Lucas Caracas de Figueiredo

Deep-Learning-Based Shape Matching Framework on 3D CAD Models

Tese de Doutorado

Thesis presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Informática.

Advisor : Prof. Waldemar Celes Filho
Co-advisor: Dr. Paulo Ivson Netto Santos

Rio de Janeiro
September 2022

Lucas Caracas de Figueiredo

Deep-Learning-Based Shape Matching Framework on 3D CAD Models

Thesis presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Informática. Approved by the Examination Committee.

Rio de Janeiro, September 16th, 2022

Lucas Caracas de Figueiredo

Lucas Figueiredo received his Bachelor degree in Computer Science from the Universidade Federal do Maranhão (UFMA) in 2015. He also received his Master Degree in Computer Science with emphasis in Computer Graphics from Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) in 2017.

Bibliographic data

Figueiredo, Lucas Caracas de

Deep-Learning-Based Shape Matching Framework on 3D CAD Models / Lucas Caracas de Figueiredo; advisor: Waldemar Celes Filho; co-advisor: Paulo Ivson Netto Santos. – Rio de janeiro: PUC-Rio, Departamento de Informática, 2022.

v., 78 f: il. color. ; 30 cm

Tese (doutorado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2022.

Inclui bibliografia

1. Informática – Teses. 2. Modelos CAD 3D;. 3. Correspondência de Formas;. 4. Aprendizado Profundo;. 5. Nuvem de Pontos.. I. Celes Filho, Waldemar. II. Ivson, Paulo. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. IV. Título.

CDD: 004

Acknowledgments

First, I would like to thank my family, Severino, Luciana, and Caetano, for cheering me up and for all the love and support. Without you, I would never have come this far.
I wish to thank my advisor Waldemar Celes for all the support, guidance, insightful conversations, and commitment to our research. I feel honored to have worked with you for all these years.
I wish to thank my friend and co-advisor Paulo Ivson for all the insightful conversations and commitment to our research. Your support was essential in the development of this research.
I also would like to thank all my friends and colleagues, especially André, Wallas, Eduardo, Rafael, Suellen, Rustam, Carol, Thiago, Ruberth, Dayson, and Dalai. Thanks for sharing your knowledge with me and for inspiring me through this journey.
My deepest thanks to prof. Thadeu, for welcoming me into the GEDi/Tecgraf, and giving me the opportunity to grow academically and professionally.
Thanks to all the PUC-Rio professors for all the knowledge shared with me.
Thanks to PUC-Rio and Tecgraf, for the financial assistance and support.
The present work was supported by the CNPq, Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil (process 161142/2017-6).
This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

Abstract

Figueiredo, Lucas Caracas de; Celes Filho, Waldemar (Advisor); Ivson, Paulo (Co-Advisor). **Deep-Learning-Based Shape Matching Framework on 3D CAD Models**. Rio de Janeiro, 2022. 78p. Tese de doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Data-rich 3D CAD models are essential during different life-cycle stages of engineering projects. Due to the recent popularization of Build Information Modeling methodology and the use of Digital Twins for intelligent manufacturing, the amount of detail, size, and complexity of these models have significantly increased. Although these models are composed of several repeated geometries, plant-design software usually does not provide any instancing information. Previous works have shown that removing redundancy in the representation of 3D CAD models significantly reduces their storage and memory requirements, whilst facilitating rendering optimizations. This work proposes a deep-learning-based shape-matching framework that minimizes a 3D CAD model's redundant information in this regard. We rely on recent advances in the deep processing of point clouds, overcoming drawbacks from previous work, such as heavy dependency on vertex ordering and topology of triangle meshes. The developed framework uses uniformly sampled point clouds to identify similarities among meshes in 3D CAD models and computes an optimal affine transformation matrix to instantiate them. Results on actual 3D CAD models demonstrate the value of the proposed framework. The developed point-cloud-registration procedure achieves a lower surface error while also performing faster than previous approaches. The developed supervised-classification approach achieves equivalent results compared to earlier, limited methods and significantly outperformed them in a vertex shuffling scenario. We also propose a self-supervised approach that clusters similar meshes and overcomes the need for explicitly labeling geometries in the 3D CAD model. This self-supervised method obtains competitive results when compared to previous approaches, even outperforming them in certain scenarios.

Keywords

3D CAD Models; Shape Matching; Deep Learning; Point Cloud.

Resumo

Figueiredo, Lucas Caracas de; Celes Filho, Waldemar; Ivson, Paulo. **Arcabouço para Correspondência de Formas baseado em Aprendizado Profundo em Modelos CAD 3D**. Rio de Janeiro, 2022. 78p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Modelos CAD 3D ricos em dados são essenciais durante os diferentes estágios do ciclo de vida de projetos de engenharia. Devido à recente popularização da metodologia Modelagem de Informação da Construção e do uso de Gêmeos Digitais para a manufatura inteligente, a quantidade de detalhes, o tamanho, e a complexidade desses modelos aumentaram significativamente. Apesar desses modelos serem compostos de várias geometrias repetidas, os softwares de projeto de plantas geralmente não proveem nenhuma informação de instanciação. Trabalhos anteriores demonstraram que removendo a redundância na representação dos modelos CAD 3D reduz significativamente o armazenamento e requisição de memória deles, ao passo que facilita otimizações de renderização. Este trabalho propõe um arcabouço para correspondência de formas baseado em aprendizado profundo que minimiza as informações redundantes de um modelo CAD 3D a esse respeito. Nos apoiamos nos avanços recentes no processamento profundo de nuvens de pontos, superando desvantagens de trabalhos anteriores, como a forte dependencia da ordenação dos vértices e topologia das malhas de triângulos. O arcabouço desenvolvido utiliza nuvens de pontos uniformemente amostradas para identificar similaridades entre malhas em modelos CAD 3D e computam uma matriz de transformação afim ótima para instancia-las. Resultados em modelos CAD 3D reais demonstram o valor do arcabouço proposto. O procedimento de registro de nuvem de pontos desenvolvido atinge um erro de superfície menor, ao mesmo tempo que executa mais rápido que abordagens anteriores. A abordagem supervisionada de classificação desenvolvida antinge resultados equivalentes em comparação com métodos limitados anteriores e os superou significativamente num cenário de embaralhamento de vértices. Propomos também uma abordagem auto-supervisionada que agrupa malhas semelhantes e supera a necessidade de rotular explicitamente as geometrias no modelo CAD 3D. Este método auto-supervisionado obtém resultados competitivos quando comparados às abordagens anteriores, até mesmo superando-as em determinados cenários.

Palavras-chave

Modelos CAD 3D; Correspondência de Formas; Aprendizado Profundo; Nuvem de Pontos.

Table of contents

List of figures

List of tables

List of abreviations

BIM	–	Building Information Modeling
CAD	–	Computer-Aided Design
CNN	–	Convolutional Neural Network
FFD	–	Free-Form Deformation
FPS	–	Frames Per Second
HDBSCAN	–	Hierarchical Density-Based Spatial Clustering of Applications with Noise
ICP	–	Iterative Closest Point
IR	–	Instance Registration
LK	–	Lucas & Kanade
MST	–	Minimum Spanning Tree
PCA	–	Principal Component Analysis
RANSAC	–	Random Sample Consensus
SICP	–	Scaling Iterative Closest Point

If you have the habit of taking things with joy, you will seldom find yourself in difficult circumstances.

Baden-Powell.

1 Introduction

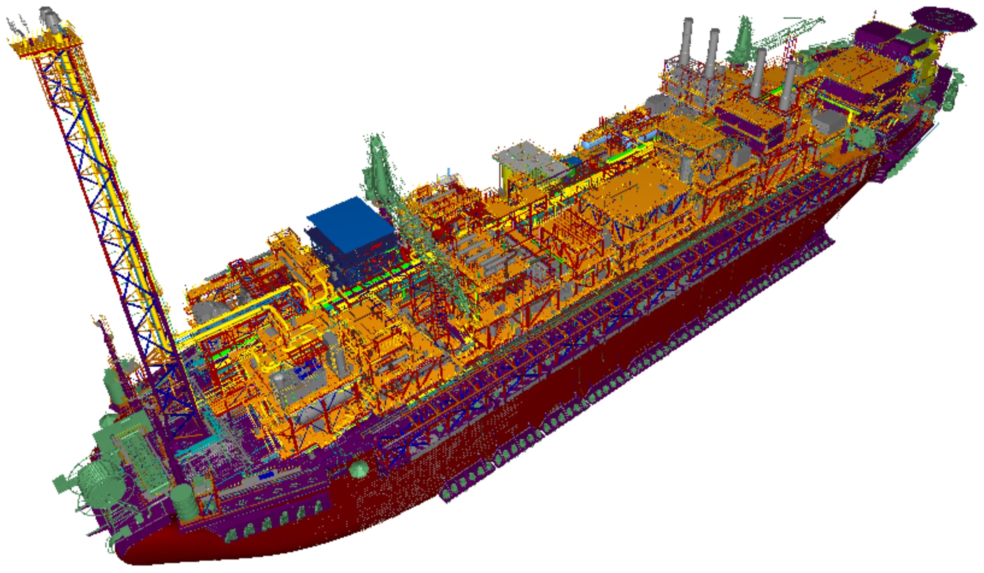
Computer-Aided Design (CAD) systems, used to design data-rich 3D CAD models, play an important role throughout the different phases of engineering projects' life cycle, such as: construction design, maintenance planning and execution, and operations monitoring. In civil construction, for example, the Build Information Modeling (BIM) methodology promotes the use of 3D CAD models to efficiently analyze large amounts of data, improving the facility management tasks while minimizing its overall costs (Eastman et al., 2011; Gielingh, 2008; Hardin & McCool, 2015; Kim et al., 2017; Process Industries STEP Consortium, 1994).

The recent popularization of the BIM methodology, alongside with the expanding application of Digital Twins for smart manufacturing (Kritzinger et al., 2018; Qi et al., 2018; Shao & Helu, 2020), are increasing the demand for more detailed and complex 3D CAD models. As a consequence, today's datasets can reach millions of polygons with a high level of detail, as shown in Figure 1.1, imposing challenges to different computing tasks, such as efficient storage, transmission, and rendering.

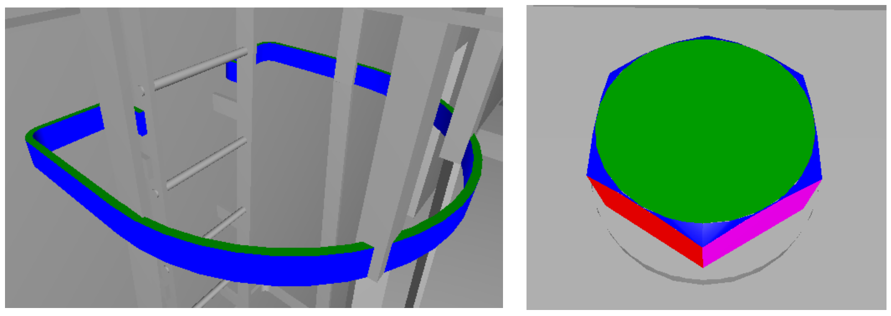
The 3D CAD models of industrial plants are composed of several instances of individual connected components, or geometries, which are represented as primitives and, often, as triangular meshes. Despite the fact that several of these individual geometries appear repeated inside the model, plant-design software usually does not provide any instancing information to minimize such representation redundancy.

Previous research has already addressed the redundancy removal task in 3D CAD models (Santos & Celes Filho, 2014), showing that it is possible to significantly reduce the size and complexity of 3D CAD models, improving their transmission and storage. The authors relied on a least-square minimization shape matching approach to identify repeated triangle meshes on the 3D CAD model, which computed an affine transformation that can be used to instantiate them. Using this approach, the authors showed that rendering performance is also improved, once it enables the use of hardware-accelerated instancing API (Pharr & Fernando, 2005).

However, this approach is heavily dependent on mesh topology and vertex



(a)



(b) (c)

Figure 1.1: In (a), we illustrate an example of a 3D CAD model containing 8,933,975 geometries, and almost half of them are represented as triangle meshes. In (b) and (c), we observe visual structures that are composed of many triangle meshes, highlighted in blue, green, red, and pink, increasing the level of detail and complexity of the CAD model.

incidence order; in reality, this is not always the case. During an engineering project's modeling stage, different parts of the digital plant can be designed by different professionals using different techniques and different software, which can result in different triangulations for the same 3D shape. Furthermore, the 3D CAD model is often updated according to modifications on the real-life industrial plant, and different techniques for data acquisition may be used, such as 3D scanners, which may lead to new 3D geometries with completely different topologies.

With the recent advances in the deep learning field of research, especially geometric deep learning, several geometric tasks obtained significant improvement, such as shape registration (Hanocka et al., 2018), and point cloud processing. In the context of point cloud processing, several tasks benefited from these recent advances, such as point cloud classification and segmentation (Qi et al., 2017a; Qi et al., 2017b), primitive classification and fitting (Li et al., 2019), point cloud registration (Aoki et al., 2019), surface mesh reconstruction (Hanocka et al., 2020), and point cloud representation and clustering (Hassani & Haley, 2019; Rao et al., 2020; Remelli et al., 2019; Zamorski et al., 2020). In spite of these advances, when it comes to the CAD domain, some limitations are found in these existing techniques, such as being specialized to work with primitive geometries or considering only uniform scale transformations, both of which limit the applicability to general 3D CAD models.

Inspired by these recent work, we propose a deep learning-based framework, which relies on uniformly sampled point clouds on triangle meshes, to remove the redundancy representation derived from the repetition of individual geometries on the 3D CAD model. The framework consists of two main steps: first, we identify similar meshes on the 3D CAD model, and second, we estimate an affine transformation matrix that can be used to instantiate them. For the first step, based on the recent success of the PointNet++ (Qi et al., 2017b) network, we developed a supervised approach, training the PointNet++ network to classify an input mesh into one of the known reference triangle meshes that are found on the model. Aiming to address the necessity of a previous knowledge about the geometries found on the model, we also developed a self-supervised approach, training an autoencoder based on the PointNet++ architecture to obtain feature vectors for the model's meshes, which are later used to cluster them in an unsupervised manner.

In order to estimate an affine transformation matrix to instantiate each mesh on the model using its reference mesh, we developed a registration technique, which relies on uniformly sampled point clouds to fit the reference mesh into the original one. To perform this task, the technique uses Principal Component Analysis (PCA) (Wold et al., 1987) and Adam optimization (Kingma & Ba, 2014), taking non-uniform scaling into consideration.

The proposed framework overcomes the main drawbacks of previous approaches. The method is independent of vertex order, and surface triangulation since it relies on point clouds that are uniformly sampled on the triangle meshes surface throughout the framework. Furthermore, the method generalizes for any kind of geometry and is well suited to the 3D CAD domain, whilst computing an affine transformation matrix, taking non-uniform scaling into account, to instantiate generic triangle meshes.

Using the proposed framework with the supervised approach, we were able to reduce the number of triangle meshes required to represent the 3D CAD model of one dataset to as few as 18, translating into a memory reduction of 97.39% from the original size, an equivalent result when compared to Santos & Celes Filho, 2014, method. However, in a worst-case scenario, where the geometries have no vertex and triangulation correspondence, we significantly outperformed the previous work. We also compared the developed point cloud registration technique with previous work, demonstrating that the technique obtains lower surface error, while also performing faster on average.

When using the proposed framework with the self-supervised approach, which removes the need of previous knowledge about the model, we obtained a maximum memory reduction of 83.93% in the same dataset of the supervised approach. This result is close to the result obtained with the supervised approach, and still represents a meaningful overall memory reduction. In another dataset, which contains a greater variety of models and meshes with no labeling information, we were still able to obtain a maximum average memory reduction of 77.63%. When comparing to the previous method proposed by Santos & Celes Filho, 2014, we improved the memory reduction by a maximum of 24.69%, and 5.62% on average.

1.1 Objectives

This work aims to optimize the representation of 3D CAD models by minimizing the redundancy representation derived from the repetition of similar triangle meshes found on them. To achieve this, we propose a supervised and a self-supervised approach that identifies instances of similar triangle meshes and estimates a transformation that can be used to instantiate them. We highlight that obtaining a transformation well suited for 3D CAD models is essential. Regarding this, this work proposes a registration procedure that estimates an affine transformation, considering the non-uniform scales that are frequently observed between instances of repeated triangle meshes, and which is also well suited for the 3D CAD visualization domain.

Furthermore, our goal is also to overcome the following drawbacks found on previous works:

1. **High dependency on vertex incidence order and triangle mesh topology.** As mentioned earlier, similar 3D shapes may present different triangulations, depending on the modeling technique and software that was used, and, as consequence, being heavily dependent on the vertices incidence order and the meshes topology, may limit the amount of

repeated triangle meshes that are identified as instances of one another. To address this issue, the proposed approaches relies on point clouds uniformly sampled on the triangle meshes surfaces to identify similar triangle meshes in the 3D CAD model, and also to estimate an affine transformation that can be used to instantiate them.

2. **Consider only primitive geometries.** The 3D CAD models are composed of primitives and triangle meshes. With the increasing demand for more detailed CAD models, triangle meshes that cannot be represented by a primitive geometry appear more frequently inside the model. In this context, considering only primitives during the redundancy removal process may limit the amount of optimization obtained. In this work, we designed a supervised and a self-supervised framework that can generalize to any kind of geometry, since it takes advantage of point clouds uniformly sampled on generic triangle mesh surfaces.

1.2 Contributions

The conducted study to minimize the redundancy representation found in 3D CAD models has the following contributions:

1. **Supervised and self-supervised frameworks for shape matching on 3D CAD models.** This work proposes both supervised and self-supervised deep-learning-based frameworks to remove redundant information on 3D CAD models. Removing this redundant information on CAD models can improve several tasks associated with their manipulation and processing. Although previous approaches have been developed, they have some limitations. In this context, the proposed supervised framework combines point clouds classification and registration techniques to overcome such limitations, obtaining competitive results by leveraging labeled geometries in the CAD model (97.39% of memory reduction), and outperforming previous work in certain scenarios. On the other hand, the self-supervised framework relies on clustering techniques to overcome, in addition to previous works limitations, also the absence of labeled geometries, maintaining a significant memory reduction on a labeled dataset (83.93%) while outperforming previous work on an unlabeled one by 5.62% on average. To the best of our knowledge, this is the first work to address this task on 3D CAD models in such a generalist way, and the obtained results emphasize the relevance of the proposed

approach. Furthermore, both proposed frameworks have the following advantages:

(a) **Independence of vertex order and mesh topology.** Instead of using point correspondence information to minimize the redundant information in a CAD model, the proposed frameworks rely on uniformly sampled point clouds on the triangles meshes' surface and, as a consequence, is independent of vertex order and mesh topology. This characteristic makes our approaches more robust, obtaining a significant memory reduction under different scenarios.

(b) **Generalization for any kind of geometry.** Considering only primitives during the redundancy removal process may limit the amount of instances found on the CAD model. In this regard, our approaches generalize for any kind of geometry, since they rely on uniformly sampled point clouds throughout the whole redundancy removal process.

(c) **Guaranteed upper bound on geometric errors.** During the 3D CAD model optimization, errors may be introduced by misclassifications or unsatisfactory clustering. Moreover, the registration procedure may eventually introduce errors due to an inadequate registration. In this regard, after the registration, we evaluate the quality of the final instantiated mesh, guaranteeing a specified upper bound on any surface errors found on the optimized model.

2. **Registration algorithm well suited for the 3D CAD domain.** Most previous registration techniques are not well suited for the 3D CAD domain. Indeed, the estimated transformation is not well suited for rendering purposes, and non-uniform scaling, commonly found on 3D CAD geometries, is not considered. The developed registration procedure considers non-uniform scaling when estimating an affine transformation matrix, which is well-suited for rendering purposes and the 3D CAD domain. Furthermore, when compared to previous work that is also well suited for the 3D CAD domain, our procedure was able to obtain lower surface error while being faster on average.

1.3 Document Organization

The remainder of this work is organized as follows. In Chapter 2, we present related works on shape matching and other machine learning tasks, highlighting drawbacks and the reasons why they are not well suited for the 3D CAD domain. Then, in Chapter 3, we make an overview of the main techniques used to perform the identification of similar meshes and the affine transformation matrix estimation.

The proposed framework with the supervised approach is introduced in Chapter 4. In this Chapter, we first present an overview of the framework, detailing each one of its stages. Furthermore, we show the results obtained using a labeled dataset, comparing it with different approaches to estimate the affine transformation matrix. Moreover, we discuss these results, demonstrating the effectiveness of the proposed solution and also its major drawback: the dependency on previous knowledge about the geometries found in the 3D CAD model.

