graham, B. and Van der Maaten, L. (2017). Submanifold sparse convolutional networks. *arXiv preprint arXiv:1706.01307*.

[21] Guo, H., Liu, S., Pan, H., Liu, Y., Tong, X., and Guo, B. (2022). Complexgen: Cad reconstruction by b-rep chain complex generation. *ACM Trans. Graph. (SIGGRAPH)*, 41(4).

[22] Hackel, T., Savinov, N., Ladicky, L., Wegner, J. D., Schindler, K., and Pollefeys, M. (2017). SEMANTIC3D.NET: A new large-scale point cloud classification benchmark. In *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, volume IV-1-W1, pages 91–98.

[23] Hu, Q., Yang, B., Xie, L., Rosa, S., Guo, Y., Wang, Z., Trigoni, N., and Markham, A. (2020). Randla-net: Efficient semantic segmentation of large-scale point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11108–11117.

[24] Huang, J., Zhang, Y., and Sun, M. (2021). Primitivenet: Primitive instance segmentation with local primitive embedding under adversarial metric. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 15323–15333.

[25] ISO 10303-203:1994 (1994). Industrial automation systems and integration — product data representation and exchange — part 203: Application protocol: Configuration controlled 3d designs of mechanical parts and assemblies. Standard, International Organization for Standardization.

[26] Kaiser, A., Ybanez Zepeda, J. A., and Boubekeur, T. (2019). A survey of simple geometric primitives detection methods for captured 3d data. In *Computer Graphics Forum*, volume 38, pages 167–196. Wiley Online Library.

[27] Koch, S., Matveev, A., Jiang, Z., Williams, F., Artemov, A., Burnaev, E., Alexa, M., Zorin, D., and Panozzo, D. (2019). Abc: A big cad model dataset for geometric deep learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[28] Krähenbühl, P. and Koltun, V. (2013). Parameter learning and convergent inference for dense random fields. In *International Conference on Machine Learning*, pages 513–521. PMLR.

[29] Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

[30] Landrieu, L. and Simonovsky, M. (2018). Large-scale point cloud semantic segmentation with superpoint graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4558–4567.

[31] Li, D. and Feng, C. (2019). Primitive fitting using deep geometric segmentation. In *ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction*, volume 36, pages 780–787. IAARC Publications.

[32] Li, L., Sung, M., Dubrovina, A., Yi, L., and Guibas, L. J. (2018). Supervised fitting of geometric primitives to 3d point clouds. *CoRR*, abs/1811.08988.

[33] Li, Y., Liu, S., Yang, X., Guo, J., Guo, J., and Guo, Y. (2023). Surface and edge detection for primitive fitting of point clouds. In *ACM SIGGRAPH 2023 conference proceedings*, pages 1–10.

[34] Li, Y., Wu, X., Chrysathou, Y., Sharf, A., Cohen-Or, D., and Mitra, N. J. (2011). Globfit: Consistently fitting primitives by discovering global relations. In *ACM SIGGRAPH 2011 papers*, pages 1–12.

[35] Limberger, F. A. and Oliveira, M. M. (2015). Real-time detection of planar regions in unorganized point clouds. *Pattern Recognition*, 48(6):2043–2053.

[36] Liu, B., Wang, M., Foroosh, H., Tappen, M., and Pensky, M. (2015). Sparse convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 806–814.

[37] Maurell, I. P., Ferreira, C., Eguti, C. A., and Drews-Jr, P. (2021). Geometric primitive fitting in large structure 3d point clouds acquired by drones. In *2021 Latin American Robotics Symposium (LARS), 2021 Brazilian Symposium on Robotics (SBR), and 2021 Workshop on Robotics in Education (WRE)*, pages 108–113.

[38] Monszpart, A., Mellado, N., Brostow, G. J., and Mitra, N. J. (2015). RAPter: Rebuilding man-made scenes with regular arrangements of planes. *ACM Trans. Graph.*, 34(4):103:1–103:12.

[39] Ochmann, S., Vock, R., Wessel, R., and Klein, R. (2016). Automatic reconstruction of parametric building models from indoor point clouds. *Computers & Graphics*, 54:94–103.

[40] Oesau, S., Verdie, Y., Jamin, C., Alliez, P., Lafarge, F., Giraudot, S., Hoang, T., and Anisimov, D. (2023). Shape detection. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.6 edition.

[41] Pătrăucean, V., Armeni, I., Nahangi, M., Yeung, J., Brilakis, I., and Haas, C. (2015). State of research in automatic as-built modelling. *Advanced Engineering Informatics*, 29(2):162–171.

[42] Piegl, L. and Tiller, W. (1996). *The NURBS Book*. Springer-Verlag, New York, NY, USA, second edition.

[43] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2016). Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593.

[44] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017a). Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[45] Qi, C. R., Yi, L., Su, H., and Guibas, L. J. (2017b). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*.

[46] Qin, Y., Lu, W., Qi, Q., Liu, X., Zhong, Y., Scott, P. J., and Jiang, X. (2017). Status, comparison, and issues of computer-aided design model data exchange methods based on standardized neutral files and web ontology language file. *Journal of Computing and Information Science in Engineering*, 17(1).

[47] Rabbani, T., Dijkman, S., van den Heuvel, F., and Vosselman, G. (2007). An integrated approach for modelling and global registration of point clouds. *ISPRS journal of Photogrammetry and Remote Sensing*, 61(6):355–370.

[48] Remacle, J.-F., Henrotte, F., Carrier-Baudouin, T., Béchet, E., Marchandise, E., Geuzaine, C., and Mouton, T. (2013). A frontal delaunay quad mesh generator using the l∞ norm. *International Journal for Numerical Methods in Engineering*, 94(5):494–512.

[49] Roth, S. D. (1982). Ray casting for modeling solids. *Computer graphics and image processing*, 18(2):109–144.

[50] Sagar, S. and Sharma, A. (2018). Splinenet: B-spline neural network for efficient classification of 3d data. pages 1–8.

[51] SAS, O. C. (2023a). Open CASCADE Technology. `https://dev.opencascade.org/`. Version 7.8.

[52] SAS, O. C. (2023b). *Open CASCADE Technology: Mesh*. Open CASCADE SAS. Version 7.8.

[53] Schnabel, R., Wahl, R., and Klein, R. (2007). Efficient ransac for point-cloud shape detection. In *Computer graphics forum*, volume 26, pages 214–226. Wiley Online Library.

[54] Schult, J., Engelmann, F., Hermans, A., Litany, O., Tang, S., and Leibe, B. (2023). Mask3D: Mask Transformer for 3D Semantic Instance Segmentation.

[55] Sharma, G., Liu, D., Maji, S., Kalogerakis, E., Chaudhuri, S., and Měch, R. (2020). Parsenet: A parametric surface fitting network for 3d point clouds. In *European Conference on Computer Vision*, pages 261–276. Springer.

[56] Sirohi, K., Mohan, R., Büscher, D., Burgard, W., and Valada, A. (2021). Efficientlps: Efficient lidar panoptic segmentation. *arXiv preprint arXiv:2102.08009*.

[57] Sun, C., Cai, G., Li, Z., Yan, K., Zhang, C., Marshall, C., Huang, J.-B., Zhao, S., and Dong, Z. (2023). Neural-pbir reconstruction of shape, material, and illumination. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 18046–18056.

[58] The CGAL Project (2023). *CGAL User and Reference Manual*. CGAL Editorial Board, 5.6 edition.

[59] Tuan Duc Ngo, Binh-Son Hua, K. N. (2023). Isbnet: a 3d point cloud instance segmentation network with instance-aware sampling and box-aware dynamic convolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[60] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.

