# Low-dimensional Representations for Diverse Collections of 3D Surface Meshes

Dissertation
zur
Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von
**Sara Vera Hahner**
aus Fulda

Bonn, Dezember 2023

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn

1. Gutachter: Prof. Dr. Jochen Garcke
2. Gutachter: Prof. Dr. Martin Rumpf

Tag der Promotion: 29.01.2024
Erscheinungsjahr: 2024

# Abstract

The surfaces of 3D shapes are discretized by surface meshes that represent them in a processable way. This enables simulations of deforming components in CAE, animations for movies, or representations of human organs to plan operations. Since surface meshes are relevant in different areas, the amount of 3D surface meshes representing diverse shapes is constantly growing.

Since many datasets containing surface meshes are high-dimensional, we are interested in low-dimensional shape features that enable the analysis of the dataset structure. Nevertheless, reducing the redundancies while preserving the essential information is not trivial. We need to disentangle the features from the mesh representation to create global shape features invariant to the discretization by the mesh.

Many dimension reduction methods for surface meshes calculate spectral shape features exploiting the basis defined by decomposing the Laplace-Beltrami operator. More recently, machine learning approaches have been applied to learn features for surface meshes, especially autoencoders in combination with mesh coarsening, which facilitates the dimension reduction. Nevertheless, the approaches depend on fixed surface mesh connectivity in the studied datasets. This limits not only the types of datasets that can be analyzed but also the transfer of learned methods and knowledge to different data.

To address these issues, I introduce two novel learning-based approaches for encoding 3D surface meshes in a low-dimensional space.

In the first approach, I transform the discretization of the surfaces to meshes with locally regular connectivity and hierarchical meshing. It allows me to divide the surfaces into patches, to which I then apply different convolutional methods for learning hierarchical features. The resulting Convolutional Semi-Regular Mesh Autoencoder (CoSMA) reconstructs surfaces not presented during training and generalizes the deformation behavior of the surface patches. In addition, I introduce a flexible optimization-based semi-regular remeshing algorithm that can handle a wide range of surface meshes and is tailored to the learning method.

In the second approach, I use correspondence maps between shapes to define a low-dimensional basis where all shapes are represented independently of their mesh representation. To this end, I propose a spectral mesh pooling technique that establishes this universal latent space, breaking free from the traditional constraints of mesh connectivity. The resulting network is called Canonical Consistent Latent Basis-Autoencoder (CCLB-AE).

In comparison to baselines, I can apply my networks to larger and more diverse datasets, whereas baselines handle only surface meshes with fixed mesh connectivity. The methods successfully learn shape features that reveal the structure of several datasets from different domains. Additionally, the reconstructions of the proposed methods are of higher quality than those of the baseline models. Moreover, the smooth embedding space allows for the generation of shapes by combining the learned low-dimensional shape representations. I conduct further experiments to evaluate the patch-based approach to learning hierarchical features for other tasks and predict time series in the joint CCLB-autoencoder embedding space.

# Contents

# Notation and Acronyms

## Notations

### Surfaces

| | |
|---|---|
| $\mathcal{M}(V, \mathcal{F})$ | triangular polygonal mesh defined by $n$ vertices $V \subset \mathbb{R}^d$ and a set of triangular faces $\mathcal{F} \subset V \times V \times V$ |
| $V \subset \mathbb{R}^d$ | set of vertices $v_1, \dots, v_n$ |
| $n \in \mathbb{N}$ | number of vertices |
| $\mathcal{F} \subset V \times V \times V$ | set of faces, pointing to the vertices that define them |
| $\mathcal{E} \subset V \times V$ | set of edges, pointing to the vertices that define them |
| $A \in \mathbb{R}^{n \times n}$ | adjacency matrix of a graph or surface |
| $f : V \to \mathbb{R}^F$ | function defined on the vertices $V$ |
| $N_r(v)$ | $r$-ring neighborhood of vertex $v$ |
| $D\mathbb{R}^{n \times n}$ | diagonal degree matrix |
| $rl \in \mathbb{N}$ | refinement level of a semi-regular mesh |
| $d_V(\mathcal{M}, \mathcal{M}')$ | vertex-wise mean squared error between two triangular polygonal meshes $\mathcal{M}$ and $\mathcal{M}'$ |
| $d_C(\mathcal{M}, \mathcal{M}')$ | average chamfer distance between two triangular polygonal meshes |
| $d_H(\mathcal{M}, \mathcal{M}')$ | Hausdorff distance between two triangular polygonal meshes |
| $d_{OT}(\mathcal{M}, \mathcal{M}')$ | Optimal Transport distance between two triangular polygonal meshes |
| $d_{geod}(v, w)$ | geodesic distance between points $v$ and $w$ on a surface |
| $\mathcal{L}_C \in \mathbb{R}^{n \times n}$ | combinatorial graph Laplacian |
| $\mathcal{L}_N \in \mathbb{R}^{n \times n}$ | normalized graph Laplacian |
| $\mathcal{L}_B \in \mathbb{R}^{n \times n}$ | Laplace-Beltrami operator |
| $\mathbf{F} \in \mathbb{R}^{n \times F}$ | $F$-dimensional vertex-wise features |
| $F$ | dimension of the vertex-wise features |
| $\Phi \in \mathbb{R}^{n \times k}$ | projection matrix to $k$-dimensional spectral basis |
| $\bullet^\dagger$ | left Moore-Penrose pseudo-inverse |
| $\mathbf{A} \in \mathbb{R}^{k \times F}$ | $F$-dimensional spectral features |
| $T_{12} : \mathcal{M}_1 \to \mathcal{M}_2$ | vertex-wise point-to-point map |
| $\Pi_{12} \in \mathbb{R}^{n_2 \times n_1}$ | matrix representation of point-to-point map $T_{12}$ |
| $C_{21} \in \mathbb{R}^{k \times k}$ | functional map that maps functions defined in the spectral basis of $\mathcal{M}_2$ to the spectral basis of $\mathcal{M}_1$ |
| $\mathcal{G}$ | graph describing a functional map network |