In Chapter 5, we present the framework using a self-supervised approach, which overcomes the previous approach drawback, highlighting the adaptations made to the framework. We also present the obtained results using the labeled dataset and a more complex unlabeled one, comparing the results with the previous supervised approach and previous work that also does not require any previous knowledge about the model, demonstrating that the proposed solution was successful.

Finally, in Chapter 6, we conclude this work, analyzing how it overcomes previous work's major drawbacks and proposing future research opportunities that may improve the redundancy removal on 3D CAD models.

2
Related Work

In this Chapter, we discuss on previous works that could be used to remove redundant representation on 3D CAD models, subdividing into two groups regarding the use of deep learning techniques, highlighting their drawbacks and how our work overcomes them. Moreover, in Section 2.2, we also comment about previous work on other deep learning tasks, analyzing how it relates to our work.

2.1
Geometric Approaches

3D shape matching is an essential topic from the geometry processing point of view. Aiming to address this task, several approaches were developed to estimate a transformation between two corresponding point sets, *e.g.* mesh vertices, relying on the least squared minimization technique (Eggert et al., 1997; Horn, 1987; Kanatani, 1994; Umeyama, 1991). These approaches estimate rotations, rigid-body, or similarity transformations, which do not take non-uniform scaling into account. However, non-uniform scaling is commonly found between 3D CAD models repeated geometries. Thus, using such techniques would restrict the instances found, limiting the amount of redundant representation removed.

Works on symmetry analysis were also developed to perform shape matching, identifying similar 3D structures and computing transformations between them (Alt et al., 1988; Gal & Cohen-Or, 2006; Martinet et al., 2006; Mitra et al., 2006). Pauly et al., 2008, developed a framework that identifies a diversity of 3D structures on triangle meshes and point clouds, decomposing it into repeated regular structures and performing a registration procedure to obtain similarity transformations that can be used to instantiate the original geometry. Nevertheless, these works do not consider non-uniform scaling on the estimated transformation, and, as mentioned earlier, the model would not be as optimized as it could.

Regarding point cloud registration, the Iterative Closest Point (ICP) (Besl & McKay, 1992) is a well-known technique that iteratively performs a least square minimization procedure using estimated point set correspondence, obtaining a rigid-body transformation from one point cloud to another. Although researches were developed to improve the ICP robustness, by introducing other techniques into the algorithm (Granger & Pennec, 2002; Sharp et al., 2002; Silva et al., 2005), and also to obtain an uniform scaling registration (Zha et al., 2000; Zinßer et al., 2005), they do not consider non-uniform scaling.

Later, Du et al., 2010, presented the Scaling Iterative Closest Point (SICP). The authors introduced a scale matrix directly into the least square minimization step, successfully adapting the ICP algorithm to take non-uniform scaling into account, obtaining an affine transformations matrix, similar to our approach. However, as presented in Section 4.3.2.1, our proposed registration technique, using PCA combined with the Adam optimization, was able to obtain more accurate results while performing faster on average.

More recently, Santos & Celes Filho, 2014, proposed a least squares minimization approach to identify similar triangle meshes on a 3D CAD model that overcomes previous constraints. Their technique relies on a point set registration procedure that computes an affine transformation matrix which minimizes the squared distances between the corresponding vertices of an input triangle mesh and a reference mesh. Using this technique to instantiate repeated triangle meshes on 3D CAD models, the authors significantly reduced the memory required to represent a 3D CAD model (only 6% of its original size), while maintaining real-time rendering performance. However, their shape matching technique has two major drawbacks:

1. It depends heavily on the mesh topology and the vertex incidence order. In other words, visually identical meshes but with vertices in a different order, with different triangulation or discretization, are not recognized as repeated triangle meshes.

2. It only considers the mesh vertices, meaning that it may disregard important surface features derived from the triangulation.

Our proposed approach overcomes these restrictions by relying on dense point clouds that are uniformly sampled on the triangle meshes' surfaces. This makes the technique independent of the mesh triangulation and the vertices incidence order, while guaranteeing an upper bound on any surface errors. Therefore, we are able to identify a greater diversity of repeated triangle meshes under different scenarios.

2.2 Deep Learning-Based Approaches

The 3D shape matching task was addressed using deep learning-based techniques. Hanocka et al., 2018, designed a deep neural network that successfully aligns a source shape into a target one using Free-Form Deformation (FFD) grids, even when the target shape is only partial. In spite of the fact that the authors obtained accurate results, FFD grids are not efficient to render a large amount of instanced meshes. On the other hand, our proposed approach estimates matrix transformations, which are efficiently used by modern graphics hardware.

In the context of point cloud registration, several researches exploited the recent advances in deep learning to address this task (Kurobe et al., 2020; Wang & Solomon, 2019; Yew & Lee, 2018). Aoki et al., 2019, combined the PointNet (Qi et al., 2017a) network architecture with a modified Lucas & Kanade (LK) algorithm (Lucas, Kanade, et al., 1981) to estimate a rigid-body transformation that aligns an input point cloud into a target one. However, as mentioned earlier, non-uniform scaling is often encountered between repeated triangle meshes on 3D CAD models, and not considering it may limit the amount of instances found.

Other research relies on geometric deep learning techniques, which are further detailed, to segment and detect primitives (cones, cylinders, spheres, and planes), while also estimating their corresponding parameters, on point clouds (Li et al., 2019). The authors adapted the PointNet++ (Qi et al., 2017b) architecture, to compute the primitives' parameters while segmenting the points in the point cloud. Also in the context of primitive fitting, Friedrich et al., 2020, proposed a hybrid framework, similar to ours, to segment and fit primitives (cuboids, planes, spheres, and cylinders) on 3D point clouds. The hybrid framework relies on the PointNet++ and the RANSAC approach (Schnabel et al., 2007), combined with a genetic algorithm to generate solid primitives. Although the estimated parameters by these approaches could be used to instantiate repeated CAD geometries, the redundancy removal would be limited to primitives only, while our proposed approach uses triangle meshes, generalizing for any kind of geometry.

The point cloud classification and segmentation tasks benefited from recent deep learning advances, and several works proposed neural network architectures to address these tasks (Ben-Shabat et al., 2017, 2018; Gomez-Donoso et al., 2017; Hegde & Gangisetty, 2021; Hermosilla et al., 2018; Ravanbakhsh et al., 2016; Te et al., 2018; Wang et al., 2021; Wang et al., 2018; Wu et al., 2019). The PointNet (Qi et al., 2017a) architecture was a breakthrough solution to perform both supervised classification and segmentation on point clouds, being able to process raw unordered point clouds as input. Lately, the PointNet++ (Qi et al., 2017b) extended the PointNet architecture to hierarchically aggregate features in local point sets, obtaining robust classification and segmentation results. In our case, the 3D CAD models are already segmented into individual geometries; thus, in our supervised approach, we leverage the PointNet++ success to only classify them.

Other unsupervised and self-supervised tasks, such as point cloud representation, also benefited from recent deep learning advances. The representation learning aims to learn a data representation that facilitates other downstream tasks (Bengio et al., 2013), such as point cloud classification (Achlioptas et al., 2018; Hassani & Haley, 2019; Jiang et al., 2021; Rao et al., 2020; Remelli et al., 2019), segmentation (Hassani & Haley, 2019), semantic segmentation (Bachmann et al., 2021; Jiang et al., 2021), clustering (Remelli et al., 2019; Zamorski et al., 2020), up-sampling (Remelli et al., 2019) and reconstruction (Achlioptas et al., 2018; Bachmann et al., 2021; Zamorski et al., 2020). To address the point cloud representation learning task, these works usually combine an encoder with a decoder that aims to reconstruct the input, and, as a consequence, the encoder learns meaningful features to compactly represent the input point cloud. In our self-supervised approach, the goal is to cluster similar point clouds sampled on the meshes surfaces to reduce the redundant information on 3D CAD models. Therefore, we leverage this autoencoder approach to obtain point cloud feature vectors that can later be used to cluster the meshes found on the model.

3
Shape Matching Background

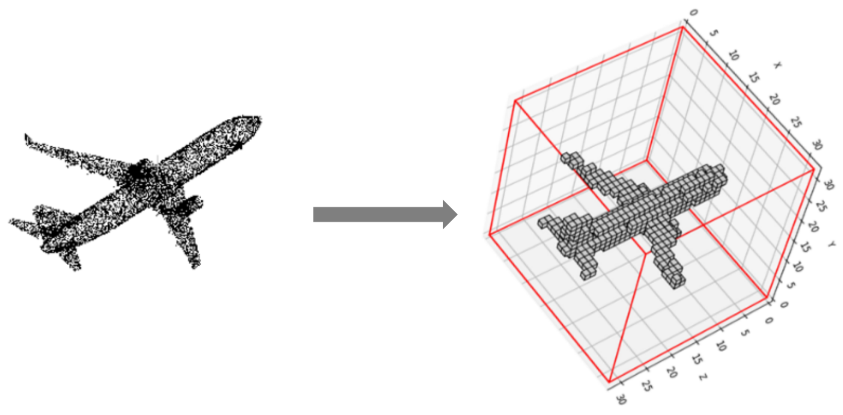
In this Chapter, we present some theoretical foundations for both the supervised and self-supervised proposed frameworks. First, we discuss the deep learning on point sets and its challenges, and detail the deep neural network that we used to perform the point cloud classification. Second, we introduce the autoencoder concept and how it can be used to extract feature vectors for an input point cloud. Third, we present the clustering algorithms used to group similar point clouds during the redundancy removal process.

3.1
Deep Learning on Point Sets

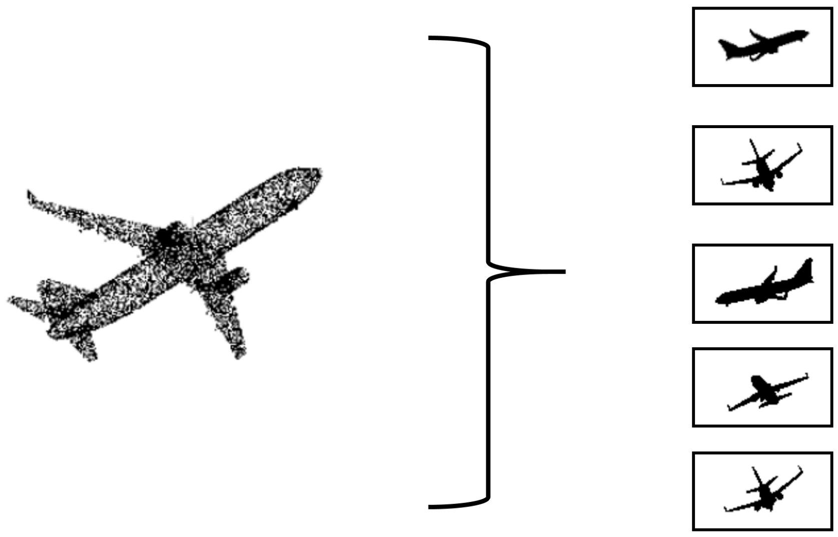
Point clouds are an important representation of 3D structures, being the closest representation of raw sensor data and also being canonical, once other types of representation, such as triangle meshes, can be easily translated into a point cloud. However, this type of representation has some challenges, especially for deep learning processing, since the data is unstructured, meaning that the distance from one point to another is not fixed like, for example, in regular grids, and it is also unordered, since the order in which the point set is stored does not change the represented scene or geometry.

Most of the earlier success of deep learning applications comes from convolutional neural networks (CNNs), which apply a convolution operation on structured and ordered data. To overcome these challenges, earlier approaches transformed the point clouds into volumetric occupancy grids or in a set of images from different points of view, as exemplified in Figure 3.1, enabling the application of the convolutional operator. Yet, these transformations produce an unnecessarily larger dataset, while also introducing quantization artifacts.

In this context, Qi et al., 2017a, introduced the PointNet, visualized in Figure 3.2, a deep neural network that processes raw unordered point clouds directly as an input. To achieve this, the authors combined two alignment networks with a simple symmetric function and also, in the segmentation network, concatenated the global point cloud features with the local point features. These characteristics respectively make the network invariant to rigid-body transformations, invariant to any input permutation, and also aggregate



(a) Example of volumetric occupancy grid of a point cloud. After the grid is obtained, volumetric CNNs may be used to classify the input. This image was adapted from Bello et al., 2020.



(b) Example of the multi-view approach to classify point clouds, where the object is rendered into multiple images and 2D CNNs are used to classify the object. Image adapted from Bello et al., 2020.

Figure 3.1: Example of earlier approaches to process point clouds.

local and global information, which is important to perform per-point tasks, such as point segmentation or point normal prediction.

However, the PointNet does not capture local structures derived from the space where the points are located, which limits its generalization capability and the ability to identify fine-grained patterns. The ability to capture local structures is an important characteristic for the success of CNNs. These networks progressively capture features along a multi-resolution hierarchy, abstracting local patterns, which allows better generalization to unseen data.

Aiming to address this issue, Qi et al., 2017b, extended the PointNet architecture, introducing the PointNet++, visualized in Figure 3.3. This extended architecture hierarchically aggregates local features by sampling and grouping the point set into local overlapping regions and using the PointNet as the local feature learner. For the sampling operation, the PointNet++ uses the iterative farthest point sampling to find centroids on the point set, obtaining a better coverage of the entire point set than random sampling with the same number of centroids, and, for the grouping operation, a ball query is used to find the neighborhood points of the centroid.

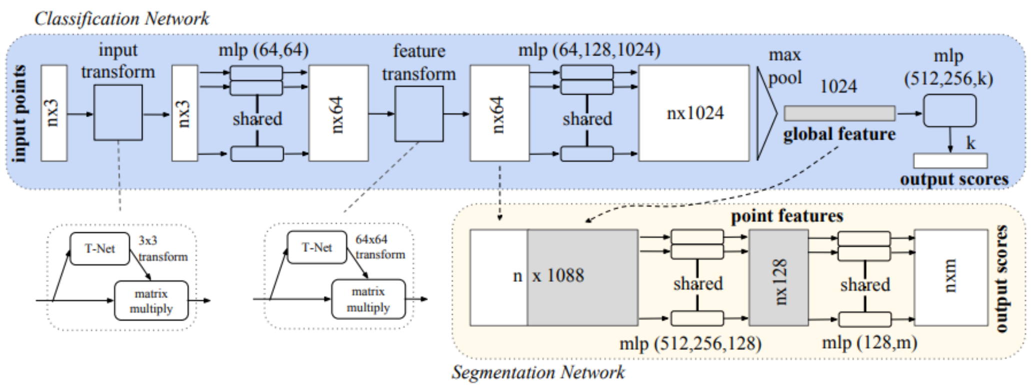


Figure 3.2: The PointNet architecture. First, point and feature transformations are applied to the input point cloud, then, the max pooling symmetric function is used to aggregate the point features. The classification network relies on the global feature vector to compute scores for k classes. The segmentation network concatenates the global feature vector with local features to compute per-point scores. Image adapted from Qi et al., 2017a.

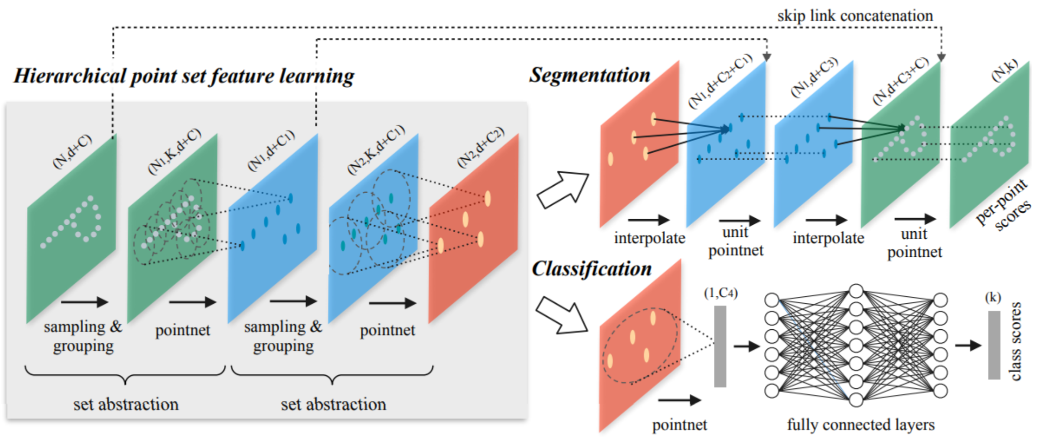


Figure 3.3: The PointNet++ architecture, which is composed of set abstraction layers that recursively sample, group, and obtain local features using the PointNet (Qi et al., 2017a). The classification network relies on the final feature vector to compute scores for k classes. The segmentation network recursively interpolates the feature values, concatenate with skip linked feature, and uses a unit PointNet, which adjusts the feature size, to compute per point scores. Image adapted from Qi et al., 2017b.

The PointNet++ architecture process point sets in a similar way to how CNNs process regular grid data, recursively capturing local geometric structures, which are further grouped and processed to obtain higher-level features. Using this powerful neural network, the authors were able to achieve state-of-the-art performance on the point cloud classification task.

3.2 Autoencoders

Rumelhart et al., 1985, introduced the autoencoder as a neural network that is trained to reconstruct its input, aiming to learn, in a self-supervised manner, a meaningful representation of the data. The autoencoder usually is composed of an encoder, a bottleneck, and a decoder, visualized in Figure 3.4. During the training process, the encoder learns to represent the input into the bottleneck structure, usually smaller than the input, obtaining a compact meaningful representation of the input, while the decoder learns to reconstruct the input using this compact representation.

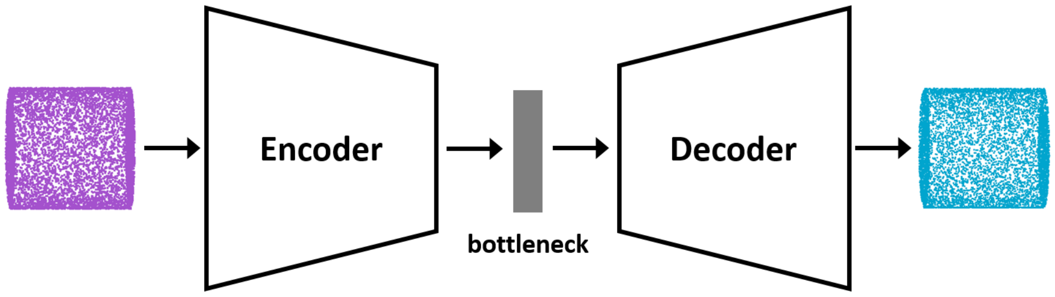


Figure 3.4: Example of an autoencoder for point clouds. The encoder receives an input point cloud and produces a compressed representation of it, while the decoder uses this compressed representation to reconstruct the input point cloud.

This autoencoder architecture may be used in tasks such as semi-supervised classification and unsupervised clustering. In the classification task, this approach is useful when large datasets are provided with only a small portion of labeled examples, where first, the autoencoder is trained to learn a meaningful representation of the data, and then, the decoder is replaced with a classification network, which is further fine-tuned using the labeled data. In the clustering task, the autoencoder may be used to obtain a compact meaningful representation, or feature vector, for each instance in the dataset, which can be further used as input to traditional clustering algorithms.

3.3 Clustering

Clustering is an essential task in the unsupervised learning research field, aiming to find an underlying structure in unlabeled datasets by grouping similar instances into the same cluster while maintaining different instances apart. One possible classification of the clustering algorithms is into partitional and hierarchical. The partitional clustering algorithms, as the name suggests, partition the data into a set of clusters with no hierarchical structure, where each cluster is represented by a centroid or a cluster representative, by iterative reallocating the instances in the set of clusters until convergence is achieved. On the other hand, the hierarchical clustering algorithms group the data successively, using previous clusters to find new ones in both top-down and bottom-up strategies. In the following sections, we describe further details on the clustering algorithms used in this work, the *K-Means* algorithm, which is partitional, and the HDBSCAN, which is hierarchical.

3.3.1 *K-Means*

The *K-Means* is a simple well-known clustering algorithm that assigns each point, or feature vector, into one of the K clusters that are desired as output, minimizing the total squared Euclidean distance between each point and its nearest center. To achieve this, first, a set of K centroids are randomly selected; second, each input point is assigned to the nearest centroid cluster; third, after assigning all the points in the dataset, the clusters centroids are updated using the average of its members; and finally, the second and third steps are repeated until there are no changes in all the clusters. This procedure is summarized in Algorithm 1.