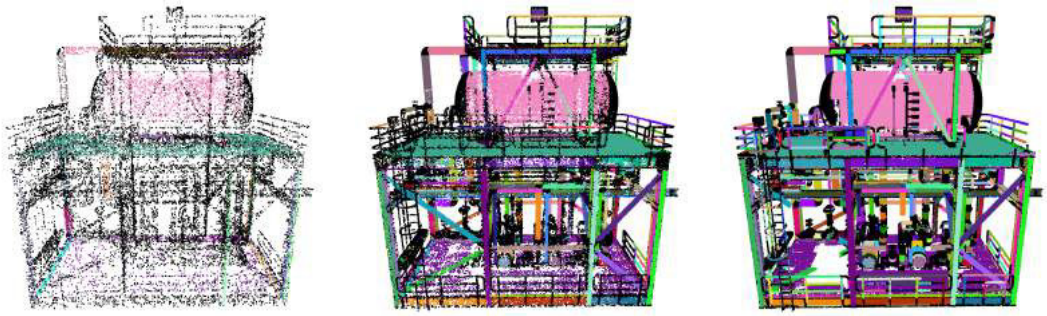
[61] Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. (2019). Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*.

[62] Wold, S., Esbensen, K., and Geladi, P. (1987). Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52.

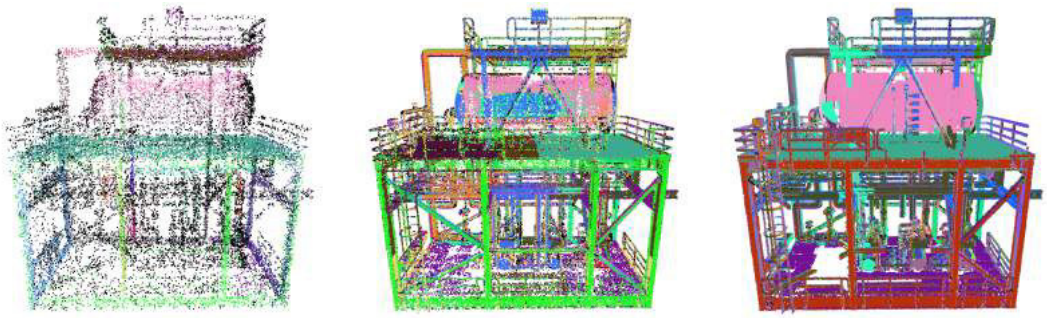
[63] Yan, S., Yang, Z., Ma, C., Huang, H., Vouga, E., and Huang, Q. (2021). Hpnet: Deep primitive segmentation using hybrid representations. *arXiv preprint arXiv:2105.10620*.

[64] Zhou, Q.-Y., Park, J., and Koltun, V. (2018). Open3D: A modern library for 3D data processing. *arXiv:1801.09847*.

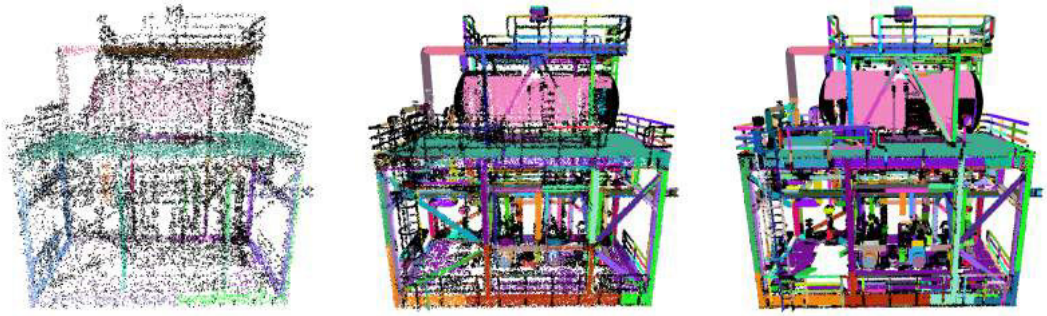
A RESULTADOS VISUAIS DO EXPERIMENTO 2



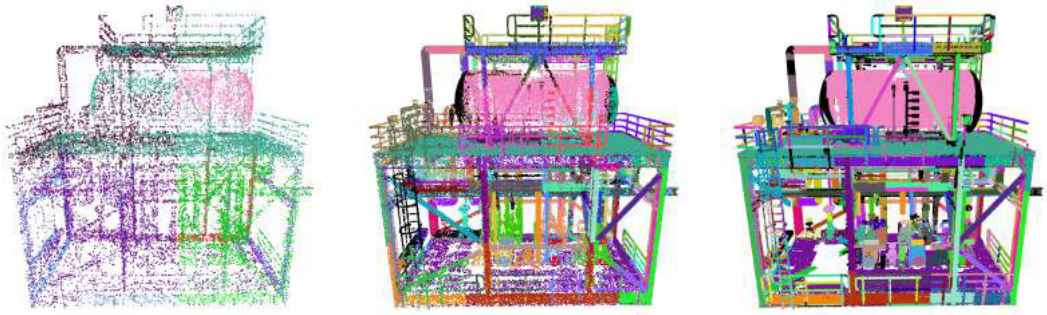
(a) *Ground Truth 8-7K* (b) *Ground Truth 4-7K* (c) *Ground Truth 2-7K*



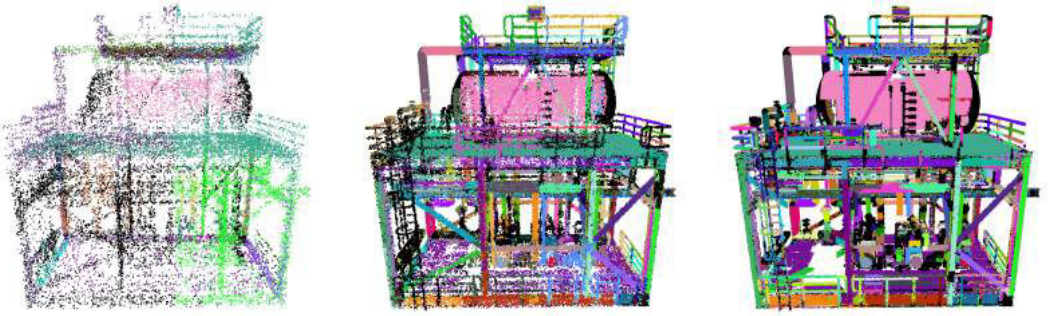
(d) ERANSAC 8-7K (e) ERANSAC 4-7K (f) ERANSAC 2-7K