| | |
|---|---|
| $Y_i \in \mathbb{R}^{k_1 \times k_1}$ | consistent latent basis functions to mesh $\mathcal{M}_i$ given a functional map network |
| $k$ or $k_1$ | dimension of the functional maps for a shape collection |
| $\widetilde{Y}_i \in \mathbb{R}^{k_1 \times k_2}$ | canonical consistent latent basis (CCLB) functions to shape $\mathcal{M}_i$ given a functional map network |
| $\mathbf{A}_i^{CCLB} \in \mathbb{R}^{k_2 \times F}$ | $F$-dimensional spectral features in the CCLB |

## Learning

| | |
|---|---|
| $f_\theta : \mathbb{R}^n \to \mathbb{R}^m$ | neural network with learnable parameters $\theta$ |
| $x_i \in \mathbb{R}^n, i = 1, \ldots, d$ | $d$ input data points |
| $y_i \in \mathbb{R}^m, i = 1, \ldots, d$ | corresponding output data |
| $\hat{y}_i$ | predicted output |
| $l : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ | training loss function, for example, the MSE $l_{MSE}$ or cross entropy $l_{CE}$ |
| $a_j : \mathbb{R} \to \mathbb{R}$ | activation function |
| $l_j : \mathbb{R}^{n_j} \to \mathbb{R}^{n_{j+1}}$ | function describing layer $j$ of a neural network |
| $n_j \in \mathbb{N}$ | number of nodes of layer $j$ |
| $\mathbf{o}$ | output of a layer |
| $W_j \in \mathbb{R}^{n_{j+1} \times n_j}$ | weight matrix of fully connected layer $j$ |
| $\mathbf{b}_j \in \mathbb{R}^{n_{j+1}}$ | bias term of fully connected layer $j$ |
| $(W_{conv})_j \in \mathbb{R}^{k \times k}$ | one-channel convolutional kernel matrix at layer $j$ |
| $*$ | convolution operator |
| $h_i \in \mathbb{R}^{hr}$ | feature representation of input $x_i$ calculated by encoder $en$ of an autoencoder |
| $en : \mathbb{R}^n \to \mathbb{R}^{hr}$ | encoder of an autoencoder network |
| $de : \mathbb{R}^{hr} \to \mathbb{R}^n$ | decoder of an autoencoder network |
| $hr$ | latent dimension of the autoencoder |

## CoSMA

| | |
|---|---|
| $v_p \in \mathbb{R}^{m \times 3}$ | $m$ vertex coordinates of patch $p$ |
| $\mathbf{W}_{hex}^{KS}$ | hexagonal convolutional filter with $1 + \sum_{k=1}^{KS} 6k$ trainable parameters |
| $KS \in \mathbb{N}$ | kernel size of a hexagonal convolutional filter |
| $\text{HexConv}_{KS}$ | function describing the hexagonal spatial convolution |
| $W_{Ch} \in \mathbb{R}^K$ | trainable parameters of a one-channel Chebyshev convolutional filter |
| $K \in \mathbb{N}$ | filter size of a Chebyshev convolutional filter |
| $\text{ChebConv}_K$ | function describing the Chebyshev spectral convolution |
| $PS \in \mathbb{N}$ | padding size |
| $\mathbf{Pad}(v_p)$ | padded patch $v_p$ |

| | |
|---|---|
| $\mathbf{P}_{hex}$ | pooling function applied to a regularly meshed patch |
| $\mathbf{UP}_{hex}$ | unpooling function |
| $hr_p$ | patch-wise embedding dimension for CoSMA models |
| $en_{CoSMA}$, $de_{CoSMA}$ | CoSMA encoder and decoder |
| $\text{MSE}_{SA}(v_p, \hat{v}_p) \in \mathbb{R}$ | surface-aware mean squared error between input patch coordinates $v_p$ and their reconstruction $\hat{v}_p$ |