Algorithm 1 *K-Means*

```
    Input:
        $K$: Number of clusters
    Output:
        $C$: Set of $K$ clusters

1: Randomly select $K$ initial centroids $\mathcal{C} = \{c_1, c_2, \dots, c_k\}$
2: while $\mathcal{C}$ is changing do
3:     for $i \leftarrow 1$ to $k$ do
4:         $C_i \leftarrow$ points closer to $c_i$ than to $c_j$ for all $j \neq i$
5:     for $i \leftarrow 1$ to $k$ do
6:         Update centroids: $c_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$
```

Although *K-Means* is extremely popular due to its simplicity, it has some drawbacks, such as how to best define the number of clusters, which is an input parameter of the algorithm, how to best define the initial cluster's centroids, and also, there is no guarantee that a global optimal partition is achieved. Aiming to define a better initialization of the cluster's centroids, Arthur & Vassilvitskii, 2006, proposed the *K-Means++*, which, instead of sampling K initial centroids with uniform probability, it selects the initial centroids with a probability that is proportional to the squared distance to its nearest cluster centroid, as shown in the third step of Algorithm 2. This initialization procedure is summarized in Algorithm 2.

Algorithm 2 *K-Means++*

```
    $X$: Set of data points.
    $d(x)$: Distance from a point $x$ to its nearest cluster center.

1: $\mathcal{C} \leftarrow x \in X$ randomly selected with uniform probability
2: while $|\mathcal{C}| < K$ do
3:     Select $x \in X$ with probability $\frac{1}{\sum_{x \in X} d(x)^2} d(x)^2$
4:     $\mathcal{C} \leftarrow \mathcal{C} \cup \{x\}$
5: Continue the K-Means execution
```

Using the *K-Means++* initialization, the authors improved the clustering, getting it closer to its global optimum, while also performing faster than the traditional *K-Means*. More recently, Bahmani et al., 2012, proposed the *K-Means*||, a parallel version of the *K-Means++*, improving its scalability and performance, while maintaining the clustering quality at least as good as previous methods. To achieve this, instead of sampling one cluster center at a time during the initialization, an oversampling factor is used to sample more potential center clusters at once, which are further reclustered to find the initial clusters centers, using the *K-Means++* with weights equal to number of points in X closer to the potential center.

Other research addressed the question of how to estimate the desired number of clusters K. The ISODATA algorithm (Ball & Hall, 1967), for example, is very similar to the standard *K-Means*; however, it dynamically updates the value of K during the iteration, merging and splitting clusters based on its similarity and standard deviation, using the updated K for the next iterations. On the other hand, aiming to obtain the parameter K which best cluster a dataset, Pelleg, Moore, et al., 2000, proposed the *X-Means*, which executes the *K-Means* for a range of possible number of clusters, choosing the one with best Bayesian Information Criterion (BIC) (Kass & Wasserman, 1995). Hamerly & Elkan, 2003, developed yet another approach, the *G-Means* algorithm, which runs the *K-Means* for a small initial number of K, and, at each iteration, split the clusters that do not follow a Gaussian distribution, stopping when no additional clusters are required.

3.3.2 HDBSCAN

Hierarchical Density-Based Spatial Clustering of Applications with Noise, or HDBSCAN (Campello et al., 2013), extends the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) (Ester et al., 1996), to make it hierarchical. The HDBSCAN does not suffer from previous drawbacks of previous density-based clustering algorithms, such as using a global density threshold, which does not properly describe datasets with varying clusters densities, not automatically simplifying the clustering hierarchy into a representation that involves only the most significant clusters, or depending on multiple input parameters.

Using $X = \{x_1, x_2, \ldots, x_n\}$ as the set of n points to be clustered, the algorithm receives as input the number of points m_{pts} which is used to determine if a point $x_p \in X$ is a core point, where x_p is a core point if in its neighborhood with distance ϵ contains at least m_{pts} including the point itself. Relying on this concept of core points, and the concept of core distance, which is the distance from the point x_p to its m_{pts}-nearest neighbor, the HDBSCAN executes five main steps:

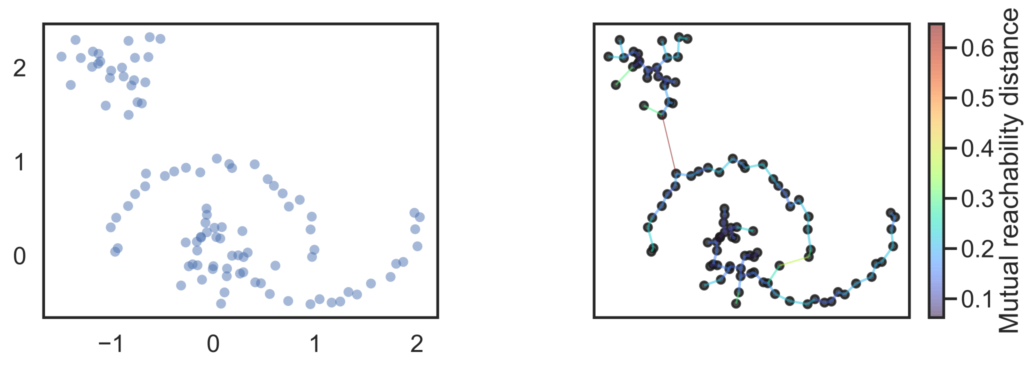
1. Compute the core distance for all $x \in X$;
2. Build a Minimum Spanning Tree (MST) using the mutual reachability distance (Equation 3-1);
3. Extract the HDBSCAN hierarchy from the MST;
4. Condense the HDBSCAN hierarchy based on the minimum cluster size;
5. Obtain the stable clusters from the condensed hierarchy.

In order to compute the core distance, several distance metrics may be used; however, the Euclidean distance is commonly used. Once the core distances are computed using the Euclidean distance, to compute the MST, first, the mutual reachability graph is constructed, using each $x \in X$ as vertices and the edge's weights are computed using the mutual reachability distance:

$$d_{mreach}(x_i, x_j) = max(d_{core}(x_i), d_{core}(x_j), d(x_i, x_j)) \tag{3-1}$$

where x_i and x_j are points of X, d_{mreach} is the mutual reachability distance, d_{core} is the core distance and d is the distance metric used; in this case, the

Euclidean distance. After computing the mutual reachability graph, the MST is constructed using Prim's algorithm (Prim, 1957). This process is exemplified in Figure 3.5.



(a) Data points to be clustered. (b) Minimum Spanning Tree.

Figure 3.5: Example of a dataset to be clustered using the HDBSCAN and the computed MST using $m_{pts} = 2$.

Having the MST computed, the HDBSCAN hierarchy can be extracted by, firstly, assigning all points to the same cluster, and then, iteratively removing the edges in a decreasing order of weights, assigning new cluster labels to the connected components that contain the vertices of the removed edges if they maintained at least one edge, and assigning as noise if they do not. The HDBSCAN hierarchy of the example in Figure 3.5b is visualized as a dendrogram in Figure 3.6.

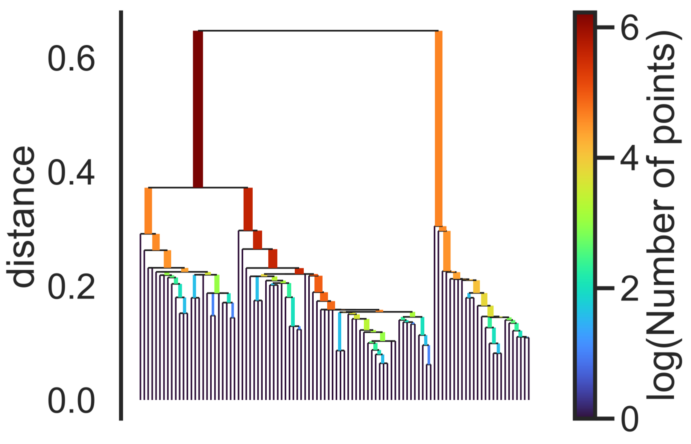


Figure 3.6: Visualization of the HDBSCAN hierarchy of the example in Figure 3.5b as a dendrogram.

At this stage, the HDBSCAN hierarchy tree contains all partitions that could be obtained using the DBSCAN, however, in a hierarchical way. In practice, the DBSCAN would perform a horizontal cut in the dendrogram at a global distance scale ϵ, obtaining the clusters and interpreting any singleton cluster as noise. On the other hand, the HDBSCAN continues processing to obtain clusters with varying densities, allowing cuts at different levels of the hierarchy to obtain the most significant clusters.

To achieve this, a condensed HDBSCAN hierarchy is computed instead, using the notion of minimum cluster size, m_{clSize} , an optional parameter that can be simplified, being equal to m_{pts}. During the computation of the condensed HDBSCAN hierarchy, components whose size is smaller than m_{clSize} are not considered as a true cluster split when disconnected from a cluster. Using this new strategy, after an edge removal, the resulting connected sub-components can be: (i) noise, if they are smaller than m_{clSize}, (ii) same cluster, if only one sub-component greater than m_{clSize} is obtained and (iii) new clusters, if two or more sub-components greater than m_{clSize} are obtained. After performing this process, and using $\lambda = \frac{1}{\epsilon}$ as a cluster persistence metric, we obtain the condensed HDBSCAN hierarchy visualized in Figure 3.7.

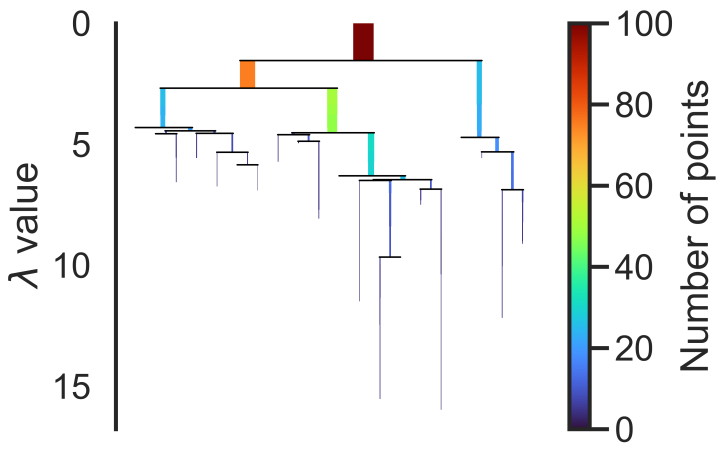
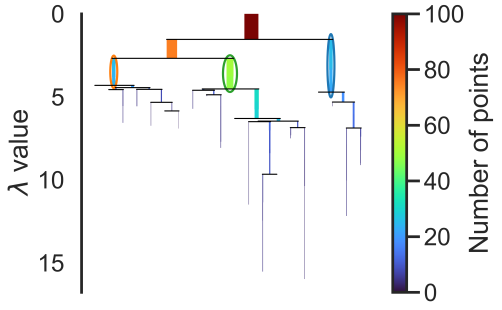


Figure 3.7: Visualization of the condensed HDBSCAN hierarchy of the example in Figure 3.5.

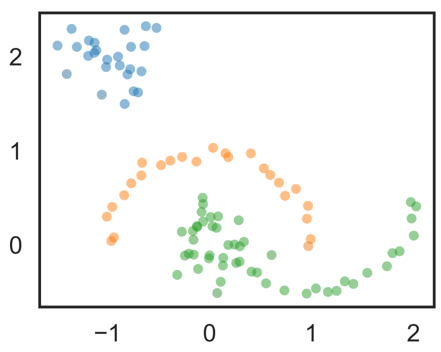
Finally, using the clusters' stability as:

$$\sum_{x \in cluster} (\lambda_x - \lambda_{min}) \tag{3-2}$$

where λ_x is the λ value in which the point x was removed from the cluster and λ_{min} is the value in which the cluster appeared, combined with the restriction that, if a cluster is selected, none of its descendants can be selected, the most stable and significant clusters can be obtained from the condensed HDBSCAN hierarchy. To perform this task, a bottom-up approach is used, in which, first, all leaf nodes are declared as selected clusters, then, if a cluster stability is greater than the sum of its children's stability, the cluster is selected and its descendants are deselected, and if it is smaller, the cluster stability is set as the sum of its children's stability. When the root is achieved, the current set of selected clusters is used as the clustering. This final result is shown in Figure 3.8.



(a) Selected clusters in the condensed HDBSCAN hierarchy.



(b) Data points color-coded according to the clustering result.

Figure 3.8: Cluster selection of the example in Figure 3.5. Although there exists a horizontal cut that could be used to select these clusters, this is not a limitation of the HDBSCAN algorithm, which, using the clusters stability to select them, can choose clusters that are in different positions in the condensed HDBSCAN hierarchy.

Furthermore, the strength cluster membership of each point $x \in cluster$, which is an important metric for some applications, can also be computed. To compute this metric, for each cluster, all of the λ_x for the points inside the cluster are normalized between $[0, 1]$, meaning that points that remained for a longer time in the cluster have greater strength membership than the ones that were removed earlier.

4
Supervised Shape Instance Matching Framework

In this Chapter, we detail the proposed supervised framework for the shape matching problem on 3D CAD models, present the results obtained using the proposed framework and discuss on them. Our objective is to obtain a shape matching solution to minimize the redundancy representation of the CAD models, which overcomes previous work drawbacks: considering only primitives during the redundancy removal process, being heavily dependent on the mesh topology and vertices incidence order, and obtaining a registration transformation that is not well suited for the 3D CAD domain. To address these drawbacks, the proposed framework relies on point clouds uniformly sampled on generic meshes surfaces, which makes it independent of the mesh topology and vertices ordering while also generalizing for any kind of geometry. The proposed framework also uses a registration procedure that obtains an affine transformation matrix, which is well suited for the 3D CAD domain, once it takes into account non-uniform scaling while enables the use of hardware-accelerated API for rendering (Pharr & Fernando, 2005).

The proposed supervised shape matching framework, illustrated in Figure 4.1, takes a single triangle mesh as input, depicted as a purple cylinder, and first executes a preprocessing step. It is important to mention that the 3D CAD models are composed of multiple triangle meshes, which are processed individually by the framework. This preprocessing step is composed of two main steps, the normalization, and the uniform point cloud sampling. In the normalization step, the center of mass of the triangle mesh is translated to the origin, $[0, 0, 0]$; then, its principal components are aligned with the 3D axis X, Y, Z, and finally, the triangle mesh is scaled into the $[-0.5, 0.5]$ range. Once the mesh is normalized, a dense and uniform point cloud $\mathbf{P} \in \mathbb{R}^{N\times 3}$ is sampled on its surface.

After completing the preprocessing step, the framework relies on the PointNet++ (Qi et al., 2017b) deep neural network to classify the point cloud $\mathbf{P}$ into one of the previously known reference types, which have a normalized triangle mesh associated. To choose the predicted reference type, we select the one with the highest score among PointNet++ output. We highlight that the supervised framework supports any number of reference types, as long as they are used to train the network.

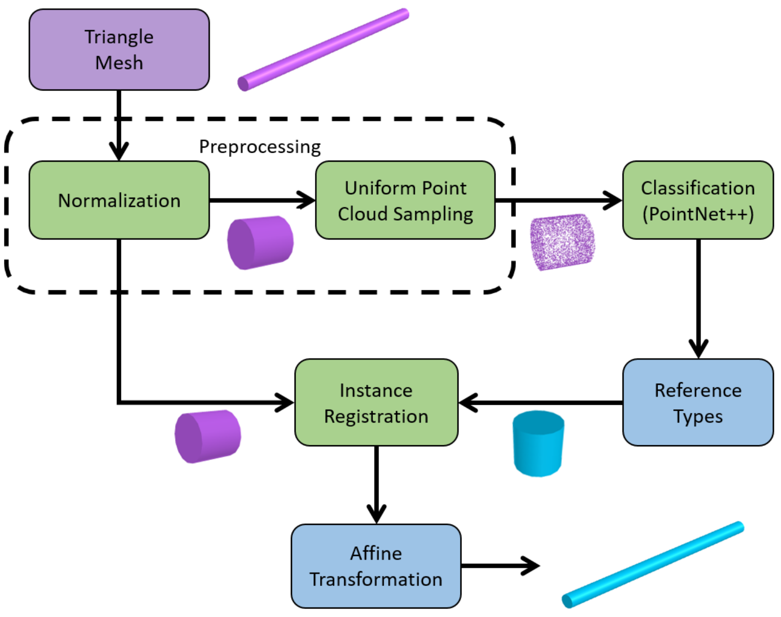


Figure 4.1: Overview of the proposed supervised framework, which is composed of three main steps: the preprocessing, the classification and the instance registration.

Having the reference type identified by the classification step, the instance registration procedure is executed. This procedure relies on point clouds sampled on the reference mesh (the source) and the normalized input mesh (the target), to perform a point cloud registration that estimates a matrix $M^{3\times3}$, that is combined with the inverse transformations used to normalize the input mesh to obtain an instancing matrix $M^{3\times4}$. We chose to represent the instancing matrix in this manner, encoding the scale, rotation, and translation, because only an affine transformation is required and not projective ones. This instancing matrix, combined with the reference mesh, acts as a compact representation of the input mesh, reducing the 3D CAD model size and enabling the instance rendering technique.

In the following sections, we further detail the preprocessing and the instance registration steps. We also present the performed experiments on a labeled dataset, describing how the PointNet++ was trained and discussing the obtained results.

4.1 Preprocessing

The first step in the preprocessing stage is the normalization of the input triangle mesh. To obtain a normalized triangle mesh, first, we translated the center of mass of the input to the origin, subtracting the mesh's arithmetic mean, V_{mean}, from the vertices coordinates. In the second operation, inspired by Souza Moreira, 2015, we align the mesh with the 3D axis X, Y, Z. To perform this alignment, first, the principal components of the mesh's vertices are obtained using the Principal Component Analysis (PCA) (Wold et al., 1987) method, then an orthonormal basis is defined using the eigenvectors of the vertices' covariance matrix. These eigenvectors correspond to the directions in which the vertex positions vary the most. To finalize the mesh alignment, each vertex is transformed using the inverse rotation matrix, M_{ev}^{-1}, which is defined by the orthonormal basis. It is important to highlight that, for better alignment, the density of points on the surface should be uniform, which is not guaranteed when the mesh's vertices are used; however, since the reference meshes found on the labeled dataset are well behaved, this was not an issue, and if that was not the case, a uniform point cloud sampled on the mesh's surface could be used.

In the final operation of the normalization procedure, we scale each coordinate of the mesh's vertices into the $[-0.5, 0.5]$ range, dividing it by the difference between each axis's maximum and minimum values. This scaling procedure applies a non-uniform scale to the mesh, since its vertices coordinates are scaled separately. In this context, the previous PCA-based alignment is used as an attempt to minimize unwanted shearing. In order to obtain the final instancing matrix, while the normalization is executed, we store the translation V_{mean}, the PCA orthonormal matrix M_{ev}, and the non-uniform scaling matrix, to denormalize the reference mesh after the registration procedure.

It is worth mentioning that the primary goal of the normalization procedure is to increase the similarity between meshes that represent the same 3D shape, facilitating the identification of the reference type, since shapes with non-uniform scaling are commonly found in the 3D CAD models. For example, a long cylinder and a flattened one become very similar after this normalization procedure. It is also worth mentioning that, since the PCA eigenvectors may not match exactly, after the normalization, even though the triangle meshes belong to the same reference type, they may not be aligned in the same direction.

Having the triangle mesh normalized, the next step in the preprocessing stage is to sample a dense and uniform point cloud on its surface. To perform this, we use the approach developed by Osada et al., 2002, which estimates the probability of a random point to be sampled on a triangle as being proportional to its area. This relation is obtained by dividing each triangle area by the total area of the triangles that represent the mesh, and performing a cumulative sum on the result, obtaining an interval in the $[0, 1]$ range, that contains smaller intervals proportional to the associated triangle area. Then, a random number in the $[0, 1]$ interval is generated, and its associated triangle is selected. To compute the point position $\mathbf{p}$ on the selected triangle $(\mathbf{a}, \mathbf{b}, \mathbf{c})$, two additional random numbers between 0 and 1 are generated, r_1, r_2, and the Equation 4-1 is used to obtain the point coordinates. This procedure is executed until the desired point cloud density is obtained. We highlight that, even if similar meshes have different discretization, this procedure obtains similar uniform point clouds for them.

$$\mathbf{p} = (1 - \sqrt{r_1})\mathbf{a} + \sqrt{r_1}(1 - r_2)\mathbf{b} + \sqrt{r_1}r_2\mathbf{c} \tag{4-1}$$

4.2 Instance Registration

The main objective of the instance registration step is to estimate a matrix M that can transform a source mesh, $\mathcal{S}$, into a target mesh, $\mathcal{T}$. To perform the registration, we use the mesh associated with the reference type predicted by the classification step as the source and the normalized input triangle mesh as the target. Both source and target meshes are illustrated in Figure 4.2. To estimate the matrix M, we developed an optimization procedure inspired in the Point2Mesh (Hanocka et al., 2020). The Point2Mesh relies on a deep neural network to shrink-wrap a point cloud with a surface by iteratively deforming an initial triangle mesh. Similarly, our optimization procedure relies on the Adam (Kingma & Ba, 2014) optimizer to iteratively update an initial matrix in order to transform $\mathcal{S}$ into $\mathcal{T}$, using uniform point clouds sampled on them.