(g) SMGF-ERANSAC 8-7K (h) SMGF-ERANSAC 4-7K (i) SMGF-ERANSAC 2-7K

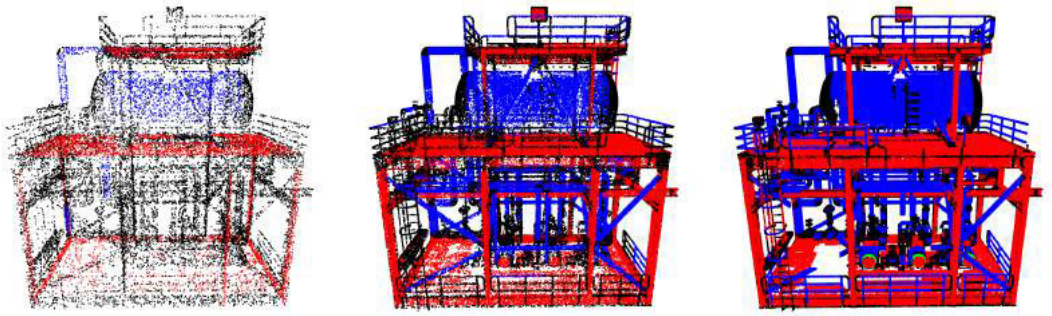


(j) SMGF-ParseNet 8-7K (k) SMGF-ParseNet 4-7K (l) SMGF-ParseNet 2-7K

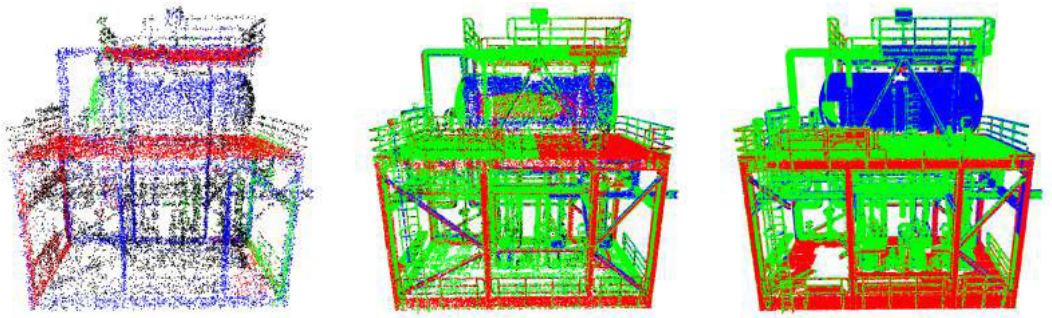


(m) SMGF-HPNet 8-7K (n) SMGF-HPNet 4-7K (o) SMGF-HPNet 2-7K

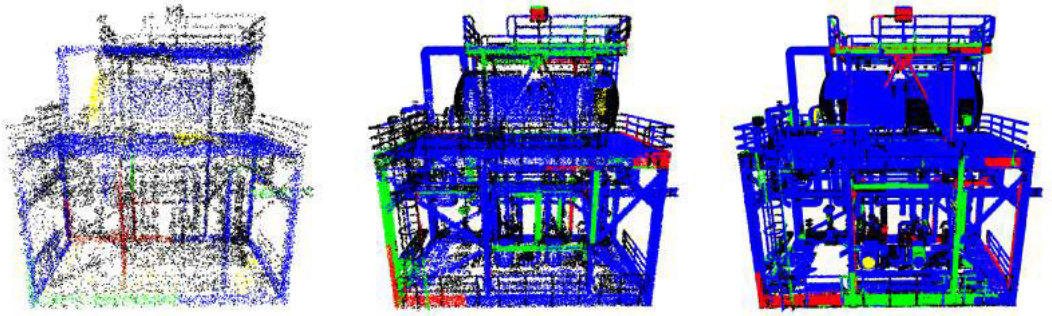
Figura 31: Resultados de Segmentação de Instâncias Geométricas do Experimento 2.



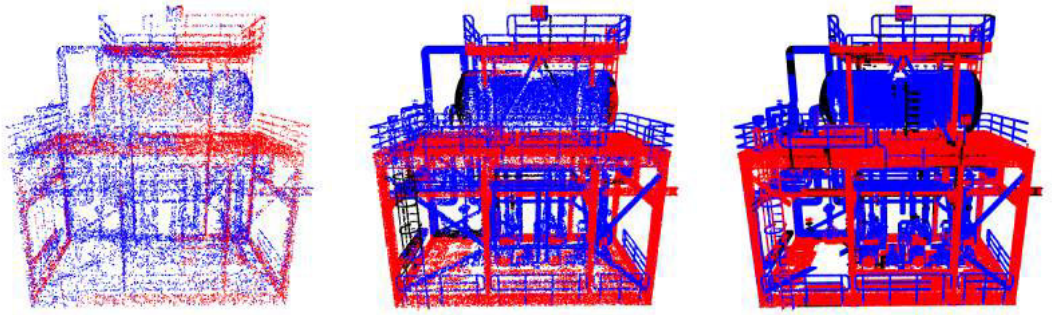
(a) *Ground Truth 8-7K* (b) *Ground Truth 4-7K* (c) *Ground Truth 2-7K*



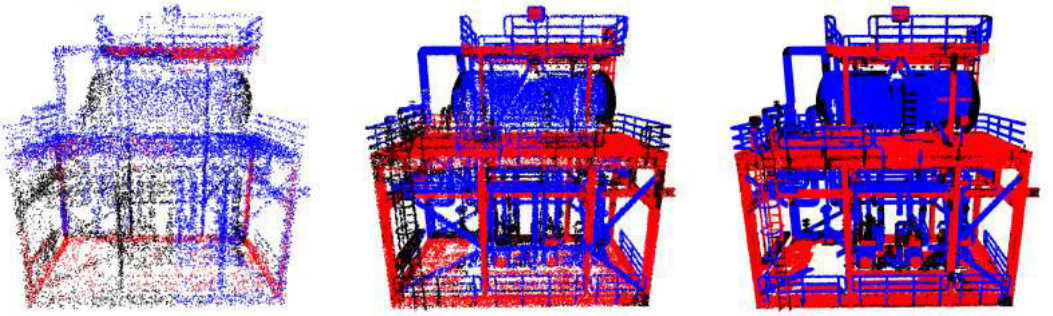
(d) ERANSAC 8-7K (e) ERANSAC 4-7K (f) ERANSAC 2-7K



(g) SMGF-ERANSAC 8-7K (h) SMGF-ERANSAC 4-7K (i) SMGF-ERANSAC 2-7K

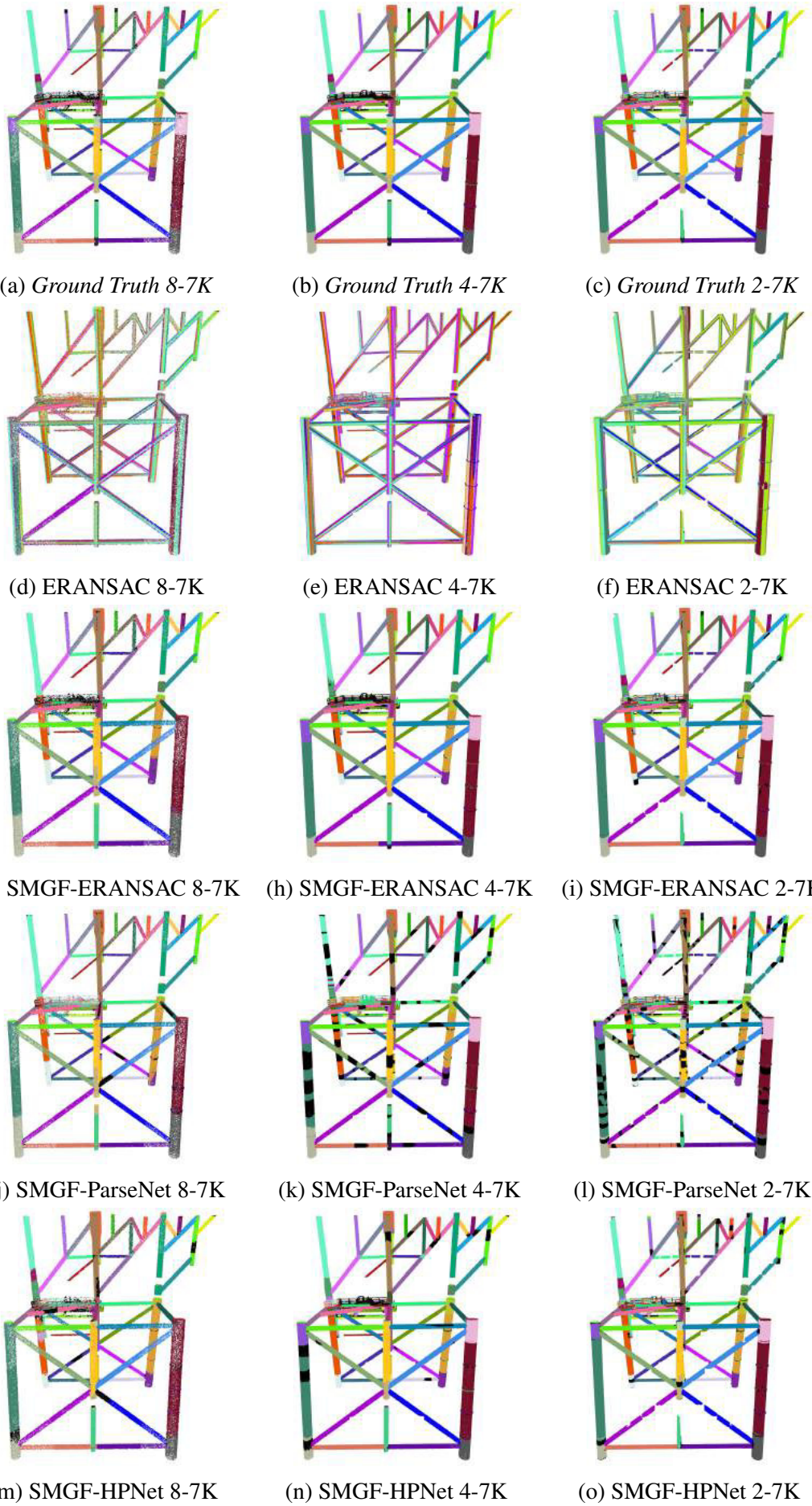


(j) SMGF-ParseNet 8-7K (k) SMGF-ParseNet 4-7K (l) SMGF-ParseNet 2-7K



(m) SMGF-HPNet 8-7K (n) SMGF-HPNet 4-7K (o) SMGF-HPNet 2-7K

Figura 32: Resultados de Segmentação de Tipos Geométricos do Experimento 2. Os tipos possíveis são: Plano (●), Cilindro (●), Cone (●), Esfera (●) e sem tipo (●.