## CCLB-autoencoder

| | |
|---|---|
| $\mathbf{P}_{Spec} : \mathbb{R}^{n \times F} \to \mathbb{R}^{k_2 \times F}$ | spectral mesh pooling |
| $\mathbf{UP}_{Spec} : \mathbb{R}^{k_2 \times F} \to \mathbb{R}^{n_t \times F}$ | spectral mesh unpooling |
| $\mathcal{M}_t$ | template mesh with $n_t$ vertices |
| $en_{CCLB}$, $de_{CCLB}$ | encoder and decoder of the CCLB-autoencoder |
| $V_{rec} \in \mathbb{R}^{n \times 3}$ | reconstructed vertex positions |
| $L_1 : \mathbb{R}^{n \times 3} \times \mathbb{R}^{n_t \times 3} \to \mathbb{R}$ | point-to-point loss for evaluating a CCLB-autoencoder reconstruction |
| $L_2 : \mathbb{R}^{n \times 3} \times \mathbb{R}^{n_t \times 3} \to \mathbb{R}$ | reconstruction loss for evaluating a CCLB-autoencoder reconstruction |
| $D^V \in \mathbb{R}^{n \times n}$ | vertex-to-vertex distance matrix of vertices $V$ |
| $\lambda_{rec} \in \mathbb{R}$ | weight of the reconstruction loss for CCLB-AE |

## Semi-regular Remeshing

| | |
|---|---|
| $\mathcal{M}_{coarse,k}$ | coarse base mesh with $k$ faces |
| $\mathcal{M}_{IR}$ | mesh with irregular connectivity |
| $\mathcal{M}_{SR,j}$ | semi-regular mesh of refinement level $j$ with $n_j$ vertices $V_{SR,j}$ |
| $d_{C,geod}(\mathcal{M}, \mathcal{M}') \in \mathbb{R}$ | chamfer distance weighted by geodesic distance |
| $d_B(\mathcal{M}, \mathcal{M}') \in \mathbb{R}$ | boundary loss between two triangular polygonal meshes |
| $r_{\mathcal{E}}(\mathcal{M} \in \mathbb{R})$ | edge length regularizer |
| $r_L(\mathcal{M}) \in \mathbb{R}$ | Laplacian smoothing regularizer |
| $r_N(\mathcal{M}) \in \mathbb{R}$ | regularizer enforcing normal consistency |
| $\omega_B$, $\omega_{\mathcal{E}}$, $\omega_L$, $\omega_N \in \mathbb{R}$ | regularization weights |
| $\boldsymbol{\zeta}_j \in \mathbb{R}^{n_j}$ | deformation vector fitting vertices of $\mathcal{M}_{SR,j}$ to $\mathcal{M}_{IR}$ |

## Experiments

| | |
|---|---|
| $\text{RecErr}(\mathcal{M}, \mathcal{M}_{rec}) \in \mathbb{R}$ | reconstruction error between $\mathcal{M}$ and $\mathcal{M}_{rec}$ used for evaluation |
| $L_i, \hat{L}_i$ | true and predicted face labels |
| $V_{pred} \in \mathbb{R}^{n \times 3}$ | predicted vertex positions |