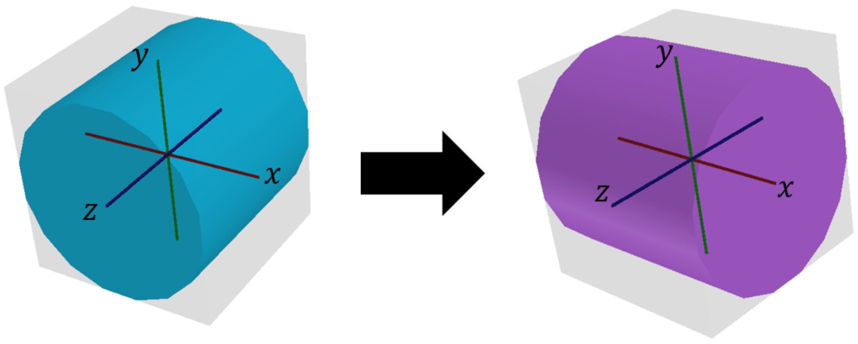


Figure 4.2: Example of input to the instance registration step. The blue cylinder represents the reference mesh (source), and the purple one the input normalized triangle mesh (target). The main objective is to estimate a matrix that transforms the source mesh into the target one. Although the meshes may look the same, this process is still required, since they are aligned on different axis.

To start the registration, first, a uniform point cloud $\mathbf{Q}$ is sampled on $\mathcal{T}$. The point cloud $\mathbf{Q}$ is used to compute the surface error of $\mathcal{S}$ transformed by the matrix M using the following Equation:

$$E_s = \frac{\sum_{\mathbf{q}\in\mathbf{Q}} D_{min}(\mathbf{q}, \mathcal{S} * M)}{N_Q} \tag{4-2}$$

where E_s is the surface error, D_{min} is the minimum Euclidean distance from a point to a mesh, and N_Q is the number of points in $\mathbf{Q}$. We highlight that, to compute E_s, we sample the point cloud on $\mathcal{T}$ instead of $\mathcal{S}$, because it does not change during the optimization procedure, while $\mathcal{S}$ is continuously changing to match $\mathcal{T}$. It is also worth mentioning that E_s is a scale-invariant metric, since both $\mathbf{S}$ and $\mathcal{T}$ are normalized.

Defining α_1, as the threshold for the desired maximum surface error, first, we check if $E_s \leq \alpha_1$, and, if so, no optimization is needed and M is set to identity, otherwise, we execute the iterative optimization procedure, which has two nested iteration loops, defined as outer and inner loop. In the outer loop, visualized in Figure 4.3, for IT_o iterations, we sample the uniform point clouds $\mathbf{P_S}$ and $\mathbf{P_T}$, each containing N_P points, respectively on $\mathcal{S}$ transformed by the current M and $\mathcal{T}$. Then, in order to facilitate the next steps, we transform $\mathbf{P_S}$ with the inverse of the current matrix, allowing M to be continuously updated. The outer loop may be interrupted if $E_s \leq \alpha_1$, or if the density of the point clouds, N_P, was incremented for $\mathbf{i}$ times.

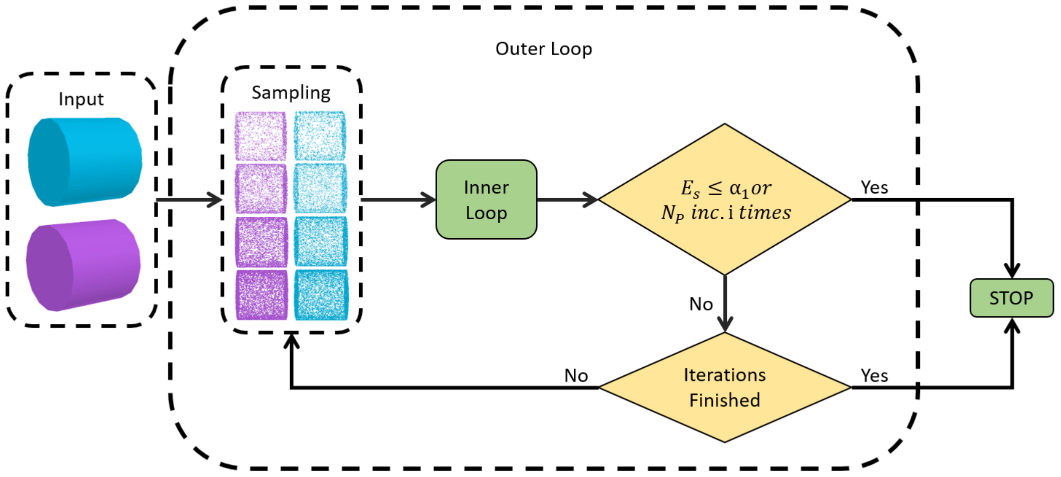


Figure 4.3: Scheme of the outer loop taking as input $\mathcal{S}$ and $\mathcal{T}$, as the blue and purple meshes, respectively. On the sampling step, $\mathbf{P_S}$ and $\mathbf{P_T}$ are obtained with density equal to N_P, and used as input to the inner loop. After the inner loop finishes, the interruption conditions are verified, continuing the iterations if none of them was satisfied.

We can interpret the outer loop as a control over the optimization, and it has three objectives:

1. Interrupt the optimization when a desired surface error is achieved or when it gets stuck in a local minimum, avoiding unnecessary computations;

2. Update the point clouds used in the inner loop, increasing their density when the inner loop stops reducing the error metric;

3. Maintain $\mathbf{P_S}$ uniform when transformed by M. Note that the area of the triangles may change while the matrix is updated.

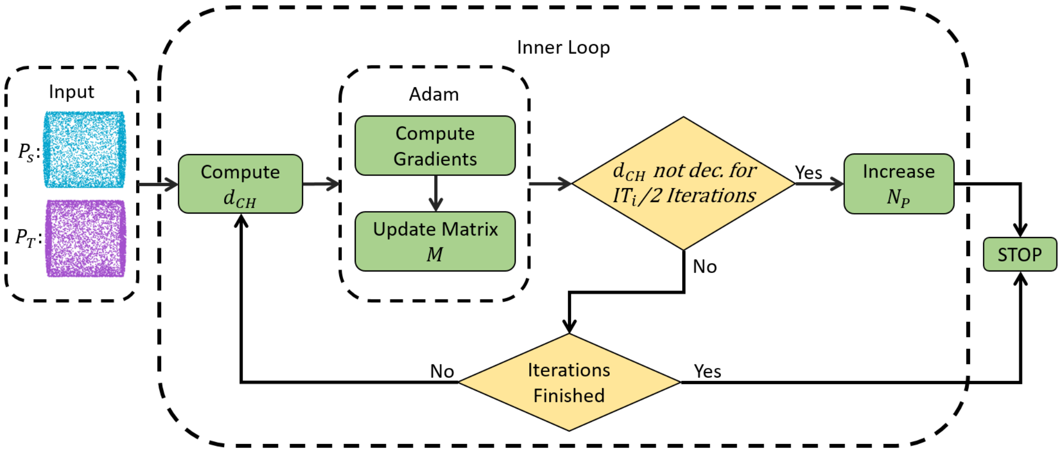


Figure 4.4: Scheme of the inner loop registration procedure taking as input $\mathbf{P_S}$ transformed by M and $\mathbf{P_T}$. The computed d_{CH} is used as cost function by the Adam optimizer to compute the gradients and update M. The optimization procedure is interrupted when the number of iterations is finished or when the cost function does not decrease for $IT_i/2$ iterations, which is a heuristic used to prevent excessive computational time on an optimization that is not improving the cost function.

In the inner loop, visualized in Figure 4.4, the point cloud registration is actually performed. Defining IT_i as the maximum number of iterations on the inner loop, first, we compute the Chamfer (pseudo) distance d_{CH}:

$$d_{CH} = \sum_{x \in X} \min_{y \in Y} ||x - y||_2 + \sum_{y \in Y} \min_{x \in X} ||x - y||_2 \tag{4-3}$$

which is fast and differentiable (Fan et al., 2017). Setting $\mathbf{P_T}$ as X and $\mathbf{P_S} * M$ as Y, we use it as a cost function on which the Adam optimizer computes the gradients and updates M. After the matrix was updated, we check if the cost function d_{CH} has improved (decreased), interrupting the optimization if it has not decreased for $IT_i/2$ iterations. If the optimization was interrupted, we also increment the point clouds density N_p. During the execution of the inner loop, $IT_i/2$ is a heuristic used to avoid unnecessary computations when the optimization is not decreasing d_{CH}.

On early experimentation, we observed that, sometimes, the optimization was getting caught in local minima, and the final registration error could be further improved. In order to improve the optimization procedure, we used the PCA on uniformly-sampled point clouds on both $\mathcal{S}$ and $\mathcal{T}$ to obtain their orthonormal basis. These orthonormal basis can be interpreted as rotations, which we combine, using the Equation 4-4, to initialize M before the whole procedure. In Equation 4-4, PCA_{INIT} is the combined rotation, $B_{\mathcal{S}}^{-1}$ is the inverse of the source rotation and $B_{\mathcal{T}}$ is the target rotation.

$$PCA_{INIT} = B_{\mathcal{T}} * B_{\mathcal{S}}^{-1} \tag{4-4}$$

In Section 4.3.3, we present results showing that, initializing M with the combined rotation PCA_{INIT}, reduced the final optimization error while also improving the average optimization time. The complete optimization procedure is summarized in Algorithm 3.

Algorithm 3 Complete optimization procedure

```
    Input:
        $\mathcal{S}$: Source mesh
        $\mathcal{T}$: Target mesh
    Output:
        $M$: Matrix that transforms $\mathcal{S}$ into $\mathcal{T}$.

 1: if $E_s <= \alpha_1$ then
 2:     Set $M$ to identity
 3: else
 4:     Set $M$ to $PCA_{INIT}$
 5:     for $n \leftarrow 0$ to $IT_o$ do
 6:         Sample $\mathbf{P_S}$ and $\mathbf{P_T}$ with $N_p$ points
 7:         Set $\mathbf{P_S} = \mathbf{P_S} * M^{-1}$
 8:         for $k \leftarrow 0$ to $IT_i$ do
 9:             Compute $d_{CH}(\mathbf{P_T}, \mathbf{P_S} * M)$
10:             Use $d_{CH}$ to update $M$ using the Adam optimizer
11:             if $d_{CH}$ has not decreased for $IT_i/2$ iterations then
12:                 Stop and increment $N_p$
13:             else
14:                 Continue
15:         if $E_s <= \alpha_1$ or $N_P$ incremented i times then
16:             Stop
17:         else
18:             Continue
```

Also during early experimentation, we observed that, even though the desired maximum surface error α_1 was not reached, in some cases, the resulting mesh still had a good visual quality. However, using a more relaxed α_1, we may interrupt the optimization procedure on instances that could still improve. In this context, we implemented one last validation step, in which we reject the instances with final surface error greater than a minimum acceptance error α_2, and consider them as unique shapes, maintaining their original triangle mesh on the final 3D CAD model. This last validation step is used to guarantee the original mesh quality, avoiding misclassified meshes being accepted as instances, while also avoiding unwanted deformations that could occur when the optimization becomes trapped in a bad local minimum. We highlight that the instance registration algorithm has full control over any surface error that may result from the registration procedure, guaranteeing an upper bound on any surface error introduced in the final 3D CAD model.

Once the complete optimization procedure is finished, and the instance was accepted by the last validation step, M is able to transform the source mesh $\mathcal{S}$ into the target mesh $\mathcal{T}$. However, to obtain the final instancing matrix that represents the input triangle mesh using the reference mesh, we need to reverse the normalization performed in the preprocessing step. Defining the inverse of the translation used to set the mesh's center of mass on the origin as T^{-1}, the rotation obtained using the PCA orthonormal basis as M_{ev}, and the inverse matrix of the non-uniform scaling used to scale the vertices in the range $[-0.5, 0.5]$ as S^{-1}, we obtain the final instancing matrix, M_{final}, using the following Equation:

$$M_{final} = T^{-1} * M_{ev} * S^{-1} * M \tag{4-5}$$

4.3 Experiments

To evaluate the proposed supervised framework, we used a real-world 3D CAD model (Figure 4.5) composed by 18851 triangle meshes labeled in 16 different reference types (Figure 4.6) as the dataset. We highlight that, even though some reference meshes may look similar, we labeled them using the restriction that an affine transformation is able to instantiate all meshes from the same type using the reference mesh. In other words, a conic and toroid meshes, for example, only differ by an affine transformation matrix if both instance and reference mesh have proportional attributes, such as the radii. By further analyzing Figure 4.6, we notice that the reference meshes present symmetries, and, although the 3D shapes found on CAD models usually have this characteristic, this is not related to our proposed solution, since we design the framework to process any kind of geometry.

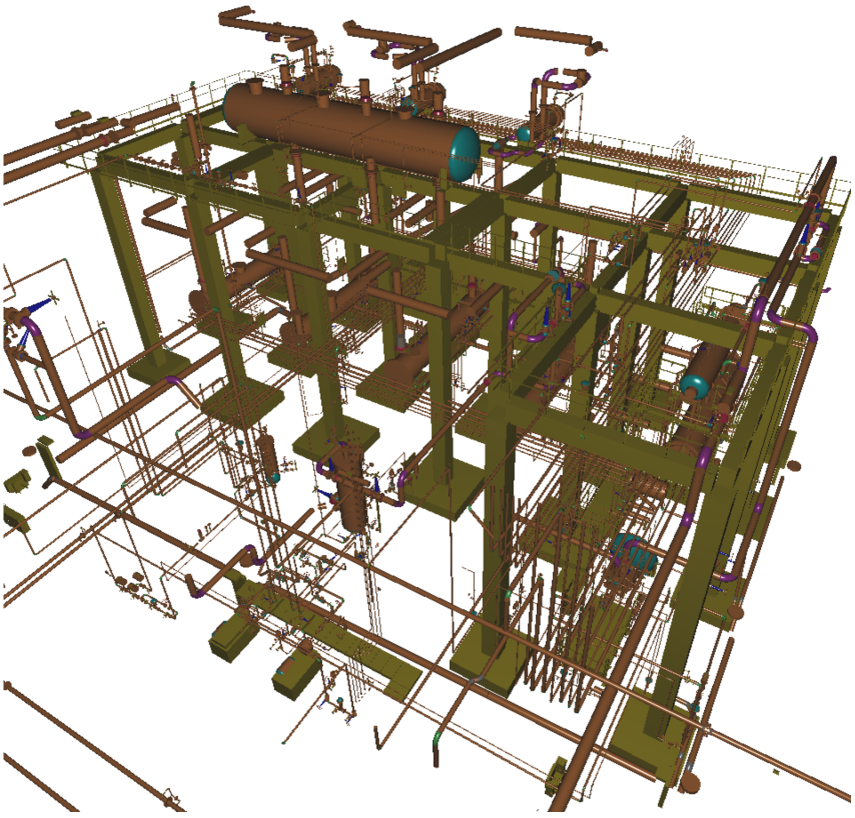


Figure 4.5: Real-world 3D CAD model used as dataset to evaluate the proposed supervised framework. The model is color-coded according to each mesh reference type.

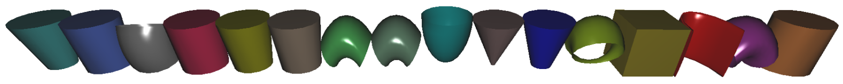


Figure 4.6: Visualization of the normalized reference meshes for each of the 16 reference types found on the dataset. These reference meshes represent cylinder, box, semi-sphere, cones, and circular toroids with different attributes proportions.

To measure the performance of our supervised solution, we implemented two metrics to evaluate the classification step, the accuracy and the $F_1 score$, detailed, respectively, in Equation 4-6, where J is the number of instances, t_j is the ground truth, and $\hat{t}_j$ is the predicted reference type, and Equation 4-7, where TP is the true positives, FP is the false positives, and FN is the false negatives. The accuracy is used to measure how the classification model is generally performing, while the $F_1 score$ is the harmonic mean between the precision, which measures the quality of the classification model, and the recall, which measures the quantity of the retrieved examples. The $F_1 score$ can be computed for each class individually, is in the $[0, 1]$ range, and, as closer to 1, the better.

$$Accuracy = \frac{\sum_{j=1}^{J} 1(t_j = \hat{t}_j)}{J}. \tag{4-6}$$

$$F_1 score = \frac{2 * TP}{2 * TP + FP + FN}. \tag{4-7}$$

We also implemented two metrics to evaluate the quality of the optimized CAD model, the mean surface error $MeanE_s$, which measure the overall quality of the CAD model with the instancing matrices estimated by the instance registration, and standard deviation surface error, $StdE_s$, which indicates how the surface error is varying. We highlight that both $MeanE_s$ and $StdE_s$ are scale invariant, since the instances' surface errors are scale invariant.

4.3.1 Training the PointNet++

To execute the supervised framework on the dataset, first, we trained the PointNet++ network using the reference types found on the dataset CAD model. In order to train the network, we split each set of reference types evenly into training, validation, and test sets, meaning that each reference type is properly represented in each set. Observing Figure 4.7 we notice that our dataset is unbalanced, which can worsen the classification results. To address this issue, we used a data augmentation routine to balance the training set, translating each triangle mesh to the origin and applying random uniform scales, rotations, and translations. We performed the augmentation until the set was balanced, containing 10^4 triangle meshes of each class.

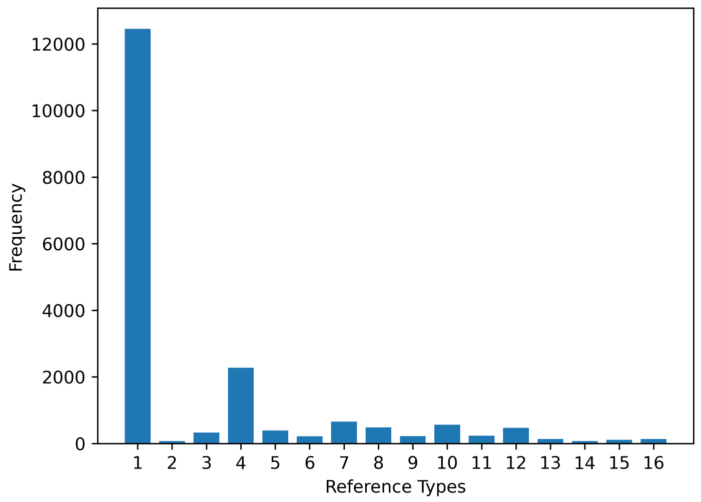


Figure 4.7: Frequency of each reference type found in the dataset.

Using this data augmentation routine, we ensure that the developed normalization procedure makes our proposed supervised framework, especially on the classification step, invariant to scale, rotation, and translation, by increasing the amount of triangle meshes in each reference type set with different transformations. We highlight that, although the normalization procedure minimizes such effect, or eliminates it, and the final training set contains sets of duplicated normalized meshes, the obtained point clouds are different even for identical meshes, since each of them was individually sampled using random parameters.

After we performed the augmentation, and sampled uniformly point clouds on each normalized mesh using the sampling described in Section 4.1, we trained the PointNet++ over 100 epochs with the cross entropy loss function, using a learning rate of $1e-3$. It is worth mentioning that we used an unbalanced validation set, since each reference type was evenly split into training, validation, and test sets, and only the training set was balanced using the augmentation routine. This is shown in Table 4.1, in which we observe that reference type 1 has significantly more triangle meshes than the others.

Table 4.1: Number of triangle meshes of each reference type in the validation set.

Reference type	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number of meshes	4151	25	109	758	131	72	219	162	75	189	79	158	45	24	37	44

Once the training was completed, we selected the epoch that obtained the highest accuracy on the validation set, resulting in a classification accuracy of 99.98% on the test set, and a $F_1score = 1.0$ for almost all reference types, with worst-case on type 6, which was 0.9931. These results show that the classification step correctly classified most of the triangle meshes. The augmentation routine was essential to such performance, especially if we analyze the F_1score. For comparison, when we trained the network using the unbalanced training set, we obtained $F_1score < 1.0$ for many reference types, with a worst-case on type 2 being equal to 0.3333.