(a) *Ground Truth 8-7K* (b) *Ground Truth 4-7K* (c) *Ground Truth 2-7K*

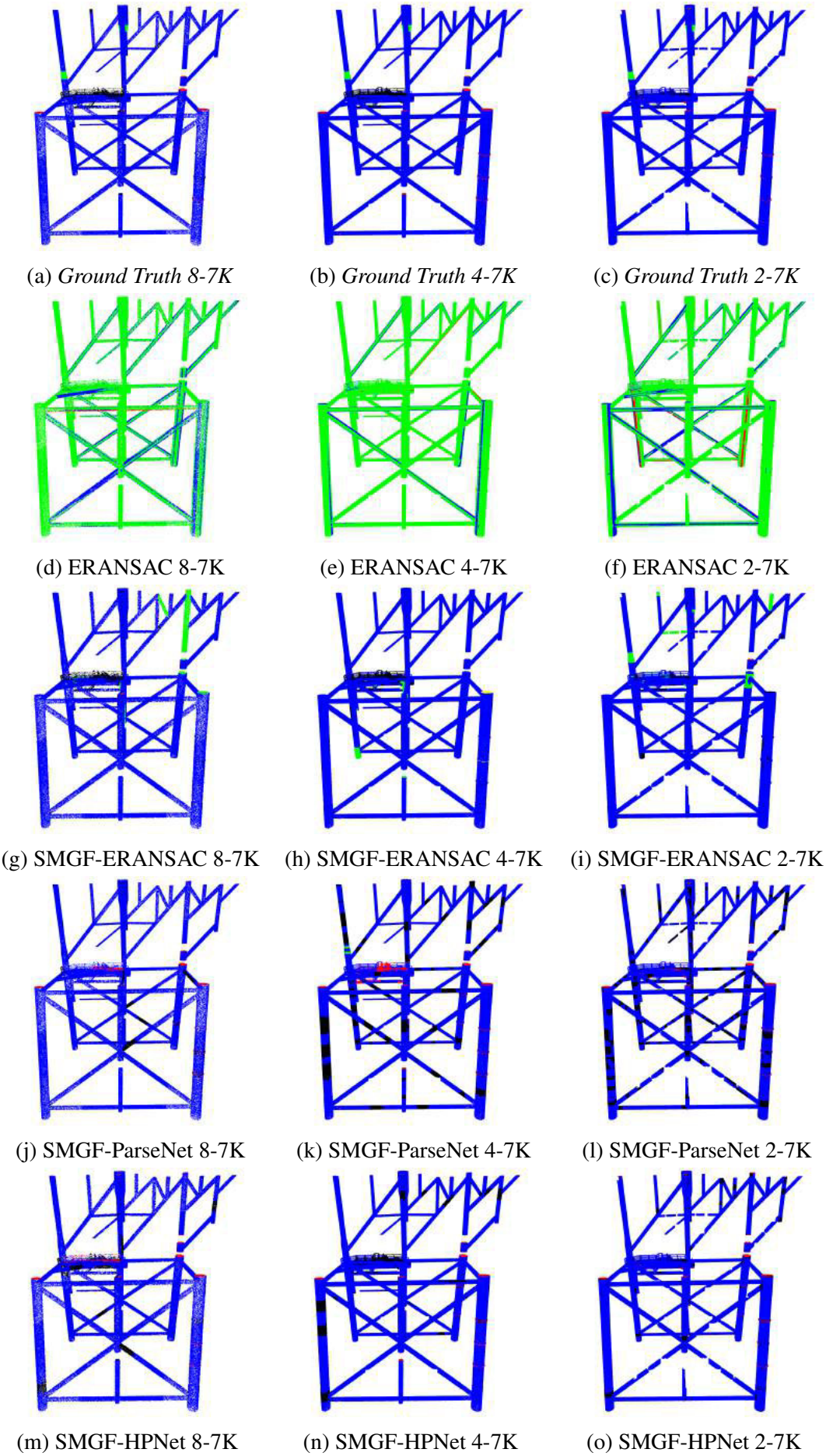
(d) ERANSAC 8-7K (e) ERANSAC 4-7K (f) ERANSAC 2-7K

(g) SMGF-ERANSAC 8-7K (h) SMGF-ERANSAC 4-7K (i) SMGF-ERANSAC 2-7K

(j) SMGF-ParseNet 8-7K (k) SMGF-ParseNet 4-7K (l) SMGF-ParseNet 2-7K

(m) SMGF-HPNet 8-7K (n) SMGF-HPNet 4-7K (o) SMGF-HPNet 2-7K

Figura 33: Resultados de Segmentação de Instâncias Geométricas do Experimento 2.



(a) *Ground Truth 8-7K* (b) *Ground Truth 4-7K* (c) *Ground Truth 2-7K*

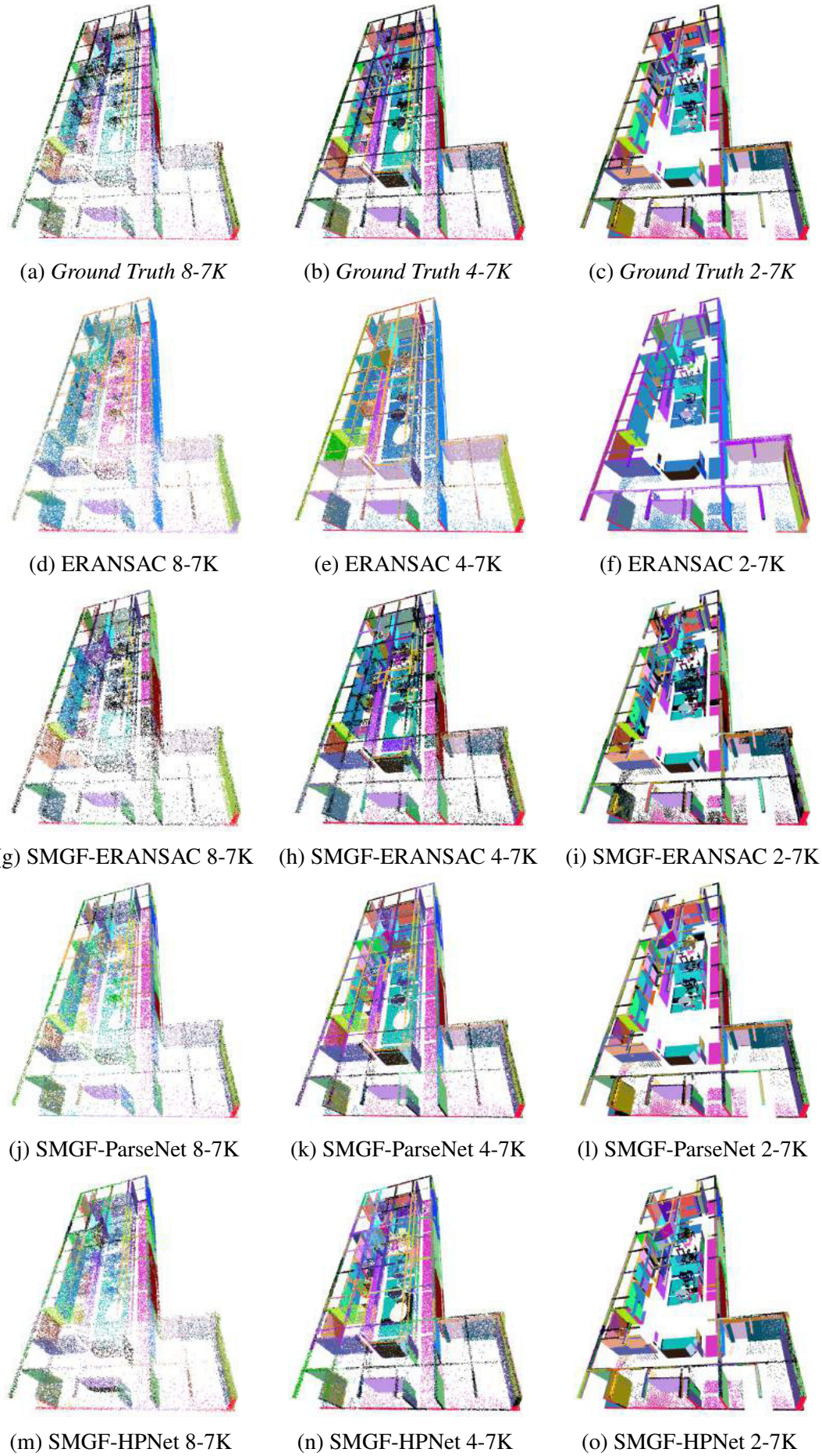
(d) ERANSAC 8-7K (e) ERANSAC 4-7K (f) ERANSAC 2-7K