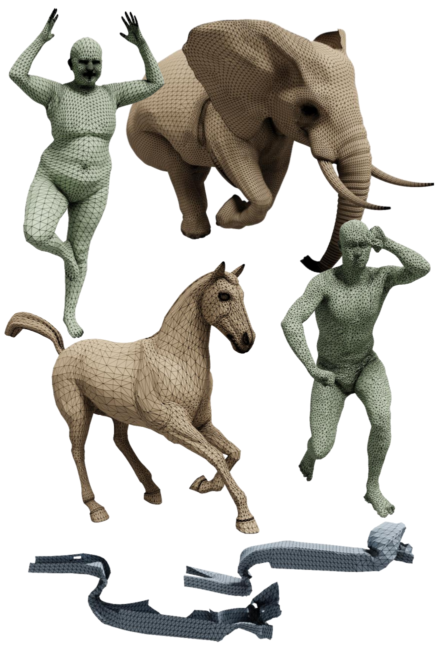
## Acronyms

| | |
|---|---|
| CCLB-AE | canonical consistent latent basis autoencoder |
| CoSMA | convolutional semi-regular mesh autoencoder |
| 3D | three-dimensional |
| 2D | two-dimensional |
| SR | semi-regular |
| OT | optimal transport |
| p2p map | point-to-point map |
| FM | functional map |
| FMN | functional map network |
| CCLB | canonical consistent latent basis |
| hks | heat kernel signature |
| wks | wave kernel signature |
| NN | neural network |
| MSE | mean squared error |
| CE | cross entropy |
| CNN | convolutional neural network |
| AE | autoencoder |
| PCA | principal component analysis |
| VAE | variational autoencoder |
| KL | Kullback–Leibler (divergence) |
| GAN | generative adversarial network |
| DMD | dynamic mode decomposition |
| GT | ground truth |
| kNN | $k$-nearest neighbor |
| mIoU | mean Intersection-over-Union |
| GPU | graphics processing unit |

# 1 Introduction

In many applications, we are interested in deformations of shapes or processes on their surface. This includes simulations or digital twins for cars and airplanes, animated shapes for computer games and movies, weather predictions on the surface of the earth, or models of human organs for planning operations. In order to digitalize and process these tasks on a computer, the surfaces of the shapes have to be discretized and represented in a processable way. 3D surface meshes are flexible surface representations that discretize the surface by multiple small faces, mostly of triangular or rectangular shape, similar to how a soccer ball is divided into small patches. Since this representation is common and its creation is, in many cases, done automatically, the amount of 3D surface meshes representing diverse shapes is constantly growing. For the analysis of these meshes and the processing to solve the specific tasks, we need to extract necessary information from the data sets. Nevertheless, this extraction of information from 3D data is not trivial. Since the shape size, deformation, and mesh representation depend on the application, hand-crafted solutions combined with human inspection are often applied to structure and analyze the data.

3D surface meshes are generally represented by tens of thousands of faces. They contain abundant features and are, therefore, high-dimensional and unwieldy. Analyzing high-dimensional data is time-consuming and sometimes not manageable using the available computational resources. Also, redundancies in the data make comparative analysis difficult. Low-dimensional representations of the data evade these difficulties by reducing the redundancies in the data while preserving the essential information. Since these representations are of lower dimensionality, the data needs less storage, the dataset structure becomes visible, and many times, they allow easier handling by a computer. For example, while for a human, the meanings of words or even sentences are easily compared and put into context, the success of recent large language models [DCLT19, TLI+23] depends highly on significant word embeddings. These word embeddings represent the meanings of words of different lengths by real values [MSC+13, PNI+18].


Selection of surface meshes.

In this work, I propose novel approaches to calculate low-dimensional representations for surface meshes and evaluate the learned representations in follow-up tasks analyzing the datasets. Because most state-of-the-art methods specialize in one type of shape, for example, only humans, with fixed mesh representations, my proposed approaches and methods loosen this strong constraint and aim to handle more diverse collections of 3D surface meshes.

### Calculating Low-dimensional Representations

Low-dimensional representations are obtained using various techniques. Dimension reduction methods project the data to a lower dimensional space using, for example, principal component analysis [Pea01]. More recent machine learning methods use so-called autoencoders to learn a function from the data that calculates the low-dimensional features. The learned features are trained to fulfill requirements for specific tasks, from reconstruction of the input to classification, anomaly detection, or interpretability [Mur22]. The approach and method to learn the features depend highly on the data type. While convolutional neural networks learn significant features for images, transformer models with attention modules calculate meaningful representations for texts. Since these methods calculate low-dimensional representations that are not known beforehand but are to be identified by the methods, they fall in the category of unsupervised learning.

### Low-dimensional Representations for 3D Surface Meshes

Many dimension reduction methods for surface meshes representing 3D shapes calculate spectral shape features by projecting to a basis defined by eigenvectors of the so-called Laplace-Beltrami operator or using the eigenvalues of the operator [KG00, RWP06, RBG$^+$09, ITG19]. More recently, machine learning approaches have been applied to learn features for surface meshes [BBL$^+$17, BBCV21, WNEH22, SACO22]. Because the surface meshes are locally two-dimensional, many of these methods are inspired by convolutional neural networks for 2D images. Convolutional neural networks for 2D images calculate global features by sliding local filters along the vertical and horizontal axes and then downsampling the resolution by applying the so-called pooling. Figure 1.1 visualizes these operations. Nevertheless, surface meshes do not align along a global grid structure that facilitates these operations. While existing convolutional layers for surface meshes calculate local vertex-wise features, there are no general solutions for pooling on surface meshes. However, methods that learn surface representations must disentangle the local features from the given surface mesh to calculate global lower dimensional shape features. If all of the meshes share the same mesh connectivity, many methods coarsen all the meshes iteratively by reducing the number of faces in a fixed pattern and calculating hierarchical features at different levels of detail [RBSB18, BBP$^+$19, ZWL$^+$20]. Autoencoders apply this downsampling by mesh coarsening to calculate meaningful compact representations for the shapes, and they provide a function for the reconstruction that allows shape editing or generation.