To further inspect the test set results, we computed the confusion matrix, which summarizes the performance of the classification on each reference type. Analyzing Table 4.2, we can observe that the PointNet++ predicted most of the triangle meshes correctly, missing only one on reference type 8, which it predicted as reference type 6, representing one false negative for shape class 8, and a false positive for shape class 6. We highlight that the last validation step, using the final surface error, is designed to reject these misclassifications.

Table 4.2: Confusion matrix of the classification results on the test set, with the rows being the ground truth reference type and the columns the predicted reference types. The zeros were omitted to improve clarity.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	4151															
2		25														
3			111													
4				759												
5					132											
6						72										
7							219									
8						1		163								
9									77							
10										191						
11											81					
12												159				
13													45			
14														24		
15															39	
16																46

4.3.2 Results

In the experiments, we set the instance registration parameters as: desired surface error for the optimization $\alpha_1 = 1e-3$, minimum acceptance error $\alpha_2 = 5e-3$, outer and inner loop maximum iterations $IT_o = 40$ and $IT_i = 200$ respectively, initial point cloud density $N_p = 2048$, incremented by 2048 for a maximum of times $\mathbf{i} = 4$, and the Adam optimizer learning rate to $1e-3$. It is worth mentioning that both α_1 and α_2 represents a percentage error on the geometry metrics. These parameters were fine-tuned to obtain a good visual quality on the optimized model. After training the PointNet++ and setting the optimization constants, we optimize the 3D CAD model previously described using the proposed supervised framework, obtaining $MeanE_s = 0.0003$ and $StdE_s = 0.0009$.

We highlight that all instances with $E_s > 5e-3$ were rejected, and the original triangle mesh was maintained in the model. Using this last validation step, we avoided any misclassifications and deformations that could derive from a sub-optimal instancing matrix. The optimized model obtained using the proposed framework, shown in Figure 4.8, requires only 2.61% of the original model size. It is also worth mentioning that, with further inspection, no visual difference is perceived between the original model, shown in Figure 4.5 and the optimized one.

To compare how the proposed framework relates with previous methods, first, we optimized the same CAD model using the shape matching approach

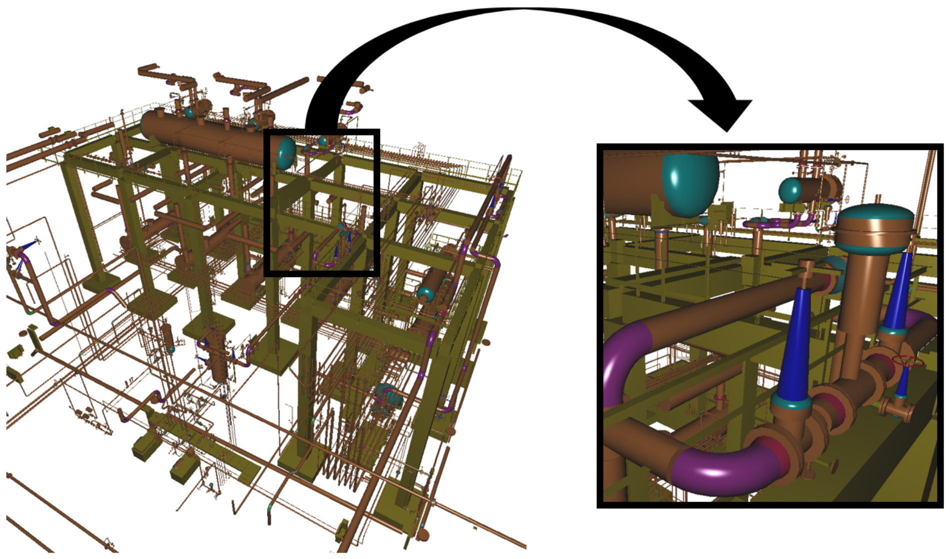


Figure 4.8: Optimized CAD model obtained using the proposed supervised framework. The optimized model has only 2.61% of the original model size, and has a $MeanE_s = 0.0003$, with a $StdE_s = 0.0009$. The model is color-coded according to each mesh reference type.

developed by Santos & Celes Filho, 2014, obtaining a memory reduction of 97.40%, a very similar memory reduction that was obtained using our proposed framework (97.39%). By further inspecting the optimized CAD model, we observed that all triangle mesh instances found by both techniques had equal vertices incidence order and surface triangulation. This first comparison demonstrates that on Santos & Celes Filho, 2014, best-case scenario, our framework has an equivalent performance.

Aiming to compare our framework with Santos & Celes Filho, 2014, worst-case scenario, where the vertex and triangulation correspondence are not found on triangle meshes, we randomly shuffled the vertices and triangulation of the meshes. Considering the mesh representation as a vector of vertices and a vector of elements, we shuffled the vector of vertices and updated the vector of elements to the new corresponding vertices indexes. Note that this is a possible scenario, since the CAD modeling is human and software dependent. After optimizing this shuffled CAD model, we obtained the same results using our proposed framework, 97.39% of memory reduction with $MeanE_s = 0.0003$ and $StdE_s = 0.0009$, while using Santos & Celes Filho, 2014, approach, since it heavily depends on the meshes topology, we obtained only a 0.12% of memory reduction. This second experiment shows that our proposed framework is capable of optimizing a greater diversity of repeated meshes in CAD models.

4.3.2.1 Instance Registration Results

Furthermore, to compare how our instance registration procedure relates to previous methods, we conducted three other experiments. The results are summarized in Table 4.3. In the first experiment, we replaced our instance registration with the SICP (Du et al., 2010) registration, running it for 1000 iterations with point clouds containing 8192 points, which is the highest density used in previous experiments. In the second experiment, we replaced only the Adam optimizer with Chamfer distance by the SICP registration, executing the optimization procedure with the same parameters used before. In the third experiment, we replaced our instance registration only by the Adam optimizer with Chamfer distance, executing it with point clouds containing 8192 points until the Chamfer distance stopped decreasing or α_1 was reached. For better comparison with our instance registration, we only executed these previous approaches for meshes that have $E_s > \alpha_1$ after the normalization, and also used the PCA approach to initialize the instancing matrix.

Analyzing Table 4.3, we can observe that using $5e-3$ as the minimum acceptance error, the memory reduction using all the approaches was similar, but, using a more strict threshold, $1e-3$, our instance registration outperforms the other approaches and still maintains a significant memory reduction. This result is directly related to the number of instanced meshes with surface error $E_s \leq 0.001$, showing that our approach optimized more triangle meshes below the optimization goal $\alpha_1 = 1e-3$

We also observe that the average time spent to estimate the instance matrix was smaller when compared to the other approaches, especially the SCIP registration. Note that all registration approaches can optimize each instance in parallel, since they are optimized independently from each other. It is also worth mentioning that we considered the computational time spent to execute all the framework steps, since the time spent on the other steps (351 milliseconds) was significantly smaller than the time spent in the registration step.

When comparing the $MeanE_s$ and the $StdE_s$ in Table 4.3, we notice that our instance registration loops improved the registration quality in both SICP and Adam with Chamfer distance approaches. This result is directly related to the decrease in the number of meshes with $E_s > 0.001$. We also notice that, when combined with the SICP approach, our instance registration loops slightly increased the number of meshes with $E_s > 0.005$. This happens because executing the SICP on point clouds that have smaller densities on earlier iterations can lead to an undesired registration that may not be reversed by later iterations on more dense point clouds.

Moreover, by further inspecting the impacts our instance registration loops had in both SICP and Adam with Chamfer distance approaches, we observe that in the Adam the average time decreased, while it increased in the SICP. This occurs because the Adam optimizer is an adaptive approach to perform the gradient descent and take advantage of the adaptive sampling performed by our instance registration loops, while the SICP registration does constant steps, executing more iterations to achieve a desired surface error when combined with the adaptive sampling.

Table 4.3: Performance comparison of the supervised framework using different registration approaches: the SICP, the SICP with our instance registration optimization loops, the Adam optimizer with d_{CH}, and the proposed instance registration (IR). These results demonstrate that the proposed instance registration procedure, with less computational time, obtains meshes with greater quality, in a more consistent way, which translates into a better overall memory reduction. Note that meshes with $E_s > 0.001$ also have $E_s > 0.005$.

	SICP	IR loops with SICP	Adam with d_{CH}	Our IR
Mem. Red. $\alpha_2 \leq 0.005$	97.20%	97.08%	**97.39%**	**97.39%**
Mem. Red. $\alpha_2 \leq 0.001$	69.77%	88.14%	93.82%	**97.16%**
$MeanE_s$	0.0010	0.0007	**0.0003**	**0.0003**
$StdE_s$	0.0026	0.0023	0.0010	**0.0009**
Avg. Time per Mesh (sec)	62.64	166.77	13.65	**8.66**
Meshes $E_s \leq 0.001$	12625	16590	17968	**18697**
Meshes $E_s > 0.001$	6226	2261	883	**154**
Meshes $E_s > 0.005$	141	218	2	**2**

4.3.2.2 Rendering Performance

To measure the rendering performance of the optimized CAD model obtained using our supervised framework, we measured the frames per second (FPS) using three different approaches on a desktop PC with an Intel Core i7

3.20GHz with 6-core processor, 32GB of RAM and an NVIDIA GeForce GTX 1070 8GB graphics card. In the first approach, we used a single draw call to render all the meshes of CAD model, obtaining an FPS of approximately 2000. Although using this approach, we obtain a high rendering performance; it is limited by the graphics card memory, meaning that it is only feasible when the model fits entirely on the graphics card memory. In the second approach, we individually rendered the meshes of the CAD model, obtaining an FPS of approximately 10. This approach presents a different limitation: the CPU becoming a bottleneck with rendering API function call overhead.

For the last approach, we implemented the same instancing technique as Santos & Celes Filho, 2014, obtaining an FPS of approximately 1000. This technique perfectly fits our framework, since affine transformation matrices are computed, which enables the instantiation of several components that share the same triangle mesh with a single draw call. We highlight that this approach addresses the drawbacks observed in the previous ones, since only a small number of triangle meshes are stored on the graphics card memory, and also only a few API function calls are executed using the hardware-accelerated instance rendering API (Pharr & Fernando, 2005), avoiding the function call overhead.

4.3.2.3
Results Using Another CAD model

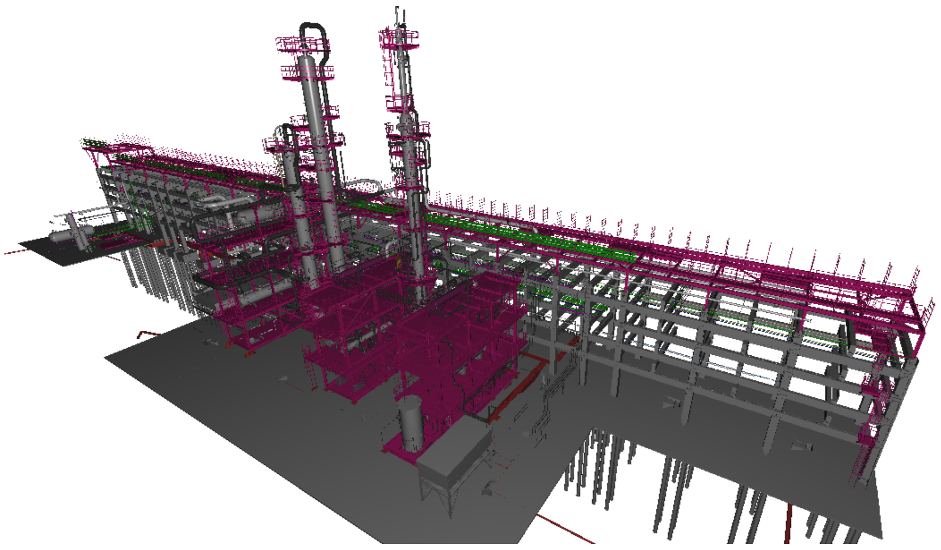


Figure 4.9: 3D CAD model with 74549 unlabeled triangle meshes.

Finally, we performed one last experiment to evaluate how the supervised framework would perform on a different CAD model than the one used to train the PointNet++. To do so, we used a CAD model that contains 74549 triangle meshes without any labeling information, shown in Figure 4.9, and obtained a memory reduction of 64.40%, with $MeanE_s = 0.0117$. By further analyzing the obtained result, we observed that the previous known reference meshes could not represent with a desired accuracy some of the triangle meshes found in this CAD model. This lack of similarity between the reference meshes and the CAD model triangle meshes decreases the classification performance and, as a consequence, worse registrations are produced, which increase the surface error while decreasing the memory reduction.

This result shows that the supervised approach depends on a previous knowledge about the 3D CAD model, since the PointNet++ needs to be trained to correctly classify the triangle meshes. Without previous information about the model, the framework may not find a sufficiently similar reference mesh, restraining the instance registration to go beyond a certain error threshold. This lack of similarity is illustrated by Figure 4.10, in which a rectangular toroid triangle mesh was misclassified as a circular toroid, since there is no rectangular toroid among the reference meshes. We can also observe that the instance registration obtained a good instancing matrix, and even though the output looks similar, their surfaces do not match, and the surface error is high.

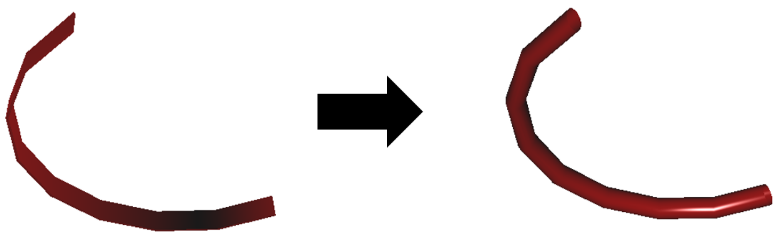


Figure 4.10: Example of a rectangular toroid misclassified as a circular toroid due to the lack of similarity between the input triangle mesh and the reference meshes. This instance is rejected by the framework, since the surface error is greater than $5e-3$.

4.3.3 Discussion

The proposed framework, as shown in Section 4.3.2 in the shuffled CAD model experiment, by relying on uniformly sampled point clouds to identify similar triangle meshes in the CAD model, and also to estimate an affine transformation matrix to instantiate them, indeed does not depend on the vertices incidence order and mesh topology. We highlight that the topology, and some details, of the original triangle mesh may not be preserved when a reference mesh, followed by an instancing matrix, is used. However, we are interested in preserving the visual appearance of the original mesh, since it is more relevant than the topology in the CAD visualization domain. Our proposed framework achieve this by employing dense point clouds, which avoid loss of details, and a last validation step, which guarantees an upper bound on the surface errors, ensuring that the final instancing is visually similar to the original.

The framework takes advantage of the PointNet++ robustness to achieve high classification performance. This is essential to prevent the execution of the instance registration with triangle meshes that are not similar, minimizing the computational time spent in the CAD model optimization. In Table 4.3, we can observe that the average time spent on each triangle mesh is reasonable; however, performing a pairwise registration, and choosing the one with the best surface error to represent the original mesh, can become unfeasible for CAD models composed by thousands of triangle meshes. Although misclassifications may be obtained by the PointNet++, they were identified, and the original triangle mesh was maintained in the optimized model.

We also highlight that, even though most of the experiments were performed with a single CAD model, the optimization performed by the framework processes each triangle mesh individually, and, since the CAD model is composed by 18851 triangle meshes labeled in 16 reference types, the obtained results provide strong evidence that the proposed approach is capable of generalizing to diverse triangle mesh with different transformations.

Last but not least, initializing the instancing matrix using the PCA orthonormal basis combination, described in Equation 4-4, greatly improved the final registration results. To support this, we optimized the dataset CAD model using three different matrix initialization strategies, the identity matrix, a random matrix, and the PCA_{INIT} matrix, summarizing the obtained results in Table 4.4.

Analyzing Table 4.4, we observe that the final registration results were significantly improved by the PCA initialization strategy. The decrease in the mean surface error and standard deviation is an evidence of that fact, since it can be interpreted as better detail preservation. Moreover, the average time spent on each triangle mesh also decreased when compared to the other initialization strategies. This derives from the fact that the PCA strategy tends to obtain a better starting point to the instance registration procedure, being closer to the desired registration, which makes the optimization susceptible to local minima. We highlight that, even though the PCA orthonormal basis is not unique for one same reference mesh, and may be off by 180-degree rotation, this was not a major issue, and the registration procedure was able to converge. We can observe this in Figure 4.8, where even in triangle meshes that does not have a lot of symmetry, such as the purple circular toroids, we obtained visually accurate registration results.

Table 4.4: Optimization results obtained initializing the instancing matrix using the identity matrix, a random matrix, and the PCA_{INIT} matrix. These results demonstrate that the PCA initialization strategy greatly improved the final mesh quality, decreasing both $MeanE_s$ and $StdE_s$, while also decreasing the average time spent on each triangle mesh.

Initialization Strategy	$MeanE_s$	$StdE_s$	Average Time per Mesh (sec)
Identity	0.0098	0.0266	16.50
Random	0.0048	0.0186	11.21
PCA_{INIT}	**0.0003**	**0.0009**	**8.66**

We highlight that, relying only on the PCA alignment strategy is not enough to obtain a final accurate representation. The PCA eigenvectors may not produce a perfect alignment when considering the reference mesh and the normalized input mesh, since small non-uniform scaling along the geometries axes may result in eigenvectors that have small deviations. This small deviation can lead to a significant deformation when the scale normalization is reversed for the final instantiation. As a comparison, we optimized the CAD model described in Section 4.3, using only the PCA initialization described in Equation 4.3.2.1 as the estimated registration, obtaining 6656 meshes with $E_s > 0.005$, and a memory reduction of 68.02% (using $\alpha_2 \leq 0.005$), a significantly worse result when compared to the results presented in Section 4.3.2.1. This results shows that it is important to perform a registration procedure after the PCA initialization strategy.

5
Self-Supervised Shape Instance Matching Framework

In this Chapter, we detail the proposed self-supervised framework for the shape matching problem on 3D CAD models, present some results and discuss them. Our objective is to overcome the main drawback of the supervised framework: the high dependency on previous knowledge about the 3D CAD model, which we observe in Section 4.3.2.3, while preserving the previous advantages of the framework. To address this issue, we modified the framework described in Chapter 4, replacing the supervised classification step with an self-supervised clustering approach, obtaining the self-supervised framework visualized in Figure 5.1. First, the framework preprocesses all the meshes of a CAD model, obtaining point clouds for each one of them. Next, the point clouds are grouped using the self-supervised clustering step, and a reference mesh is selected for each cluster. Finally, an affine transformation is estimated between each triangle mesh of the CAD model and the cluster reference mesh.

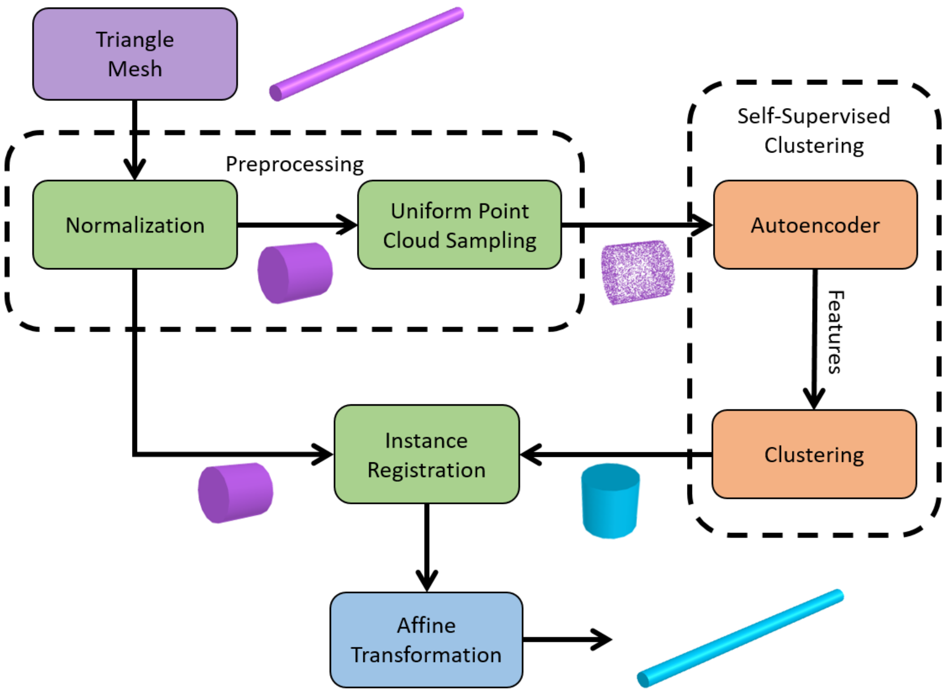


Figure 5.1: Overview of the proposed self-supervised framework, in which the classification was replaced with an self-supervised clustering step.