(g) SMGF-ERANSAC 8-7K (h) SMGF-ERANSAC 4-7K (i) SMGF-ERANSAC 2-7K

(j) SMGF-ParseNet 8-7K (k) SMGF-ParseNet 4-7K (l) SMGF-ParseNet 2-7K

(m) SMGF-HPNet 8-7K (n) SMGF-HPNet 4-7K (o) SMGF-HPNet 2-7K

Figura 34: Resultados de Segmentação de Tipos Geométricos do Experimento 2. Os tipos possíveis são: Plano (●), Cilindro (●), Cone (●), Esfera (●) e sem tipo (●.



(a) *Ground Truth 8-7K* (b) *Ground Truth 4-7K* (c) *Ground Truth 2-7K*

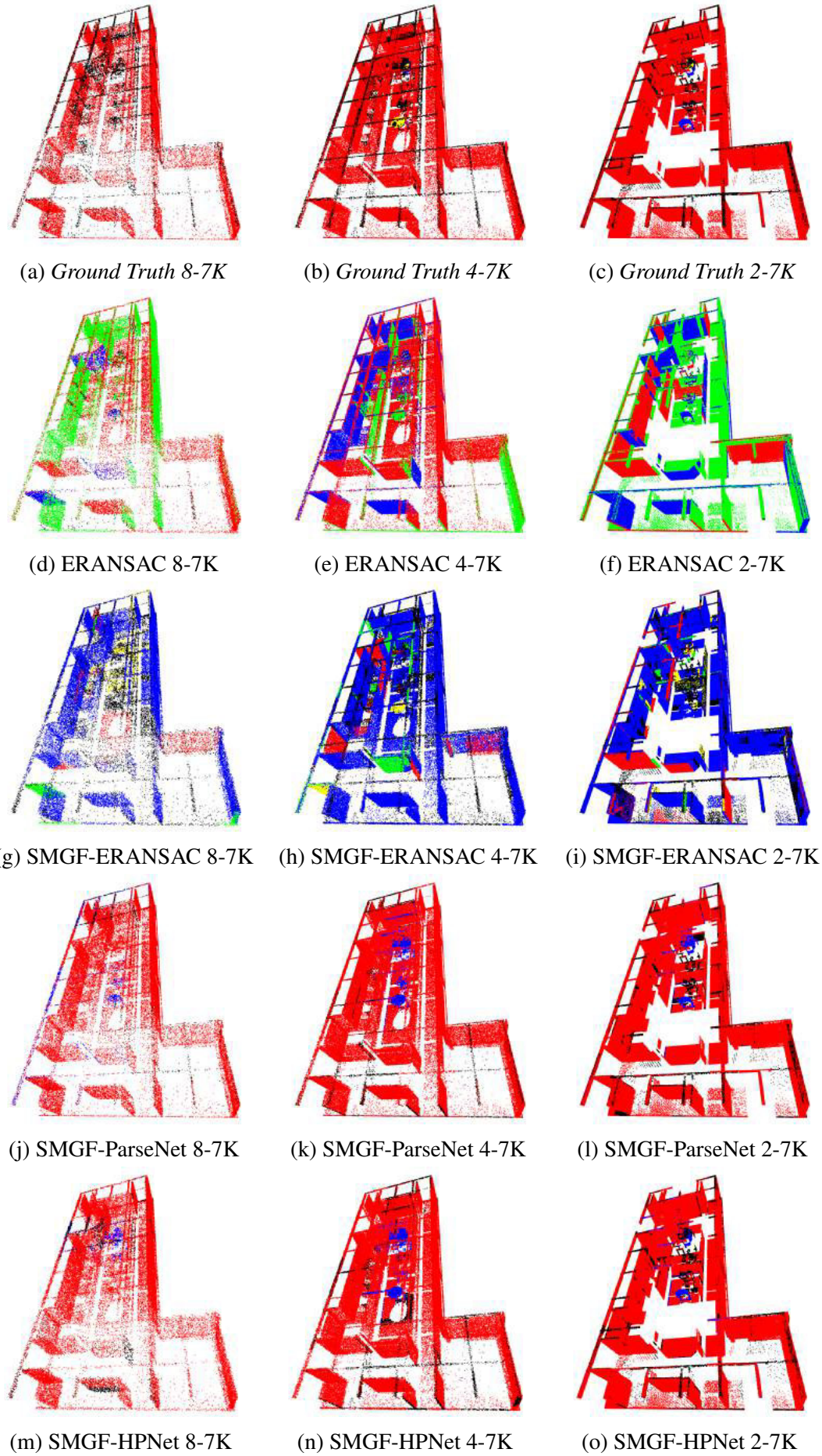
(d) ERANSAC 8-7K (e) ERANSAC 4-7K (f) ERANSAC 2-7K

(g) SMGF-ERANSAC 8-7K (h) SMGF-ERANSAC 4-7K (i) SMGF-ERANSAC 2-7K

(j) SMGF-ParseNet 8-7K (k) SMGF-ParseNet 4-7K (l) SMGF-ParseNet 2-7K

(m) SMGF-HPNet 8-7K (n) SMGF-HPNet 4-7K (o) SMGF-HPNet 2-7K

Figura 35: Resultados de Segmentação de Instâncias Geométricas do Experimento 2.