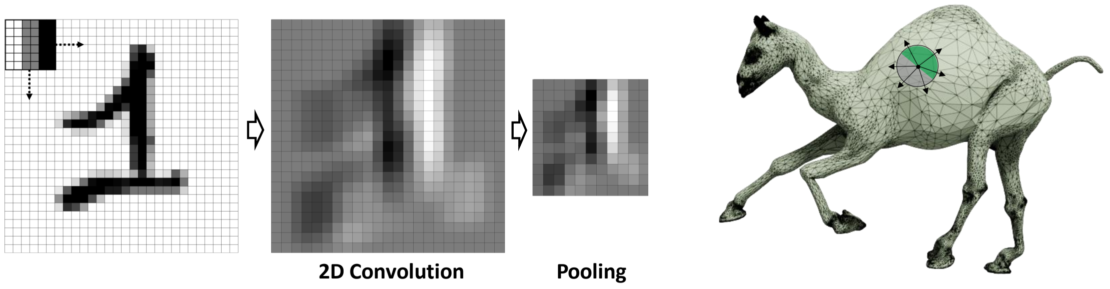


Figure 1.1: Convolution and pooling on 2D images and transferring the idea of convolutional filters to surface meshes, where filters cannot be aligned nor translated along a global grid.

### Low-dimensional Representations for 3D Surface Meshes of Different Mesh Connectivity

Both dimension reduction methods for surface meshes, based on the decomposition of a discrete Laplace-Beltrami operator or coarsening of the mesh, depend on fixed mesh connectivity for all the samples. The discrete Laplace-Beltrami operator changes under changes in mesh connectivity or non-isometrically deformations (not preserving distances on the shape), and a method coarsening the surface mesh for dimension reduction cannot be transferred to a different mesh representation. That means a learned method is no longer fitting, and calculated low-dimensional spectral features of different shapes cannot be compared. Given the substantial limitations of existing methods, this thesis develops and evaluates machine learning approaches to learn meaningful low-dimensional representations using methods that handle various shapes with different mesh connectivities.

In a first approach, I transform the discretization of the surfaces to semi-regular meshes with locally regular connectivity and hierarchical meshing. It allows me to divide the surface meshes into patches, to which I then apply convolutional methods for learning hierarchical features. The resulting **Convolutional Semi-Regular Mesh Autoencoder** (CoSMA) reconstructs surfaces not presented during training and generalizes the deformation behavior of the surface patches. Since existing semi-regular remeshing algorithms place strong requirements on the input surface meshes, I introduce a flexible optimization-based **semi-regular remeshing algorithm** that can handle a wide range of surface meshes and is tailored to the learning method.

In a second approach, I use correspondence maps between shapes to define a low-dimensional basis where all shapes are represented independently of their mesh representation without projecting the meshes to a template or reference shape. To this end, I propose a spectral mesh pooling technique that establishes this universal latent space, breaking free from the traditional constraints of mesh connectivity. This network is called **Canonical Consistent Latent Basis-Autoencoder** (CCLB-AE) because the CCLB defines the universal latent space.

I apply these two general approaches to learn shape features for several datasets and evaluate the reconstruction quality and the significance of the learned features. Both

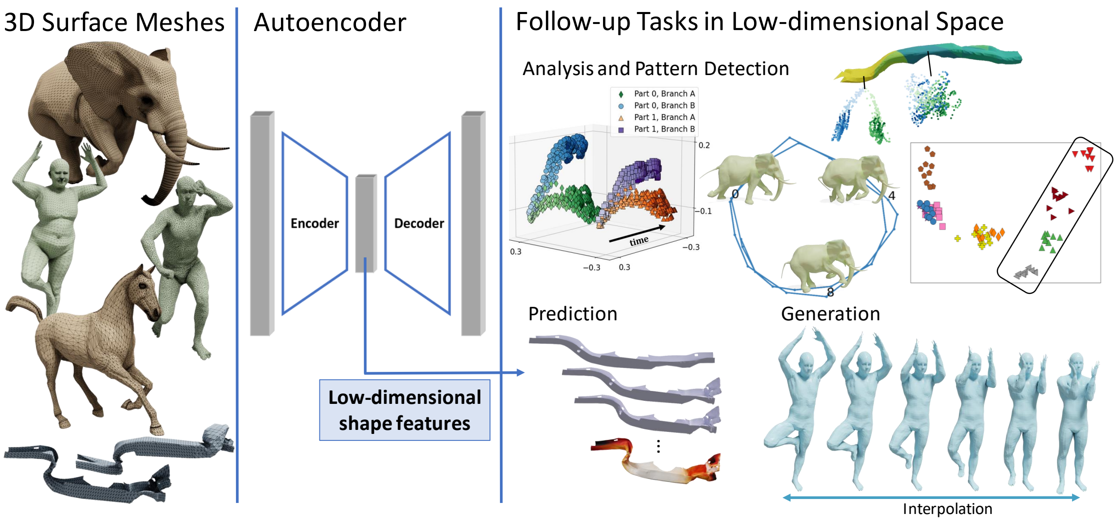


Figure 1.2: Pipeline calculating low-dimensional representations of surface meshes using autoencoders to facilitate the analysis of collections of 3D shapes.