The new self-supervised clustering has two main steps. First, we obtain feature vectors for the 3D CAD model meshes, using the PointNet++ (Qi et al., 2017b) encoder, trained using the autoencoder illustrated in Figure 5.2. Second, we cluster the meshes using the obtained feature vectors. The autoencoder, first, encodes the uniformly sampled point clouds of the preprocessing step, using the PointNet++ encoder, into a 1024-dimensional feature vector, which is used by a decoder composed by two fully connected layers, with the first one having batch normalization (Ioffe & Szegedy, 2015) and ReLU activation, to reconstruct the input point cloud. We highlight that we used the 1024-dimensional feature vector already implemeted by PointNet++ encoder architecture, and feature vectors with different dimensions could also be used.

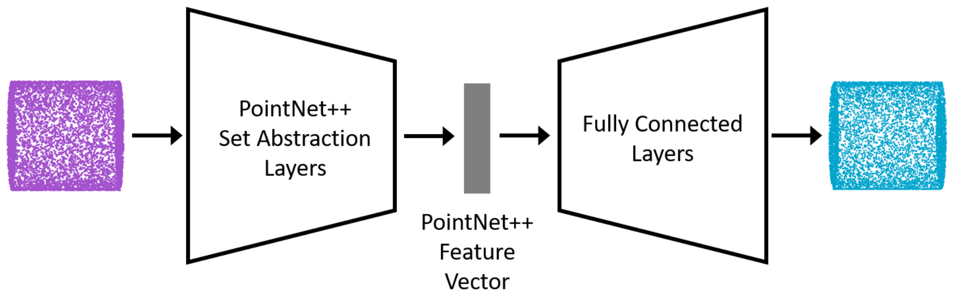


Figure 5.2: Visualization of the autoencoder based on the PointNet++.

To extract feature vectors for the 3D CAD model meshes, we trained the described autoencoder using a variety of shapes found on 3D CAD models, using the Adam optimizer (Kingma & Ba, 2014) and the averaged Chamfer distance as cost function:

$$d_{aCH}(X,Y) = \frac{\sum_{x\in X} \min_{y\in Y} ||x-y||_2^2}{N_X} + \frac{\sum_{y\in Y} \min_{x\in X} ||x-y||_2^2}{N_Y} \qquad (5\text{-}1)$$

where X, Y, N_X, N_Y are, respectively, the input and reconstructed point clouds, and its respective number of points. Once the training was finished, the encoder is able to meaningfully represent the input point cloud with a 1024-dimensional feature vector, which the decoder uses to reconstruct the input point cloud.

This feature vector is then extracted for the 3D CAD model meshes and used to cluster them. To cluster the triangle meshes, we experimented with two different approaches: *K-Means*|| (Bahmani et al., 2012) and HDBSCAN (Campello et al., 2013), described in Section 3.3. We chose the *K-Means*|| algorithm due to its simplicity and popularity, and the HDBSCAN because, as shown by Campello et al., 2013, it was able to outperform previous density-based clustering solutions.

In the *K-Means*||, although some approaches were developed to estimate the desired number of cluster K, as mentioned in Section 3.3.1, in our domain, we can estimate K as a target percentage of the number of reference meshes that we want to maintain on the 3D CAD model. On the other hand, for the HDBSCAN, we just need to set m_{pts}, which is the number of points in the neighborhood of a point to consider it a core point (including itself). We highlight that, using the HDBSCAN, some triangle meshes may be considered noise, meaning that they are not assigned to any cluster. In our case, they were considered as unique and maintained in 3D CAD model. In section 5.1.2, we further discuss on the results obtained with each clustering technique.

After clustering the 3D CAD model meshes, one reference mesh is selected for each cluster. When the *K-Means*|| is used, we select as reference mesh the mesh that has its feature vector closest to the cluster centroid, while with the HDBSCAN, we select the one that has greater strength membership. Once the reference mesh for each cluster is selected, the instance registration described in Section 4.2 is executed for each mesh on the 3D CAD model. We highlight that, for implementation purposes, we avoided the squared root computations in the Chamfer (pseudo) distance d_{CH}, using it as:

$$d_{CH} = \sum_{x \in X} \min_{y \in Y} ||x - y||_2^2 + \sum_{y \in Y} \min_{x \in X} ||x - y||_2^2 \tag{5-2}$$

During early experimentation, we observed that the surface error, described in Equation 4-2, may accept meshes that are not completely similar, due to asymmetry: it measures only the mean distance from the target shape point cloud to the reference mesh transformed by the registration matrix. This asymmetric measurement ensures that the target shape is covered by the reference one, but not the other way around. This way, some details can be lost, as shown in Figure 5.3, where a doorway is missing.

To address this issue, we modified the instance registration procedure, replacing the asymmetric surface error E_s with a symmetric one, shown in Equation 5-3. In this Equation, $SymE_s$ is the symmetric error, $\mathcal{T}$ is the target mesh that we are trying to instantiate, $\mathcal{S} * M$ is the reference mesh transformed by the registration matrix, and $P_{\mathcal{T}}$ and $P_{\mathcal{S}M}$ are point clouds sampled, respectively, on the target mesh and reference mesh transformed by the registration matrix.

$$SymE_s(\mathcal{T}, \mathcal{S} * M) = max(E_s(P_{\mathcal{T}}, \mathcal{S} * M), E_s(P_{\mathcal{S}M}, \mathcal{T})) \tag{5-3}$$

Finally, we also modified the last validation step to make it more rigid, ensuring that the final instancing matrix accurately represents the original mesh. To achieve this, in addition to verifying only if the symmetric surface error is below an acceptance threshold α_{Sym}, we also implemented two other metrics: the maximum difference and the area surface error. In the maximum

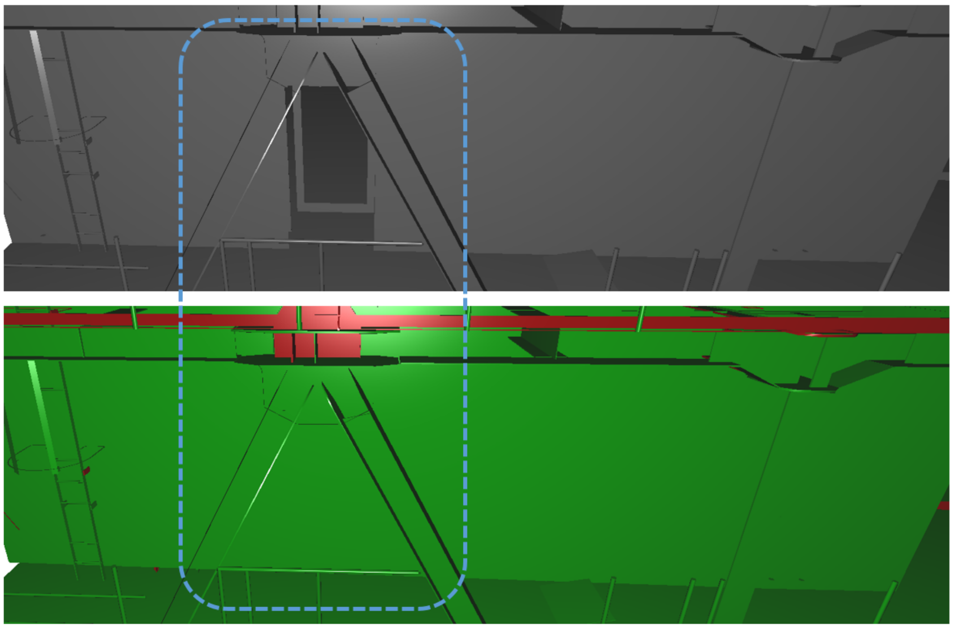


Figure 5.3: Example of a missing doorway, with the original model in gray and the optimized in green. The plane with the doorway is the target, and is covered by the plane that does not have the doorway, but not the other way around.

difference, we verify if the maximum distance from $P_{\mathcal{T}}$ and $P_{\mathcal{S}M}$ to, respectively, $\mathcal{S} * M$ and $\mathcal{T}$ is below a maximum threshold α_{md}. In the area surface error, we verify if the difference between the final surface area of the original mesh and the instantiated one is below an area threshold α_A.

Using these three metrics combined, we guarantee that the instance is similar to the original mesh, with the original mesh details and the intrinsic characteristics, such as surface area, preserved. Once the complete framework is executed for each mesh on the 3D CAD model, the meshes that had their registration accepted are inserted into the model as instances, while the rejected ones are maintained as unique meshes.

5.1 Experiments

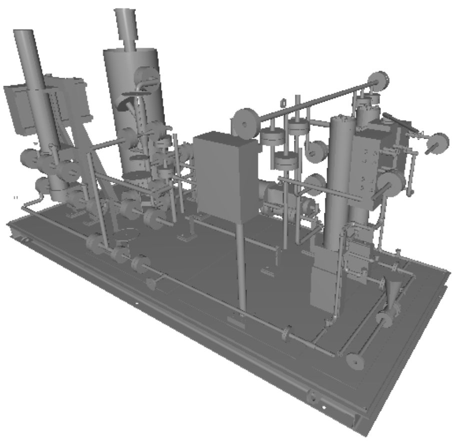
To evaluate the proposed self-supervised framework, we used two different real-world datasets. The first one is the same labeled CAD model described in Section 4.3; however, since our approach is self-supervised, the labeling information was not considered for any training loss computation during the training stage. The second dataset is a collection of 5 different 3D CAD models, visualized in Figure 5.4, containing a total of 61561 geometries without any label information. This dataset has a wider variety of triangle meshes, which would make the labeling task complex and time-consuming, posing a bigger challenge for the supervised approach. In Figure 5.5, we can observe some of the triangle meshes found in the dataset. To facilitate reading, we will refer to the first dataset as labeled dataset, and the second one as unlabeled dataset.

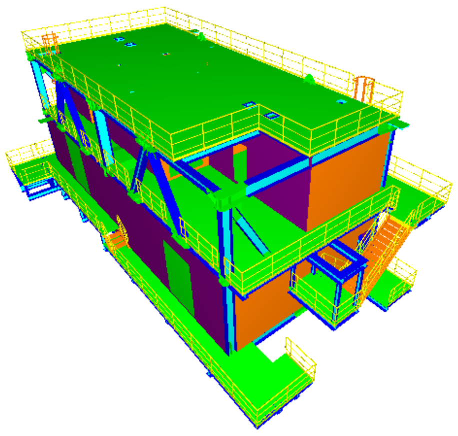
To measure the point cloud reconstruction quality of the implemented autoencoder, which also indicates the encoder capability of computing feature vectors that meaningfully represent the data, inspired by Bachmann et al., 2021, we used the averaged Chamfer distance, described in Equation 5-1. Similar to presented in Chapter 4, we used the mean symmetric surface error and standard deviation to measure the quality of the instanced meshes. Finally, to measure the clustering performance, we used the memory reduction of the self-supervised framework itself, since the clusters quality is directly related to the final optimization quality. We highlight that, even though the clustering algorithm may obtain clusters that have similar meshes but present small differences, *e.g.* a plane and another plane with a small hole, the last validation step is designed to identify these cases, rejecting the instance and maintaining the original mesh in the final 3D CAD model.

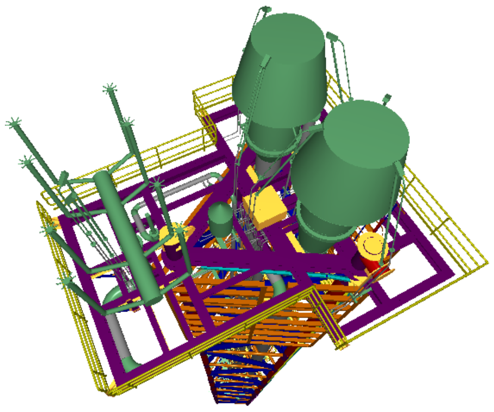
5.1.1 Training the Autoencoder

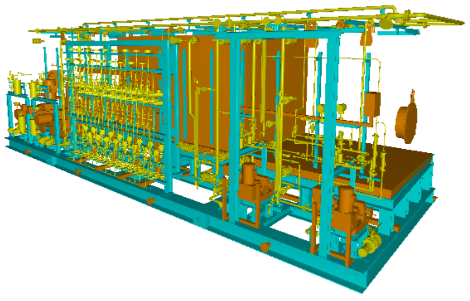
To execute the self-supervised framework on the datasets, first, we trained the autoencoder separately for each one. For the labeled dataset, we used the same train, validation, and test sets obtained in Section 4.3.1; however, in self-supervised tasks, the dataset is usually split into train and test sets, so we merged the train and validation sets. We highlight that, although the dataset is unbalanced, we did not perform any type of data augmentation, since, in an unlabeled dataset we would not correctly balance the data. Next, we trained the autoencoder for 100 epochs, using the Adam optimizer with a learning rate of $1e-3$. Then, we selected the epoch with the best mean reconstruction quality in the merged training set, which was $1e-4$, translating into a mean reconstruction quality also of $1e-4$ in the test set. These results indicate that the encoder was able to obtain feature vectors that meaningfully represent the triangle meshes for the labeled dataset.

For the unlabeled dataset, first, we randomly split each 3D CAD model in the dataset into two halves, and considered just the first half during the training and evaluation of the autoencoder. We chose to split the models first, using a 50%/50% ratio to ensure that the triangle meshes used to train the autoencoder do not impact in the clustering and registration steps. Then, we randomly split each CAD model first half into train and test sets using a


(a) Model 1


(b) Model 2


(c) Model 3


(d) Model 4

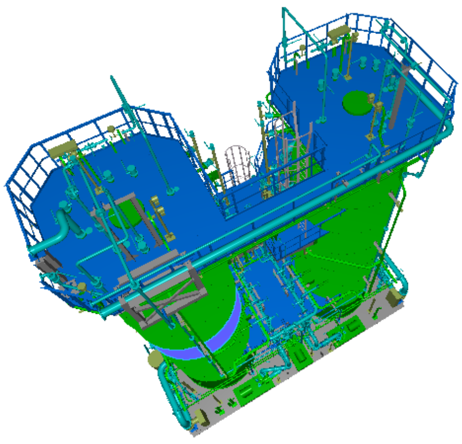

(e) Model 5

Figure 5.4: Visualization of the 3D CAD models found in the unlabeled dataset.

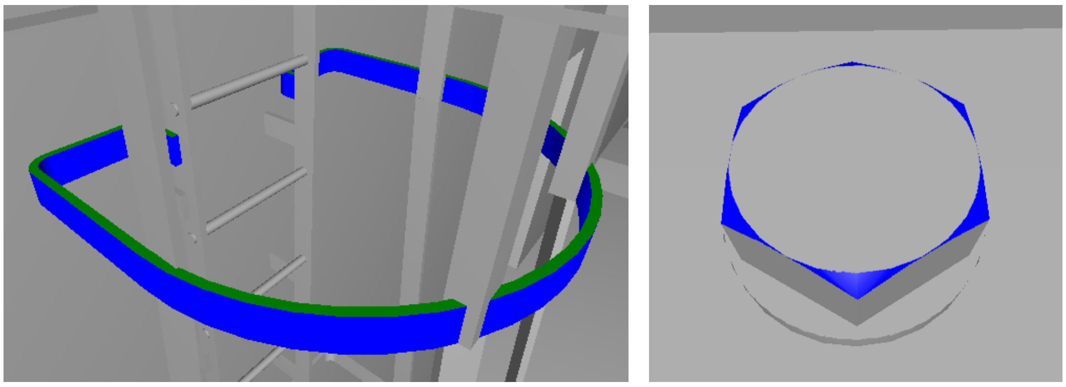


Figure 5.5: Example of complex triangle meshes (not parametric surfaces) found in the unlabeled dataset. The triangle meshes are highlighted in blue and green.

90%/10% ratio. After all the models were split, we grouped their train and test sets, so that each model is represented in the sets used to train and evaluate the autoencoder. Once the dataset was split, we trained the autoconder using the same parameters of the first dataset, and also selected the epoch with the best mean reconstruction quality for the validation set, $1e-5$, which translated into a mean reconstruction quality in the test set also of $1e-5$. These results also indicate that the encoder was able to obtain feature vectors that meaningfully represent the triangle meshes for the unlabeled dataset.

5.1.2 Results

In the experiments, we used the same optimization parameters described in Section 4.3.2. We fine-tuned the last validation metrics, obtaining $\alpha_{Sym} = 0.05$, $\alpha_{md} = 0.07$ and $\alpha_A = 0.01$. It is worth mentioning that both α_{Sym} and α_{md} represents a percentage error on the geometry metrics, and α_A is in squared meters. Using these values, we were able to obtain visually accurate results after executing the framework in both datasets, avoiding the asymmetric errors, as shown in Figure 5.6.

To select the clustering parameter K for the *K-Means*||, we empirically used 5% of the total number of meshes in the CAD model as the target of the number of meshes that should maintained on the model. Using this parameter, we set an upper bound of 95% on the redundancy removal, since we have at least 5% of the model's meshes on the final optimized model. For the HDBSCAN clustering, since we want to minimize the number of meshes considered as outliers, which are directly considered as unique meshes on the model, we set $m_{pts} = 2$. This way, a pair of meshes are already considered a cluster, and the redundancy removal may occur.

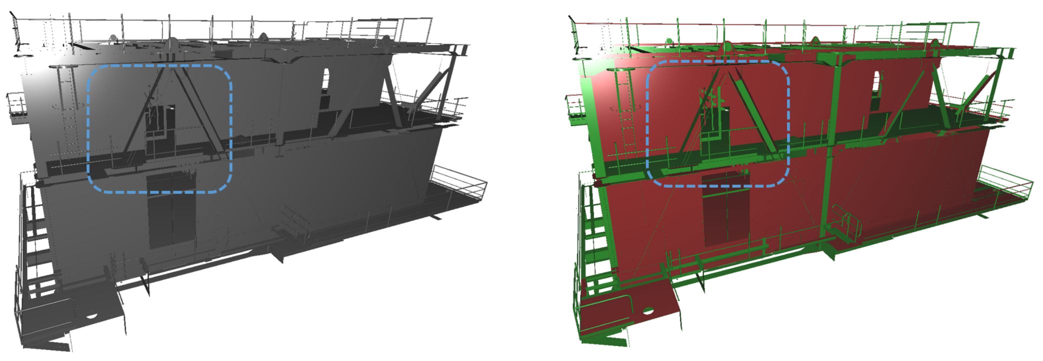

(a) Original (b) Optimized

Figure 5.6: Second half of the unlabeled dataset's Model 2. In (b), the accepted meshes are green, and the rejected ones are red. We can observe that, using the new validation metrics, the asymmetric errors were avoided, and the doorway was maintained in the final optimized model.

5.1.2.1 Results in the labeled dataset

Using the metric thresholds and clustering parameters set, we executed the self-supervised framework using the antoencoder trained for the labeled dataset to optimize its test set. When using the *K-Means*|| clustering, we were able to obtain a memory reduction of 83.93% with a mean $SymE_s = 0.0077$ and 0.0031 of standard deviation on the accepted instances, while using the HDBSCAN we obtained a 76.90% with a mean $SymE_s = 0.0075$ and 0.0033 of standard deviation on the accepted instances. These results indicate that the accepted instances did not impact the model quality, as shown in Figure 5.7, while also removing a significant amount of redundant information on the model.

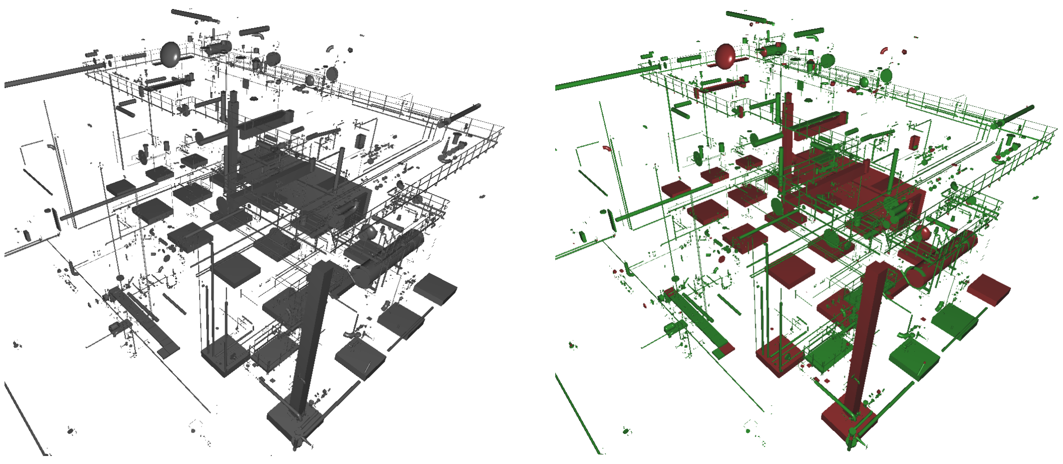

(a) Original (b) Optimized

Figure 5.7: Test set of the labeled dataset. In (b), the accepted meshes are green, and the rejected ones are red.