(a) *Ground Truth 8-7K* (b) *Ground Truth 4-7K* (c) *Ground Truth 2-7K*

(d) ERANSAC 8-7K (e) ERANSAC 4-7K (f) ERANSAC 2-7K

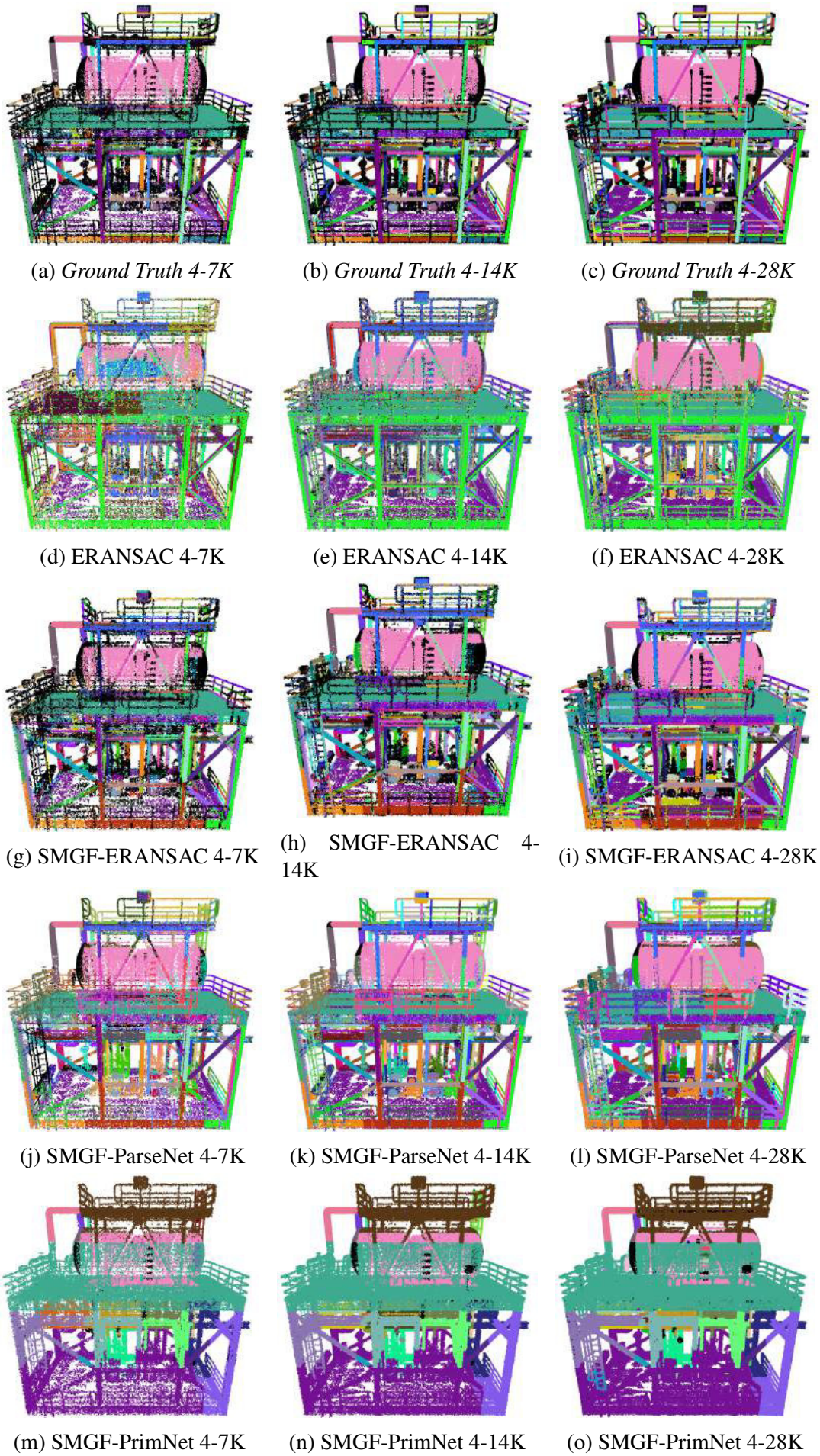
(g) SMGF-ERANSAC 8-7K (h) SMGF-ERANSAC 4-7K (i) SMGF-ERANSAC 2-7K

(j) SMGF-ParseNet 8-7K (k) SMGF-ParseNet 4-7K (l) SMGF-ParseNet 2-7K

(m) SMGF-HPNet 8-7K (n) SMGF-HPNet 4-7K (o) SMGF-HPNet 2-7K

Figura 36: Resultados de Segmentação de Tipos Geométricos do Experimento 2. Os tipos possíveis são: Plano (●), Cilindro (●), Cone (●), Esfera (●) e sem tipo (●.

B RESULTADOS VISUAIS DO EXPERIMENTO 3



(a) *Ground Truth 4-7K* (b) *Ground Truth 4-14K* (c) *Ground Truth 4-28K*

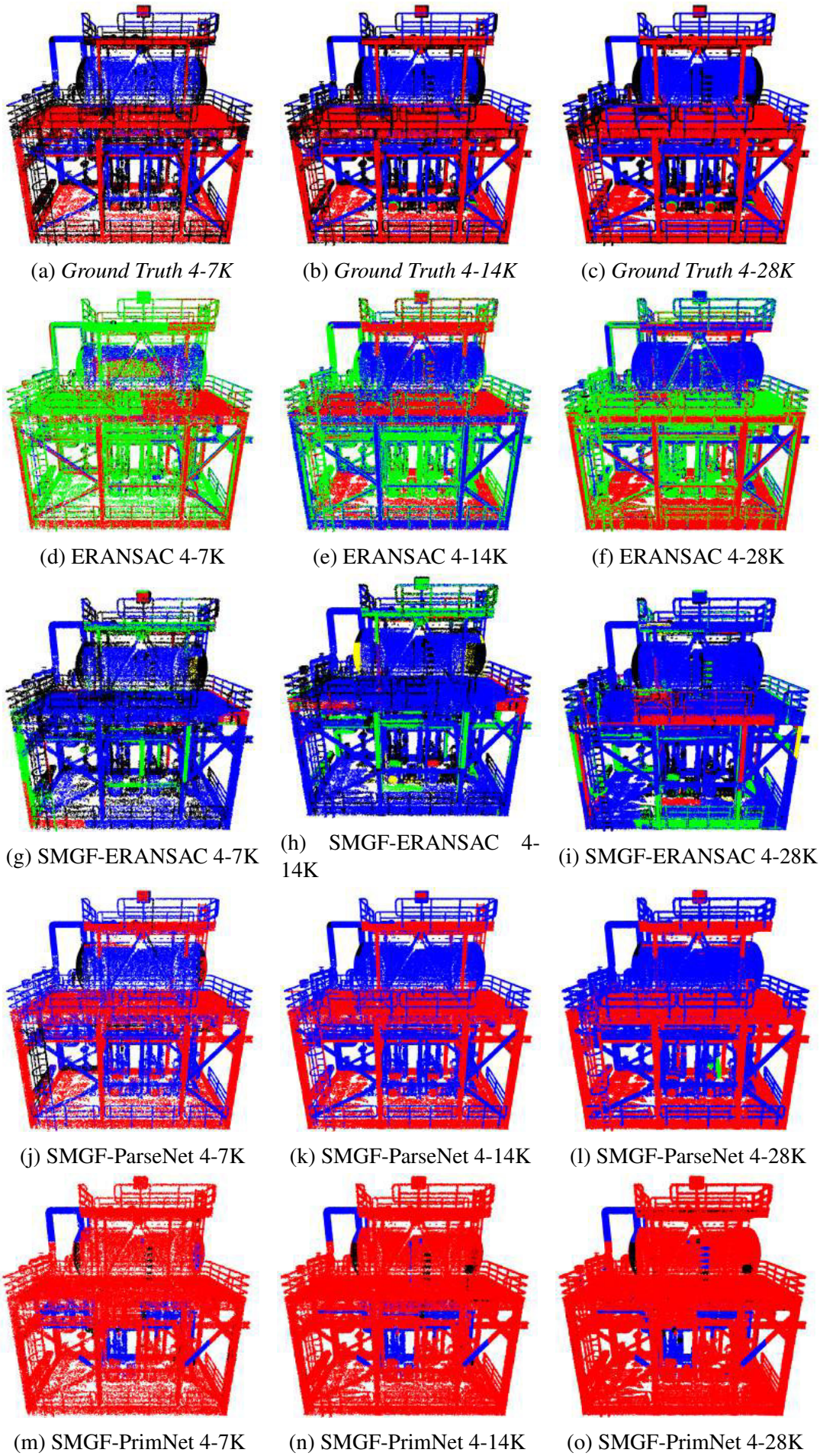
(d) ERANSAC 4-7K (e) ERANSAC 4-14K (f) ERANSAC 4-28K

(g) SMGF-ERANSAC 4-7K (h) SMGF-ERANSAC 4-14K (i) SMGF-ERANSAC 4-28K

(j) SMGF-ParseNet 4-7K (k) SMGF-ParseNet 4-14K (l) SMGF-ParseNet 4-28K

(m) SMGF-PrimNet 4-7K (n) SMGF-PrimNet 4-14K (o) SMGF-PrimNet 4-28K

Figura 37: Resultados de Segmentação de Instâncias Geométricas do Experimento 3.