methods are applied to four different datasets from different domains: surface meshes representing animals and humans in various poses, components from a car model deforming in a car crash simulation, and a synthetic triangular mesh dataset. Model-specific loss functions optimize both models, in the case of the CoSMAs leveraging the division of the surface into patches and for the CCLB-autoencoder regularizing erroneous correspondence maps. I can apply my networks to larger and more diverse datasets compared to baselines handling only surface meshes with fixed mesh connectivity. Additionally, the reconstructions of the proposed methods are of higher quality than those of the baseline models. Moreover, I test the smoothness of the learned embedding space by interpolating and combining the learned low-dimensional shape representations. I conduct further experiments to evaluate the patch-based approach to learning hierarchical features for other tasks. Finally, I also apply methods for time series prediction in the joint low-dimensional embedding space created by the novel spectral mesh pooling to evaluate whether this common space allows the joint analysis of several shapes of different connectivity. Figure 1.2 visualizes the pipeline of learning surface mesh autoencoders that calculate low-dimensional shape representations, facilitating analysis tasks on the datasets.

## 1.1 List of Accepted Papers and Collaborations

Valuable collaborations contributed to the work presented in this thesis, and part of it has been published in a series of joint publications:

[HG22] Sara Hahner and Jochen Garcke. Mesh Convolutional Autoencoder for Semi-Regular Meshes of Different Sizes. In *Proceedings of the IEEE/CVF*

*Winter Conference on Applications of Computer Vision*. pages 2344-2353, 2022, doi: 10.1109/WACV51458.2022.00240.

[HKG22] Sara Hahner, Felix Kerkhoff, and Jochen Garcke. Transfer Learning Using Spectral Convolutional Autoencoders on Semi-Regular Surface Meshes. In *Proceedings of the First Learning on Graphs Conference*. volume 198, pages 18:1–18:19. PMLR, 2022.

[HAGO23] Sara Hahner, Souhaib Attaiki, Jochen Garcke, and Maks Ovsjanikov. Unsupervised Representation Learning for Diverse Deformable Shape Collections. Accepted at *International Conference on 3D Vision 2024*.

Felix Kerkhoff contributed to the implementation of the spectral CoSMA [HKG22]. The work on the CCLB-Autoencoder [HAGO23] started during a visit of the research group of Maks Ovsjanikov at École Polytechnique. I published the paper in equal collaboration with Souhaib Attaiki. While I developed the model architecture and spectral mesh pooling, he contributed the unsupervised case and learned the correspondence maps between the shapes.

I also want to thank the students who implemented additional experiments for me as student assistants at Fraunhofer SCAI: Felix Blanke implemented mesh segmentation experiments using a CoSMA-inspired patch-based approach. Ole Schügl coded the time series prediction in the low-dimensional embedding space and additional regularizers for the mesh fitting of the introduced semi-regular remeshing. Lennart Metz implemented additional regularizers for the mesh coarsening.

## 1.2 Structure

At first, three chapters give an overview of relevant work, starting with chapter 2 on surfaces, their discretization, semi-regular remeshing, and their analysis using the Laplace-Beltrami operator and its decomposition. Chapter 3 summarizes related work on machine learning and neural networks. Chapter 4 combines the first two and states related work to learning methods and approaches on and for surface meshes.

Chapter 5 introduces the two proposed autoencoding approaches, CoSMA and the CCLB-autoencoder, for feature learning for surface meshes that do not share the same mesh connectivity. Since one approach requires the surfaces to be represented by semi-regular meshes, in chapter 6, I introduce the proposed remeshing method tailored to fulfill the requirements of the CoSMA. Chapter 7 introduces the various datasets from diverse sources on which the proposed models are trained and evaluated. This chapter also evaluates the remeshing results on the datasets. Chapter 8 presents the results on representation learning of surface meshes using the proposed autoencoders. It evaluates their reconstruction quality and compares it to several baselines. Afterward, the low-dimensional representations are analyzed, and the embedding spaces are evaluated for shape generation. In chapter 9, additional experiments evaluate parts of the CoSMA and CCLB-autoencoder approaches on human body part segmentation and time series prediction in the embedding space. Finally, chapter 10 concludes this thesis, states open questions, and provides an outlook.