For comparison, we optimized the test set of the labeled dataset using the supervised framework, with the modified instance registration and last validation step. Using this strategy, we obtained a memory reduction of 97.03%, with a mean $SymE_s = 0.0003$ and 0.0005 of standard deviation on the accepted instances. These results show how powerful a supervised solution can be, obtaining a better redundancy removal, and, as the reference mesh selection is also better, the instance quality also improves. We also optimized this test set using the Santos & Celes Filho, 2014, solution, which also does not require any previous knowledge of the model, obtaining a memory reduction of 97.14%, which is very similar to the supervised solution. These results show that, when the labeling information is available, and also the model is composed mostly of meshes similar to parametric geometries, the supervised and Santos & Celes Filho, 2014, approaches were able to outperform our self-supervised framework.

5.1.2.2 Results in the unlabeled dataset

After experimenting on the labeled dataset, using the autoencoder trained for all the models on the unlabeled dataset, we executed the self-supervised framework, with both *K-Means*|| and HDBSCAN clustering, to optimize the halves of the 5 models that were not used during the training phase. With the *K-Means*||, we obtained a mean $SymE_s = 0.0039$ with mean standard deviation of 0.0040, and, with the HDBSCAN, a mean $SymE_s = 0.0041$ with mean standard deviation of 0.0041 on the accepted instances. These results indicate that, using both clustering techniques, we were able to maintain the overall model visual quality, observed in Figures 5.8 and 5.9.

We further compare our results on memory reduction with the unsupervised solution of Santos & Celes Filho, 2014, which are summarized in Table 5.1. We can observe that, in the models in which our solution obtained better memory reduction than Santos & Celes Filho, 2014, we were able to obtain significant improvement over the previous unsupervised solution, and, when we were outperformed, we still were able to obtain a significant amount of memory reduction. By further inspecting the CAD models which Santos & Celes Filho, 2014, obtained a better memory reduction, we observed that the CAD model is composed mostly of meshes similar to parametric geometries. This result demonstrates that, when the CAD model is composed of more complex geometries, as shown in Figure 5.10, we were able to outperform the previous unsupervised approach on average.

Furthermore, we highlight that our approach with the *K-Means*|| clustering was able to obtain better average results than with the HDBSCAN.

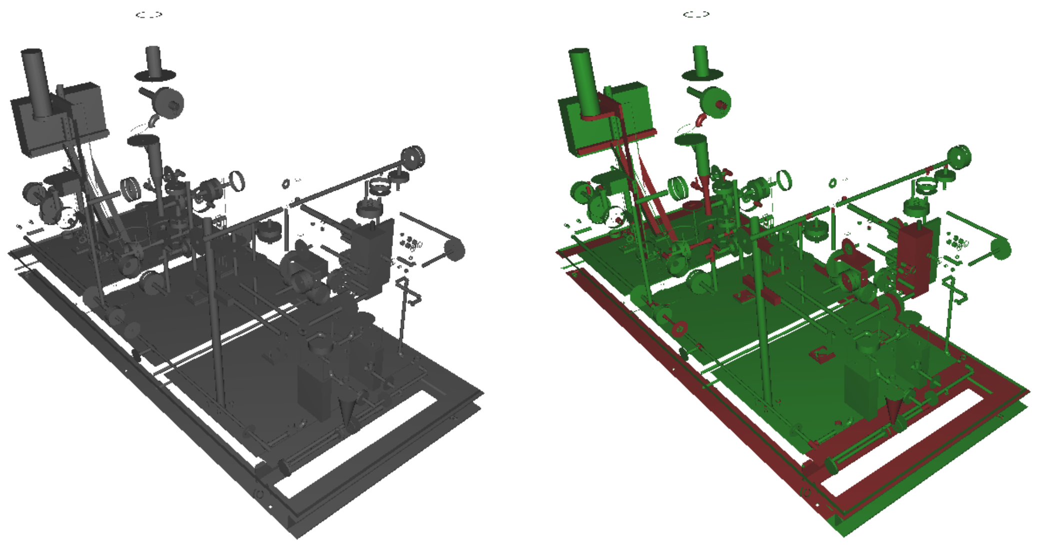
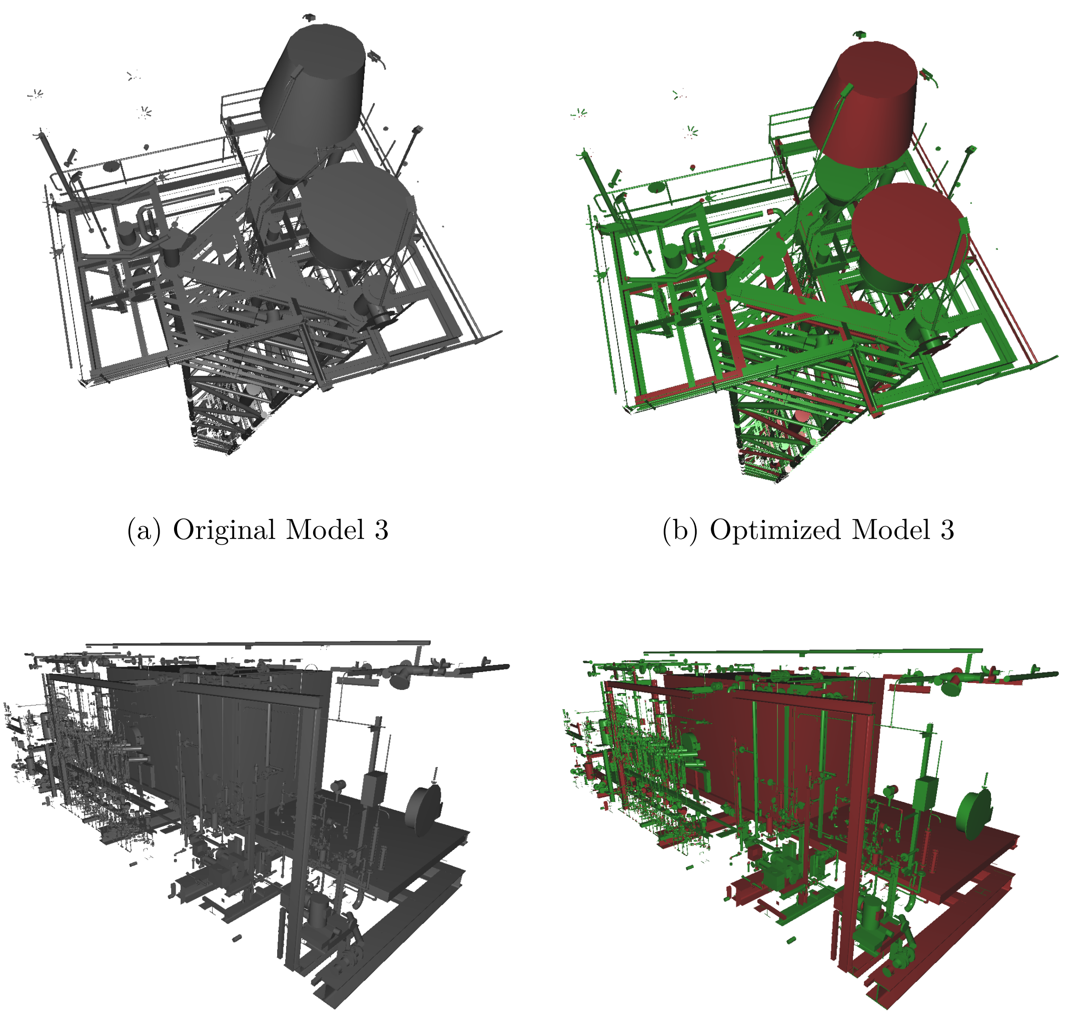

(a) Original (b) Optimized

Figure 5.8: Second half of the unlabeled dataset's Model 1. In (b), the accepted meshes are green, and the rejected ones are red.

Table 5.1: Comparison of the memory reduction between the self-supervised framework with HDBSCAN and *K-Means*|| and the Santos & Celes Filho, 2014, approach.

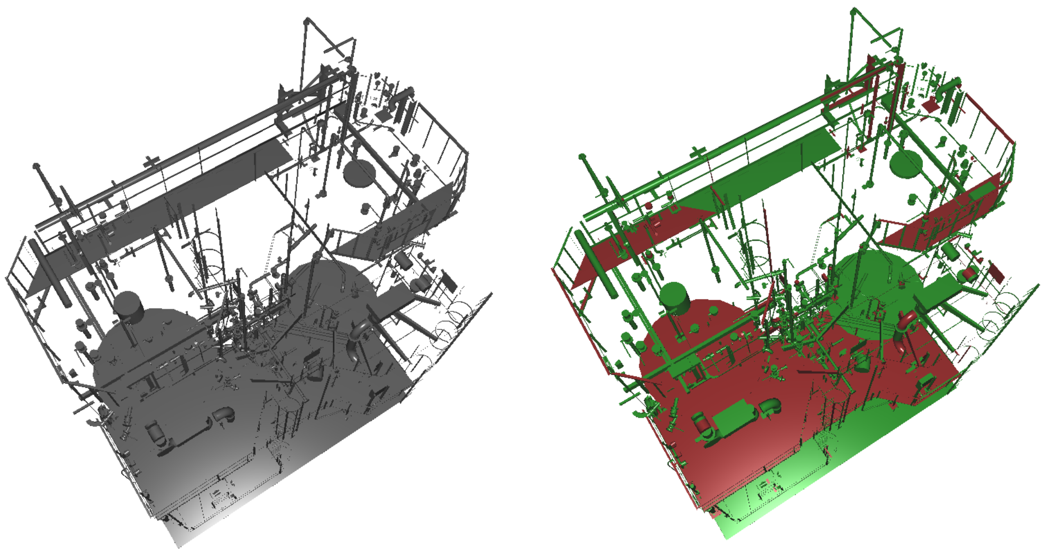
	Ours with HDBSCAN	Ours with *K-Means*||	Santos & Celes Filho, 2014
Model 1	76.08%	**76.58**%	51.89%
Model 2	73.54%	**78.32**%	65.66%
Model 3	**80.75**%	80.71%	72.29%
Model 4	67.94%	70.31%	**82.15**%
Model 5	70.81%	82.25%	**88.09**%
Average	73.82%	**77.63**%	72.01%

This result is consistent with the observation made by Caron et al., 2018, that "*some amount of oversegmentation is beneficial*". In fact, as shown in Table 5.2, we observe that, except on Model 1, more clusters were obtained using the approach with *K-Means*||, and, once the similar meshes may be divided into more clusters, the reference mesh selection may be better to instantiate the other meshes in the cluster, due to the rotational ambiguities found on the PCA alignment (Crespo & Aguiar, 2011).



(a) Original Model 3 (b) Optimized Model 3

(c) Original Model 4 (d) Optimized Model 4



(e) Original Model 5 (f) Optimized Model 5

Figure 5.9: Second half of the unlabeled dataset's Model 3, 4, and 5. In (b), (d), and (f), the accepted meshes are green, and the rejected ones are red.

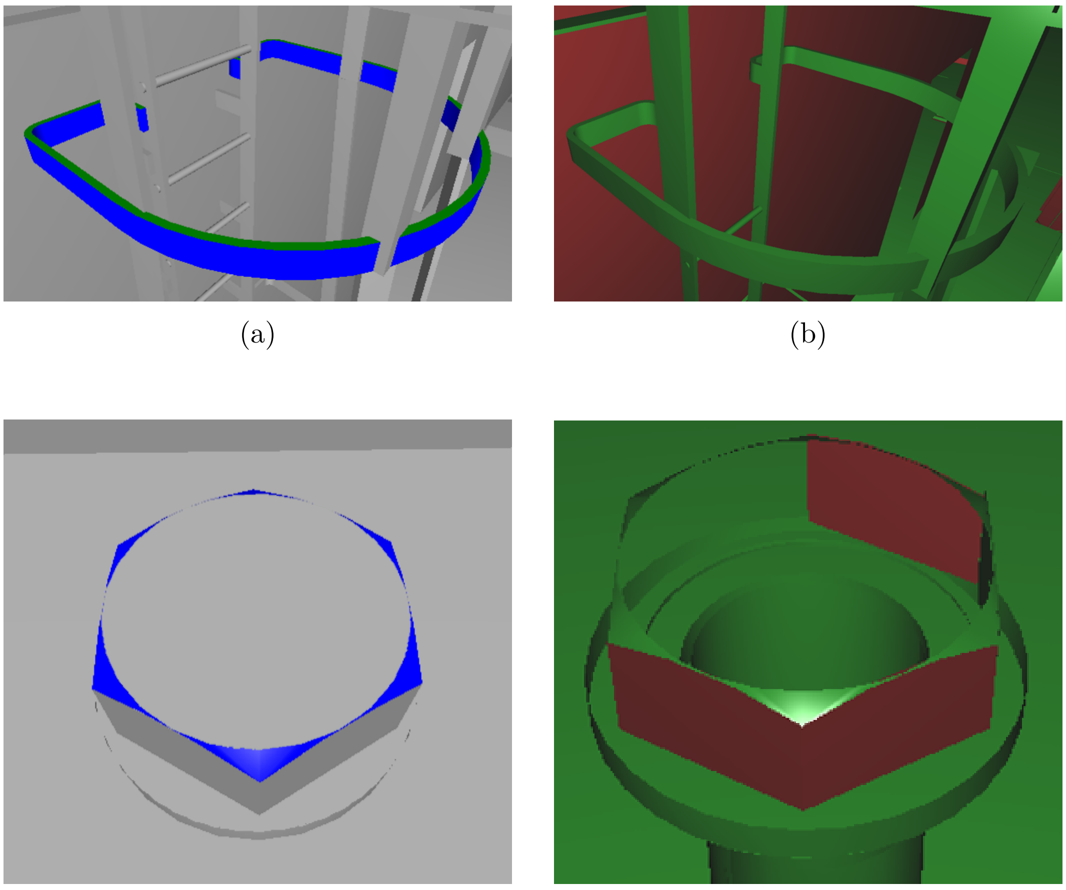

(a) (b) (c) (d)

Figure 5.10: Example of some complex geometries that we were able to optimize with the self-supervised framework. In (a) and (c), the input triangle meshes are highlighted in green and blue. In (b) and (d), we observe that the corresponding instance mesh was optimized and accepted (green) by the last validation step.

Table 5.2: Comparison of the number of clusters obtained on each model between the HDBSCAN and *K-Means*|| clustering algorithms.

	Meshes	*K-Means*||	HDBSCAN
Model 1	1581	79	97
Model 2	2598	129	55
Model 3	10332	516	204
Model 4	11323	566	369
Model 5	4947	247	182

5.1.2.3 Rendering performance

Finally, we compare the rendering performance between the models optimized by the self-supervised framework with both *K-Means*|| and HDBSCAN.

Table 5.3: Comparison of rendering performance between the optimized models by the self-supervised framework with HDBSCAN and *K-Means*|| clustering algorithms. The total number of meshes in the optimized model is the combination of the reference meshes and the unique meshes found by our self-supervised framework.

	K-Means||		HDBSCAN	
	Total Meshes	FPS	Total Meshes	FPS
Model 1	386	57	442	49
Model 2	362	60	468	44
Model 3	1805	12	1883	11
Model 4	3435	06	3404	06
Model 5	779	27	1083	20

To do so, we measured the frames per second (FPS) on each CAD model of the unlabeled dataset using a desktop PC with an Intel Core i7 3.20GHz with 6-core processor, 32GB of RAM and a NVIDIA GeForce GTX 1070 8GB graphics card, with the instancing technique described by Santos & Celes Filho, 2014.

Analyzing Table 5.3, we can observe that, in almost all models optimized using the *K-Means*|| clustering, a smaller amount of meshes were required to represent the original model than the HDBSCAN. This result directly translates into a slightly better FPS performance on the *K-Means*|| models than the HDBSCAN ones, since one draw call is executed for each one of the reference and unique meshes.

5.1.3 Discussion

Using the proposed self-supervised framework, as demonstrated in our experiments, we indeed overcame the major drawback of the supervised solution. By relying on an self-supervised clustering approach, no previous knowledge about the model is required. To perform the self-supervised clustering, the framework takes advantage of the PointNet++ encoder robustness to obtain meaningful feature vectors for the triangle meshes in the CAD model.

The framework also takes advantage of previous well-known clustering algorithms, *K-Means*|| and HDBSCAN, to group the triangle meshes using the feature vectors extracted with the PointNet++ encoder. This is essential to obtain clusters that contain similar meshes, while maintaining different meshes in separated clusters. Moreover, when comparing the results obtained, we noticed a better model optimization with the *K-Means*|| clustering than the HDBSCAN. Indeed, analyzing Tables 5.2 and 5.3, we observe that, even though the *K-Means*|| resulted in more clusters, the total amount of meshes on the final model was smaller than the HDBSCAN. These results indicate that the clusters obtained by the *K-Means*|| had a better quality than the ones obtained by the HDBSCAN.

Furthermore, we improved the instance registration step, using a symmetric error to obtain the instancing matrix and using more metrics to measure the quality of the instanced mesh. Improving the registration, as shown in Section 5.1.2, we avoided asymmetric errors that could derive from the previous strategy used in the supervised framework. We highlight that, when we optimized the labeled dataset, using the previous supervised solution with the improved validation step, we obtained very similar memory reduction and surface quality to the ones detailed in Section 4.3.2. These results confirm that the supervised solution, even though relying on an asymmetric quality measure, was able to obtain good optimization results.

When comparing the results obtained on the labeled dataset between the supervised and self-supervised frameworks, we observe that the supervised removed more redundant information than the self-supervised one, while also obtaining better quality on the instanced meshes. This result shows that the supervised approach correctly identified more triangle meshes as instances from one another, and, as the reference mesh selection is also better, the quality of the instanced triangle meshes improves.

Finally, when comparing the results obtained on the unlabeled dataset between our self-supervised framework and a previous unsupervised approach, we observe that our framework removed more redundant information on the models on average, outperforming the previous approach. This result demonstrates the effectiveness of the proposed framework, especially when the CAD models are composed of meshes with more complexity (different from parametric geometries).

6 Conclusions

The efficient representation of a CAD model is an important field of research, especially due to the recent popularization of methodologies that relies on CAD models to improve the engineering process. Previous research has proposed a CAD model optimization that minimizes the representation redundancy commonly found in 3D CAD models; however, it has major drawbacks: high dependency on the triangle mesh topology and vertices incidence order.

With the recent advances in the deep learning field of research, several geometric tasks, such as shape and point cloud registration, significantly improved. Previous deep learning approaches could be adapted to minimize the CAD models' redundant information; however, they also have some drawbacks: being limited to primitive geometries, obtaining transformations not well-suited for the 3D CAD visualization domain, and not considering the non-uniform scale commonly found on the CAD domain.

In this work, we proposed a supervised and a self-supervised framework that minimizes the CAD models' redundant information, finding instances of repeated triangle meshes and computing an instancing matrix between them. Both frameworks, by relying on point clouds uniformly sampled on generic meshes' surfaces, overcome most of the previous drawbacks, being independent of the mesh topology and vertex order while also generalizing for any kind of geometry.

Furthermore, both frameworks perform an instance registration step that relies on the PCA method and the Adam optimizer with Chamfer distance as a cost function to estimate an affine transformation matrix between two meshes. The estimated affine transformation considers the non-uniform scale between the triangle meshes and is well suited for the 3D CAD visualization domain. We highlight that the instance registration has full control over the final surface error, guaranteeing an upper bound on any surface error that the optimization procedure may introduce.

The proposed supervised framework takes advantage of the PointNet++ classification network to identify instances of repeated triangle meshes accurately and performs the instance registration to obtain an optimal affine transformation between them. Using the proposed supervised framework, we optimized a 3D CAD model to 2.61% of its original size while preserving the original triangle meshes visual quality.

When compared to previous method, the proposed supervised framework obtained similar results; however, for the previous method's worst-case scenario, the proposed supervised framework obtained significantly better results. We also demonstrated that the developed instance registration procedure achieves a lower surface error while also performing faster than previous approaches that also consider non-uniform scaling.

The proposed self-supervised framework leverages the robustness of the PointNet++ encoder to obtain feature vectors that meaningfully represent the CAD model triangle meshes. Then, using these feature vectors, clustering algorithms are executed to group the triangle meshes. After that, each group of triangle meshes is optimized using the instance registration step. Using this self-supervised clustering approach, the proposed self-supervised framework overcomes the main drawback found on the supervised framework: the high dependency on a previous knowledge about the 3D CAD model.