(a) *Ground Truth 4-7K* (b) *Ground Truth 4-14K* (c) *Ground Truth 4-28K*

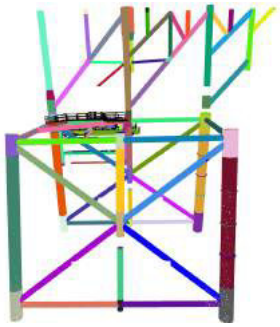
(d) ERANSAC 4-7K (e) ERANSAC 4-14K (f) ERANSAC 4-28K

(g) SMGF-ERANSAC 4-7K (h) SMGF-ERANSAC 4-14K (i) SMGF-ERANSAC 4-28K

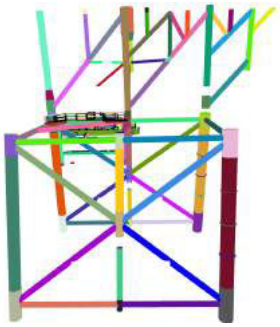
(j) SMGF-ParseNet 4-7K (k) SMGF-ParseNet 4-14K (l) SMGF-ParseNet 4-28K

(m) SMGF-PrimNet 4-7K (n) SMGF-PrimNet 4-14K (o) SMGF-PrimNet 4-28K

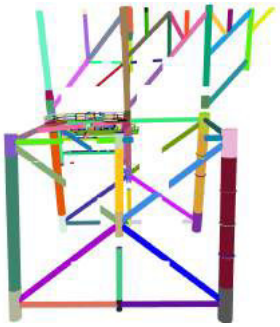
Figura 38: Resultados de Segmentação de Tipos Geométricos do Experimento 3. Os tipos possíveis são: Plano (●), Cilindro (●), Cone (●), Esfera (●) e sem tipo (●).



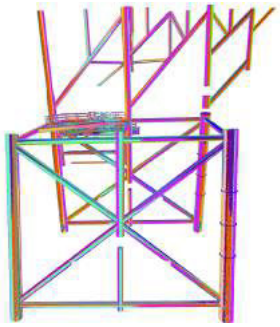
(a) *Ground Truth 4-7K*



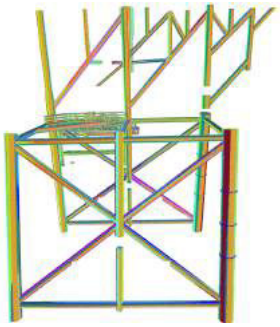
(b) *Ground Truth 4-14K*



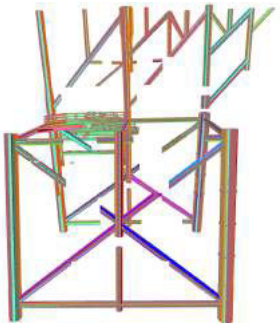
(c) *Ground Truth 4-28K*



(d) ERANSAC 4-7K



(e) ERANSAC 4-14K



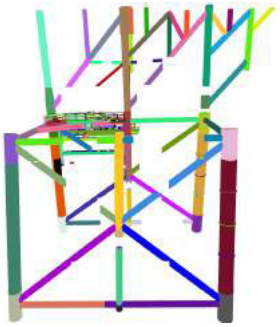
(f) ERANSAC 4-28K



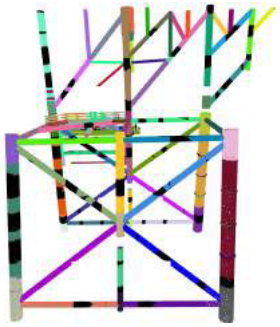
(g) SMGF-ERANSAC 4-7K



(h) SMGF-ERANSAC 4-14K



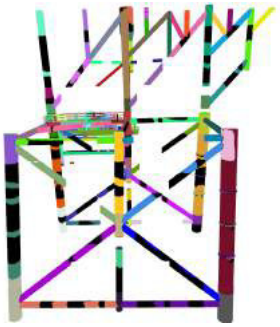
(i) SMGF-ERANSAC 4-28K



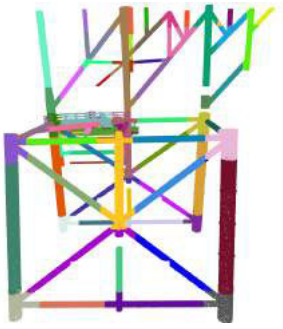
(j) SMGF-ParseNet 4-7K



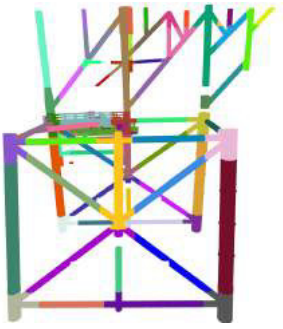
(k) SMGF-ParseNet 4-14K



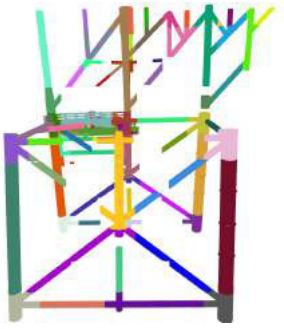
(l) SMGF-ParseNet 4-28K



(m) SMGF-PrimNet 4-7K

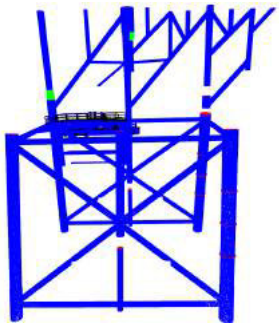


(n) SMGF-PrimNet 4-14K



(o) SMGF-PrimNet 4-28K

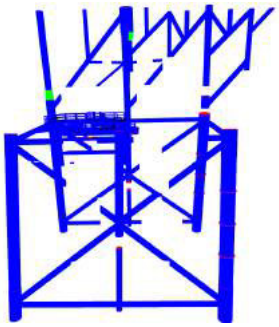
Figura 39: Resultados de Segmentação de Instâncias Geométricas do Experimento 3.