## 1.3 Acknowledgements

This endeavor would not have been possible without the support of my professor, Prof. Dr. Jochen Garcke. He always took his time to answer my questions and provided promising research directions. Also, he provided excellent working conditions in the group at Fraunhofer SCAI and gave me the necessary academic time and freedom to develop and investigate the research topics of this thesis. Additionally, I could not have undertaken this journey without my defense committee. Therefore, I am grateful to Prof. Dr. Martin Rumpf for being the co-reviewer of this thesis, to Prof. Dr. Sergio Conti for chairing my doctoral committee, to Prof. Dr. Hilde Kühne for representing the computer science institute in the committee, and to Prof. Dr. Angkana Rüland for participating in the review process. I would also like to express my gratitude to Prof. Dr. Maks Ovsjanikov, whose insights helped me develop new ideas during my research stay at École Polytechnique.

I give special thanks to my colleagues at Fraunhofer SCAI. I want to point out the automotive team, who provided me with essential motivation and data for my research and insights into the applications of my ideas. Together with the other doctoral students in the team, I also had great and insightful conversations about how to proceed and successfully finish our studies. Thanks should also go to the indispensable IT department at Fraunhofer SCAI, which maintains the computing cluster at our research institute and assisted me with many questions. Of course, I thank all my colleagues and dearest friends from my Math studies for their editing help, which greatly supported me in the last weeks before handing in my dissertation.

Last but not least, I would be remiss if I did not mention my family, especially my parents and my dear husband Bruno. My parents always have an open ear for my concerns, and I can always count on their help. Bruno's belief in me has kept my spirits up and my motivation high throughout this process. Without his support and encouragement, I would not have been able to complete my projects as successfully as I have.

# 2 Surfaces

This work analyzes surfaces that are two-dimensional manifolds embedded in three-dimensional space. This chapter introduces general related work regarding surfaces and provides a solid foundation for understanding the building blocks of the proposed methods for analyzing 3D shapes.

For their computational analysis, the surfaces are discretized by surface meshes. Following a general definition in section 2.1, the section focuses on the discretization of surfaces and how we influence and condition the discretization by a mesh to facilitate their analysis by neural networks. Additionally, section 2.2 explores distances measured on and between surfaces that I use to evaluate and optimize the proposed methods of this work. Afterward, section 2.3 introduces Laplace operators that are calculated for surface meshes and describe their smoothness. The decomposition of the Laplace operator defines a basis, also referred to as spectral basis, in which functions defined on the surfaces can be represented in lower dimensionality. Section 2.4 motivates why many tasks can be solved more efficiently in this representation and introduces functional maps that take advantage of the fact that a representation in the basis of the Laplace operator can be of lower dimensionality. Finally, section 2.5 introduces spectral shape descriptors, whose calculation also involves this spectral basis.

## 2.1 Surface Meshes

For the computational analysis, the surfaces in $\mathbb{R}^3$ are discretized by a surface mesh defined by vertices and faces, also referred to as elements. The proposed and baseline mesh autoencoders handle triangular polygonal meshes where the faces are triangular.

**Definition 1** (Triangular Polygonal Mesh)**.** *A triangular polygonal mesh $\mathcal{M}$ is defined by $n$ vertices $V \subset \mathbb{R}^d$ and a set of triangular faces $\mathcal{F} \subset V \times V \times V$, which describe the shape surface and point to the vertices that define them. The edges $\mathcal{E} = \{\{v_1, v_2\} \in V \times V \mid \exists\ f \in \mathcal{F}$ s.t. $v_1 \in f$ and $v_2 \in f\}$ are undirected, i.e. $(v, w) \in \mathcal{E} \Rightarrow (w, v) \in \mathcal{E}$.*

Since this work considers 3D triangular surface meshes, it holds $d = 3$.

In other areas, for example, Computer Aided Engineering, quadrilateral faces are more common when the component structure allows it by being relatively flat. Quadrilateral faces generally lead to regular meshes and a lower number of total faces, which provides high accuracy and efficiency when simulating processes on them. Nevertheless, the creation of quadrilateral meshes is more complicated on complex or curved structures. Since every quadrilateral mesh can be remeshed to be a triangular mesh by splitting every quadrilateral face into two triangular ones, I only focus on the more versatile triangular polygonal meshes in this work.