When compared to the supervised framework, the self-supervised obtained a smaller but still significant memory reduction. Moreover, when compared to previous unsupervised method, using an unlabeled dataset composed by more complex geometries, the self-supervised framework outperformed the previous unsupervised method by a maximum of 24.69%, and 5.62% on average. These results demonstrate the effectiveness of the proposed solution.

6.1 Future Work

Since recent CAD models contain several triangle meshes that cannot be represented by a primitive geometry, and labeling those meshes could be a time-consuming task, we point out some suggestions for future work:

- **Autoencoder improvement.** More recent autoencoder architectures, such as variational autoencoders and adversarial autoencoders, could be used to improve the feature vectors extracted from point clouds, as shown by Zamorski et al., 2020. Obtaining better feature vectors can directly impact the CAD model optimization since clusters with better quality could be achieved and, as a consequence, better registration results.
- **Point cloud distance function improvement.** Choosing a good metric to measure the discrepancy between point clouds is an important step to obtaining a good representation learning using neural networks (autoencoders included). Although the Chamfer distance is widely used in

point cloud analysis, since it is fast and differentiable (Fan et al., 2017), other less favored distances have proven to obtain better results. Fan et al., 2017 show that using the Earth Mover's distance, they obtained better point set generation results. However, the Earth Mover's distance has a high computational cost. Nguyen et al., 2021 show that using the sliced Wasserstein distance, they obtained better feature vectors for the point cloud representation while maintaining a computational cost close to the Chamfer distance. Using this distance could improve the learning capabilities of the autoencoder and also improve the Adam-based registration procedure, resulting in better CAD model optimization.

– **Deep Clustering.** Some research proposed the combination of the network-specific loss, in our case the Chamfer distance, with a clustering loss to obtain cluster-friendly features (Hassani & Haley, 2019; Song et al., 2013). Usually, these methods have an upper bound on the number of clusters that the trained network is able o obtain, which could be an issue, since the number of clusters varies from one CAD model to another. However, a CAD model partition strategy could be implemented to ensure a good clustering. Moreover, the intra-cluster registration errors could also be used as loss during the training phase. Using such strategy, the network will be able to obtain clusters that directly minimize the registration errors and maximize the redundancy removal.

– **Primitive optimization.** As discussed in Souza Moreira, 2015, the primitive representation is more compact than the mesh one, and, by adapting this generic-meshes solution to also be able to identify and obtain the primitive parameters to instantiate the CAD model meshes, we will be able to further improve the memory reduction. To achieve this, the cluster's reference mesh could be classified into previously known primitives, and if a good enough score is obtained for a particular primitive, the entire cluster could be considered as an instance of the primitive, and their attributes could be estimated.

– **Scanned point clouds.** 3D CAD models are commonly updated to match the modifications executed on the real industrial plant. During this process, scanned point clouds may be used as the modeling technique for new components. We encourage future research to apply the proposed framework in order to obtain better CAD meshes to update the model. In this case, the framework should be adapted to perform point cloud segmentation, once the scanned point clouds usually are not segmented into single geometries.

Bibliography

ACHLIOPTAS, PANOS; DIAMANTI, OLGA; MITLIAGKAS, IOANNIS; GUIBAS, LEONIDAS. **Learning representations and generative models for 3d point clouds**. International conference on machine learning. PMLR. 2018, pp. 40–49.

ALT, HELMUT; MEHLHORN, KURT; WAGENER, HUBERT; WELZL, EMO. **Congruence, similarity, and symmetries of geometric objects**. Discrete & Computational Geometry 3.3 (1988), pp. 237–256.

AOKI, YASUHIRO; GOFORTH, HUNTER; SRIVATSAN, RANGAPRASAD ARUN; LUCEY, SIMON. **Pointnetlk: Robust & efficient point cloud registration using pointnet**. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019, pp. 7163–7172.

ARTHUR, DAVID; VASSILVITSKII, SERGEI. **k-means++: The advantages of careful seeding**. Tech. rep. Stanford, 2006.

BACHMANN, JOËL; BLOMQVIST, KENNETH; FÖRSTER, JULIAN; SIEGWART, ROLAND. **Points2Vec: Unsupervised object-level feature learning from point clouds**. arXiv preprint arXiv:2102.04136 (2021).

BAHMANI, BAHMAN; MOSELEY, BENJAMIN; VATTANI, ANDREA; KUMAR, RAVI; VASSILVITSKII, SERGEI. **Scalable k-means++**. arXiv preprint arXiv:1203.6402 (2012).

BALL, GEOFFREY H; HALL, DAVID J. **A clustering technique for summarizing multivariate data**. Behavioral science 12.2 (1967), pp. 153–155.

BELLO, SAIFULLAHI AMINU; YU, SHANGSHU; WANG, CHENG; ADAM, JIBRIL MUHMMAD; LI, JONATHAN. **Deep learning on 3D point clouds**. Remote Sensing 12.11 (2020), p. 1729.

BEN-SHABAT, YIZHAK; LINDENBAUM, MICHAEL; FISCHER, ANATH. **3d point cloud classification and segmentation using 3d modified fisher vector representation for convolutional neural networks**. arXiv preprint arXiv:1711.08241 (2017).

BEN-SHABAT, YIZHAK; LINDENBAUM, MICHAEL; FISCHER, ANATH. **3dmfv: Three-dimensional point cloud classification in real-**

time using convolutional neural networks. IEEE Robotics and Automation Letters 3.4 (2018), pp. 3145–3152.

BENGIO, YOSHUA; COURVILLE, AARON; VINCENT, PASCAL. **Representation learning: A review and new perspectives**. IEEE transactions on pattern analysis and machine intelligence 35.8 (2013), pp. 1798–1828.

BESL, PAUL J; MCKAY, NEIL D. **Method for registration of 3-D shapes**. Sensor fusion IV: control paradigms and data structures. Vol. 1611. International Society for Optics and Photonics. 1992, pp. 586–606.

CAMPELLO, RICARDO JGB; MOULAVI, DAVOUD; SANDER, JÖRG. **Density-based clustering based on hierarchical density estimates**. Pacific-Asia conference on knowledge discovery and data mining. Springer. 2013, pp. 160–172.

CARON, MATHILDE; BOJANOWSKI, PIOTR; JOULIN, ARMAND; DOUZE, MATTHIJS. **Deep clustering for unsupervised learning of visual features**. Proceedings of the European conference on computer vision (ECCV). 2018, pp. 132–149.

CRESPO, JOAO BFP; AGUIAR, PEDRO MQ. **Revisiting complex moments for 2-D shape representation and image normalization**. IEEE transactions on image processing 20.10 (2011), pp. 2896–2911.

DU, SHAOYI; ZHENG, NANNING; XIONG, LEI; YING, SHIHUI; XUE, JIANRU. **Scaling iterative closest point algorithm for registration of m–D point sets**. Journal of Visual Communication and Image Representation 21.5-6 (2010), pp. 442–452.

EASTMAN, CM; EASTMAN, C; TEICHOLZ, P; SACKS, R. **BIM handbook: A guide to building information modeling for owners, managers, designers, engineers and contractors**. 2011.

EGGERT, DAVID W; LORUSSO, ADELE; FISHER, ROBERT B. **Estimating 3-D rigid body transformations: a comparison of four major algorithms**. Machine vision and applications 9.5 (1997), pp. 272–290.

ESTER, MARTIN; KRIEGEL, HANS-PETER; SANDER, JÖRG; XU, XIAOWEI, et al. **A density-based algorithm for discovering clusters in large spatial databases with noise.** kdd. Vol. 96. 34. 1996, pp. 226–231.

FAN, HAOQIANG; SU, HAO; GUIBAS, LEONIDAS J. **A point set generation network for 3d object reconstruction from a single image**. Proceedings of the IEEE conference on computer vision and pattern recognition. 2017, pp. 605–613.

FRIEDRICH, MARKUS; ILLIUM, STEFFEN; FAYOLLE, PIERRE-ALAIN; LINNHOFF-POPIEN, CLAUDIA. **A Hybrid Approach for Segmenting and Fitting Solid Primitives to 3D Point Clouds.** VISIGRAPP (1: GRAPP). 2020, pp. 38–48.

GAL, RAN; COHEN-OR, DANIEL. **Salient geometric features for partial shape matching and similarity**. ACM Transactions on Graphics (TOG) 25.1 (2006), pp. 130–150.

GIELINGH, WIM. **An assessment of the current state of product data technologies**. Computer-Aided Design 40.7 (2008), pp. 750–759.

GOMEZ-DONOSO, FRANCISCO; GARCIA-GARCIA, ALBERTO; GARCIA-RODRIGUEZ, J; ORTS-ESCOLANO, SERGIO; CAZORLA, MIGUEL. **Lonchanet: A sliced-based cnn architecture for real-time 3d object recognition**. 2017 International Joint Conference on Neural Networks (IJCNN). IEEE. 2017, pp. 412–418.

GRANGER, SÉBASTIEN; PENNEC, XAVIER. **Multi-scale EM-ICP: A fast and robust approach for surface registration**. European Conference on Computer Vision. Springer. 2002, pp. 418–432.

HAMERLY, GREG; ELKAN, CHARLES. **Learning the k in k-means**. Advances in neural information processing systems 16 (2003).

HANOCKA, RANA; FISH, NOA; WANG, ZHENHUA; GIRYES, RAJA; FLEISHMAN, SHACHAR; COHEN-OR, DANIEL. **Alignet: Partial-shape agnostic alignment via unsupervised learning**. ACM Transactions on Graphics (TOG) 38.1 (2018), pp. 1–14.

HANOCKA, RANA; METZER, GAL; GIRYES, RAJA; COHEN-OR, DANIEL. **Point2Mesh: a self-prior for deformable meshes**. arXiv preprint arXiv:2005.11084 (2020).

HARDIN, B; MCCOOL, D. **BIM and construction management: proven tools, methods, and workflows**. 2015.

HASSANI, KAVEH; HALEY, MIKE. **Unsupervised multi-task feature learning on point clouds**. Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019, pp. 8160–8171.

HEGDE, SINDHU; GANGISETTY, SHANKAR. **PIG-Net: Inception based deep learning architecture for 3D point cloud segmentation**. Computers & Graphics 95 (2021), pp. 13–22.

HERMOSILLA, PEDRO; RITSCHEL, TOBIAS; VÁZQUEZ, PERE-PAU; VINACUA, ÀLVAR; ROPINSKI, TIMO. **Monte carlo convolution for learning on non-uniformly sampled point clouds**. ACM Transactions on Graphics (TOG) 37.6 (2018), pp. 1–12.

HORN, BERTHOLD KP. **Closed-form solution of absolute orientation using unit quaternions**. Josa a 4.4 (1987), pp. 629–642.

IOFFE, SERGEY; SZEGEDY, CHRISTIAN. **Batch normalization: Accelerating deep network training by reducing internal covariate shift**. International conference on machine learning. PMLR. 2015, pp. 448–456.

JIANG, JINCEN; LU, XUEQUAN; OUYANG, WANLI; WANG, MEILI. **Unsupervised representation learning for 3d point cloud data**. arXiv preprint arXiv:2110.06632 (2021).

KANATANI, KEN-ICHI. **Analysis of 3-D rotation fitting**. IEEE Transactions on pattern analysis and machine intelligence 16.5 (1994), pp. 543–549.

KASS, ROBERT E; WASSERMAN, LARRY. **A reference Bayesian test for nested hypotheses and its relationship to the Schwarz criterion**. Journal of the american statistical association 90.431 (1995), pp. 928–934.

KIM, BYUNG CHUL; JEON, YOUNGJUN; PARK, SANGJIN; TEIJGELER, HANS; LEAL, DAVID; MUN, DUHWAN. **Toward standardized exchange of plant 3D CAD models using ISO 15926**. Computer-Aided Design 83 (2017), pp. 80–95.

KINGMA, DIEDERIK P; BA, JIMMY. **Adam: A method for stochastic optimization**. arXiv preprint arXiv:1412.6980 (2014).

KRITZINGER, WERNER; KARNER, MATTHIAS; TRAAR, GEORG; HENJES, JAN; SIHN, WILFRIED. **Digital Twin in manufacturing: A categorical literature review and classification**. IFAC-PapersOnLine 51.11 (2018), pp. 1016–1022.

KUROBE, AKIYOSHI; SEKIKAWA, YUSUKE; ISHIKAWA, KOHTA; SAITO, HIDEO. **Corsnet: 3d point cloud registration by deep neural network**. IEEE Robotics and Automation Letters 5.3 (2020), pp. 3960–3966.

LI, LINGXIAO; SUNG, MINHYUK; DUBROVINA, ANASTASIA; YI, LI; GUIBAS, LEONIDAS J. **Supervised fitting of geometric primitives to 3d point clouds**. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019, pp. 2652–2660.

LUCAS, BRUCE D; KANADE, TAKEO, et al. **An iterative image registration technique with an application to stereo vision**. Vancouver, British Columbia. 1981.

MARTINET, AURÉLIEN; SOLER, CYRIL; HOLZSCHUCH, NICOLAS; SILLION, FRANÇOIS X. **Accurate detection of symmetries in 3d**

shapes. ACM Transactions on Graphics (TOG) 25.2 (2006), pp. 439–464.

MITRA, NILOY J; GUIBAS, LEONIDAS J; PAULY, MARK. **Partial and approximate symmetry detection for 3d geometry**. ACM Transactions on Graphics (TOG) 25.3 (2006), pp. 560–568.

NGUYEN, TRUNG; PHAM, QUANG-HIEU; LE, TAM; PHAM, TUNG; HO, NHAT; HUA, BINH-SON. **Point-set distances for learning representations of 3d point clouds**. Proceedings of the IEEE/CVF International Conference on Computer Vision. 2021, pp. 10478–10487.

OSADA, ROBERT; FUNKHOUSER, THOMAS; CHAZELLE, BERNARD; DOBKIN, DAVID. **Shape distributions**. ACM Transactions on Graphics (TOG) 21.4 (2002), pp. 807–832.

PAULY, MARK; MITRA, NILOY J; WALLNER, JOHANNES; POTTMANN, HELMUT; GUIBAS, LEONIDAS J. **Discovering structural regularity in 3D geometry**. ACM SIGGRAPH 2008 papers. 2008, pp. 1–11.

PELLEG, DAN; MOORE, ANDREW W, et al. **X-means: Extending k-means with efficient estimation of the number of clusters.** Icml. Vol. 1. 2000, pp. 727–734.

PHARR, MATT; FERNANDO, RANDIMA. **Gpu gems 2: programming techniques for high-performance graphics and general-purpose computation**. Addison-Wesley Professional, 2005.

PRIM, ROBERT CLAY. **Shortest connection networks and some generalizations**. The Bell System Technical Journal 36.6 (1957), pp. 1389–1401.

PROCESS INDUSTRIES STEP CONSORTIUM. **STEP In The Process Industries: Process Plant Engineering Activity Model**. 1994.

QI, CHARLES R; SU, HAO; MO, KAICHUN; GUIBAS, LEONIDAS J. **Pointnet: Deep learning on point sets for 3d classification and segmentation**. Proceedings of the IEEE conference on computer vision and pattern recognition. 2017, pp. 652–660.

QI, CHARLES RUIZHONGTAI; YI, LI; SU, HAO; GUIBAS, LEONIDAS J. **Pointnet++: Deep hierarchical feature learning on point sets in a metric space**. Advances in neural information processing systems. 2017, pp. 5099–5108.

QI, QINGLIN; TAO, FEI; ZUO, YING; ZHAO, DONGMING. **Digital twin service towards smart manufacturing**. Procedia Cirp 72 (2018), pp. 237–242.

RAO, YONGMING; LU, JIWEN; ZHOU, JIE. **Global-local bidirectional reasoning for unsupervised representation learning of 3d point clouds**. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020, pp. 5376–5385.

RAVANBAKHSH, SIAMAK; SCHNEIDER, JEFF; POCZOS, BARNABAS. **Deep learning with sets and point clouds**. arXiv preprint arXiv:1611.04500 (2016).

REMELLI, EDOARDO; BAQUE, PIERRE; FUA, PASCAL. **Neuralsampler: Euclidean point cloud auto-encoder and sampler**. arXiv preprint arXiv:1901.09394 (2019).

RUMELHART, DAVID E; HINTON, GEOFFREY E; WILLIAMS, RONALD J. **Learning internal representations by error propagation**. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

SANTOS, PAULO IVSON NETTO; CELES FILHO, WALDEMAR. **Instanced rendering of massive cad models using shape matching**. 2014 27th SIBGRAPI Conference on Graphics, Patterns and Images. IEEE. 2014, pp. 335–342.

SCHNABEL, RUWEN; WAHL, ROLAND; KLEIN, REINHARD. **Efficient RANSAC for point-cloud shape detection**. Computer graphics forum. Vol. 26. 2. Wiley Online Library. 2007, pp. 214–226.

SHAO, GUODONG; HELU, MONEER. **Framework for a digital twin in manufacturing: Scope and requirements**. Manufacturing Letters 24 (2020), pp. 105–107.

SHARP, GREGORY C; LEE, SANG W; WEHE, DAVID K. **ICP registration using invariant features**. IEEE Transactions on Pattern Analysis and Machine Intelligence 24.1 (2002), pp. 90–102.

SILVA, LUCIANO; BELLON, OLGA REGINA PEREIRA; BOYER, KIM L. **Precision range image registration using a robust surface interpenetration measure and enhanced genetic algorithms**. IEEE transactions on pattern analysis and machine intelligence 27.5 (2005), pp. 762–776.

SONG, CHUNFENG; LIU, FENG; HUANG, YONGZHEN; WANG, LIANG; TAN, TIENIU. **Auto-encoder based data clustering**. Iberoamerican congress on pattern recognition. Springer. 2013, pp. 117–124.

SOUZA MOREIRA, ANDRÉ de. **Engenharia reversa em modelos cad utilizando descritores de forma e maquina de vetores de suporte**. PhD thesis. MA thesis. PUC–Rio, 2015.

TE, GUSI; HU, WEI; ZHENG, AMIN; GUO, ZONGMING. **Rgcnn: Regularized graph cnn for point cloud segmentation**. Proceedings of the 26th ACM international conference on Multimedia. 2018, pp. 746–754.

UMEYAMA, SHINJI. **Least-squares estimation of transformation parameters between two point patterns**. IEEE Transactions on Pattern Analysis & Machine Intelligence 13.04 (1991), pp. 376–380.

WANG, YECHAO; CAO, JINMING; LI, YANGYAN; TU, CHANGHE. **APM: Adaptive permutation module for point cloud classification**. Computers & Graphics 97 (2021), pp. 217–224.

WANG, YUE; SOLOMON, JUSTIN M. **Deep closest point: Learning representations for point cloud registration**. Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019, pp. 3523–3532.

WANG, ZHEN; ZHANG, LIQIANG; ZHANG, LIANG; LI, ROUJING; ZHENG, YIBO; ZHU, ZIDONG. **A deep neural network with spatial pooling (DNNSP) for 3-D point cloud classification**. IEEE Transactions on Geoscience and Remote Sensing 56.8 (2018), pp. 4594–4604.

WOLD, SVANTE; ESBENSEN, KIM; GELADI, PAUL. **Principal component analysis**. Chemometrics and intelligent laboratory systems 2.1-3 (1987), pp. 37–52.

WU, BICHEN; ZHOU, XUANYU; ZHAO, SICHENG; YUE, XIANGYU; KEUTZER, KURT. **Squeezesegv2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a lidar point cloud**. 2019 International Conference on Robotics and Automation (ICRA). IEEE. 2019, pp. 4376–4382.

YEW, ZI JIAN; LEE, GIM HEE. **3dfeat-net: Weakly supervised local 3d features for point cloud registration**. Proceedings of the European Conference on Computer Vision (ECCV). 2018, pp. 607–623.

ZAMORSKI, MACIEJ; ZIEBA, MACIEJ; KLUKOWSKI, PIOTR; NOWAK, RAFAŁ; KURACH, KAROL; STOKOWIEC, WOJCIECH; TRZCIŃSKI, TOMASZ. **Adversarial autoencoders for compact representations of 3D point clouds**. Computer Vision and Image Understanding 193 (2020), p. 102921.

ZHA, HONGBIN; IKUTA, MAKOTO; HASEGAWA, TSUTOMU. **Registration of range images with different scanning resolutions**. Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0. Vol. 2. IEEE. 2000, pp. 1495–1500.

ZINSSER, TIMO; SCHMIDT, JOCHEN; NIEMANN, HEINRICH. **Point set registration with integrated scale estimation**. International conference on pattern recognition and image processing. 2005, pp. 116–119.