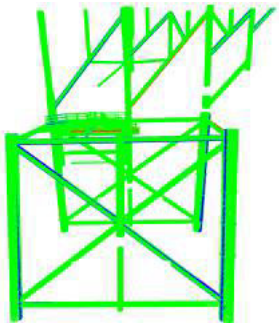
(a) *Ground Truth 4-7K*



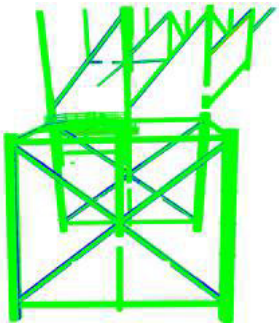
(b) *Ground Truth 4-14K*



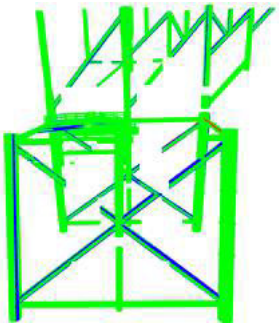
(c) *Ground Truth 4-28K*



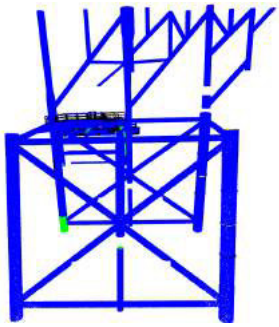
(d) ERANSAC 4-7K



(e) ERANSAC 4-14K



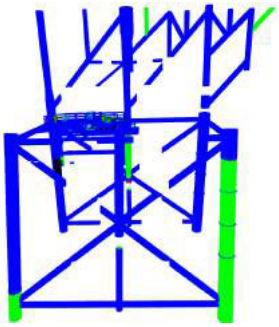
(f) ERANSAC 4-28K



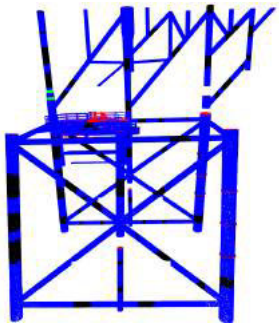
(g) SMGF-ERANSAC 4-7K



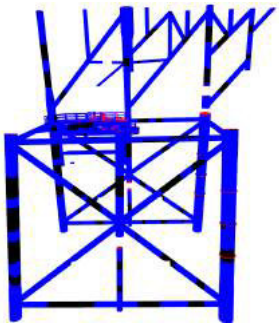
(h) SMGF-ERANSAC 4-14K



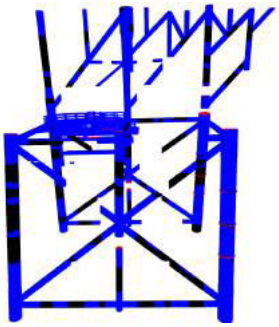
(i) SMGF-ERANSAC 4-28K



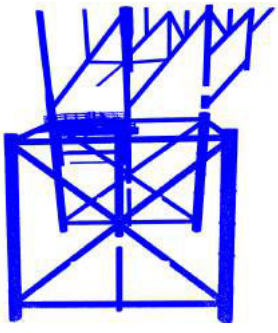
(j) SMGF-ParseNet 4-7K



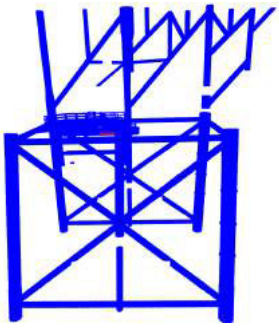
(k) SMGF-ParseNet 4-14K



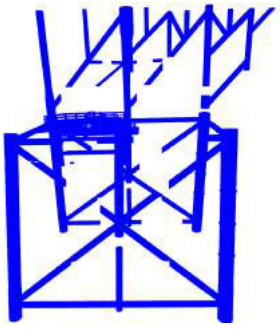
(l) SMGF-ParseNet 4-28K



(m) SMGF-PrimNet 4-7K

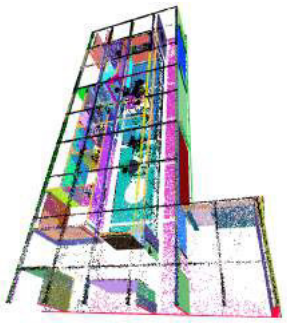


(n) SMGF-PrimNet 4-14K



(o) SMGF-PrimNet 4-28K

Figura 40: Resultados de Segmentação de Tipos Geométricos do Experimento 3. Os tipos possíveis são: Plano (●), Cilindro (●), Cone (●), Esfera (●) e sem tipo (●).



(a) *Ground Truth 4-7K*



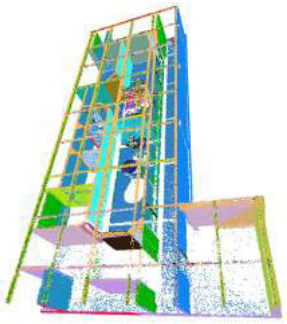
(b) *Ground Truth 4-14K*



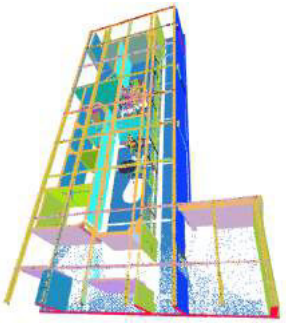
(c) *Ground Truth 4-28K*



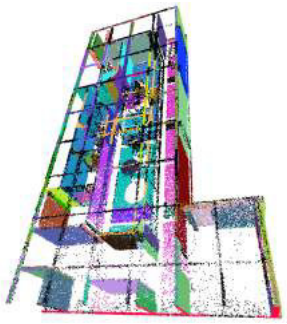
(d) ERANSAC 4-7K



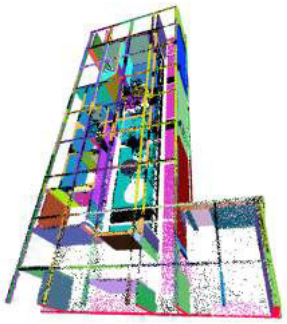
(e) ERANSAC 4-14K



(f) ERANSAC 4-28K



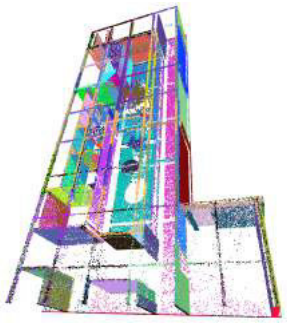
(g) SMGF-ERANSAC 4-7K



(h) SMGF-ERANSAC 4-14K



(i) SMGF-ERANSAC 4-28K



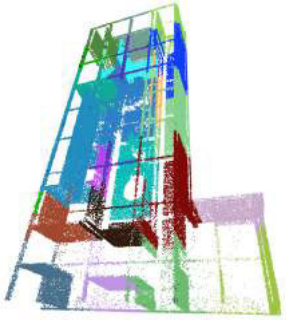
(j) SMGF-ParseNet 4-7K



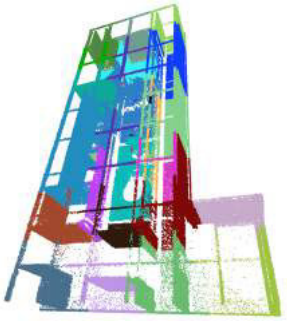
(k) SMGF-ParseNet 4-14K



(l) SMGF-ParseNet 4-28K



(m) SMGF-PrimNet 4-7K



(n) SMGF-PrimNet 4-14K



(o) SMGF-PrimNet 4-28K

Figura 41: Resultados de Segmentação de Instâncias Geométricas do Experimento 3.



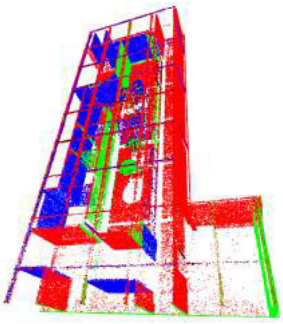
(a) *Ground Truth 4-7K*



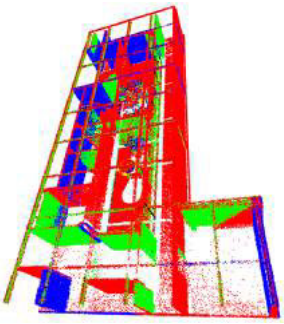
(b) *Ground Truth 4-14K*



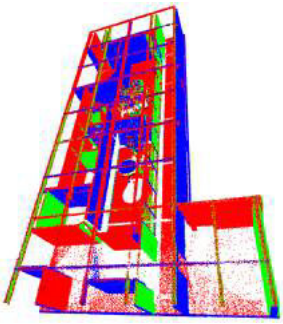
(c) *Ground Truth 4-28K*



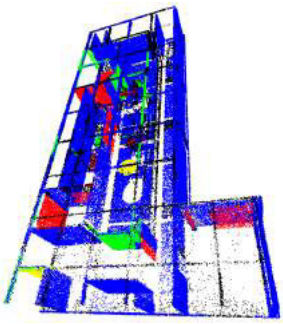
(d) ERANSAC 4-7K



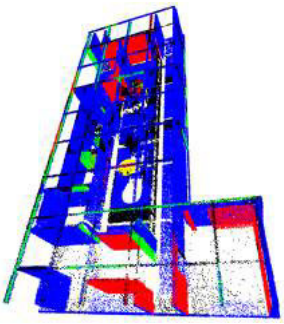
(e) ERANSAC 4-14K



(f) ERANSAC 4-28K



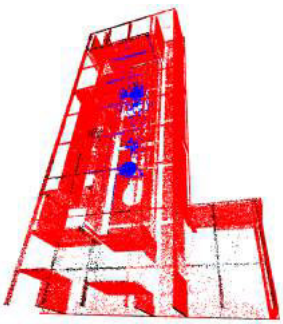
(g) SMGF-ERANSAC 4-7K



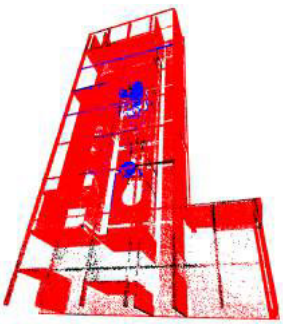
(h) SMGF-ERANSAC 4-14K



(i) SMGF-ERANSAC 4-28K



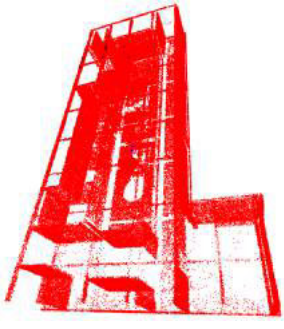
(j) SMGF-ParseNet 4-7K



(k) SMGF-ParseNet 4-14K



(l) SMGF-ParseNet 4-28K



(m) SMGF-PrimNet 4-7K



(n) SMGF-PrimNet 4-14K



(o) SMGF-PrimNet 4-28K

Figura 42: Resultados de Segmentação de Tipos Geométricos do Experimento 3. Os tipos possíveis são: Plano (●), Cilindro (●), Cone (●), Esfera (●) e sem tipo (●).