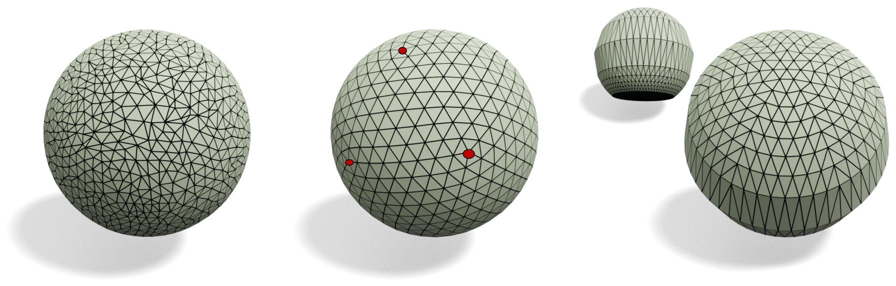


Figure 2.1: Irregular (left) and semi-regular (center, irregular vertices in red) surface mesh of a sphere, and a regular (right) one of a spherical cap.

Additionally, we define the adjacency matrix $A \in \mathbb{R}^{n \times n}$ of a graph or surface that stores all the edges. Its elements $A_{ij}$ are one if there is an edge from vertex $v_i$ to $v_j$ and zero otherwise. Since the polygonal meshes are defined using undirected edges ($(v_i, v_j) \in \mathcal{E} \Rightarrow (v_j, v_i) \in \mathcal{E}$), the adjacency matrix is symmetric. In some cases, the edges $(v_i, v_j)$ can be assigned a weight $w_{ij}$, which is then stored in the adjacency matrix $A_{ij} = w_{ij}$.

Given a triangular polygonal mesh $\mathcal{M}$, we define more characteristics for the mesh, its edges, and vertices. If every edge $e \in \mathcal{E}$ is adjacent to at most two faces in $\mathcal{F}$, the mesh is a manifold or surface mesh. An edge $e \in \mathcal{E}$ is a boundary edge if it is adjacent to exactly one face in $\mathcal{F}$. The vertices $v \in V$ of a mesh $\mathcal{M}$ can store information defined by a function $f : V \rightarrow \mathbb{R}^F$.

For each vertex $v \in V$, the $r$-ring neighborhood $N_r(v)$ are all the vertices that are connected to $v$ by at most $r$ edges:

$$\begin{aligned} N_r(v) = \{w \in V \mid\ & w \text{ and } v \text{ connected by} \\ & r \text{ or less edges in } \mathcal{E} \text{ and } v \neq w\} \subset V. \end{aligned} \tag{2.1}$$

Then, the degree or valence of a vertex $v \in V$ is the size of its one-ring neighborhood $N_1(v)$. The diagonal matrix of size $n \times n$ with the vertex degrees on its diagonal is referred to as the diagonal degree matrix $D$.

For triangular surface meshes, we refer to a vertex with degree six as regular, and a triangular surface mesh has regular connectivity if all vertices in $V$ that do not lie on the boundary are regular [BKP+10]. When working with functions that operate on the vertices and their 1-ring neighborhood, regularity in their degrees facilitates the function definition, efficiency of the computations, and calculations on a GPU because the 1-ring neighborhoods are of constant size. Nevertheless, a triangular surface mesh with only regular vertices has limited representation power. Only surface meshes that are topologically (part of) a torus can be represented by a regular mesh [BKP+10]. Nevertheless, most meshes in the considered datasets are topologically a sphere, having genus 0. These meshes must have some irregular vertices in their triangular polygonal mesh representation, see Figure 2.1.

Images represented in pixels can also be interpreted as two-dimensional triangular surface meshes of rectangular shape in $\mathbb{R}^2$ [BBL+17]. Every pixel is a vertex, the feature

| Characteristics | 2D image | Surface mesh | Semi-regular surface mesh |
|---|---|---|---|
| Data | information saved on vertices | | |
| Grid structure | global structure | locally Euclidean | locally Euclidean |
| Connectivity | fixed | - | semi-regular |
| Distance to neighbors | fixed | - | - |
| Size of instances | similar | highly different | highly different with similar local patches |

Table 2.1: Relevant characteristics of images and surface meshes in $\mathbb{R}^3$ for convolutional neural networks

function $f$ outputs the pixel color values, and edges connect every vertex to the eight neighboring pixels (horizontal, vertical, and diagonal).

### 2.1.1 Semi-regular Surface Meshes

Semi-regular (SR) (or subdivision connectivity) meshes are a flexible representation of surfaces in $\mathbb{R}^3$ that have a regular local structure and allow irregularities at selected vertices, whose positions can be controlled [BKP+10, EDD+95, GVSS00, LSS+98, PRS15].

We follow the definition of semi-regular meshes from [PRS15], which gives one condition on the specific structure: iteratively merging four triangular faces into one leads to a low-resolution mesh. This means that all vertices of a semi-regular mesh are regular (i.e., have six neighbors) besides the vertices of the low-resolution mesh, see Figure 2.1. Therefore, a semi-regular mesh is obtained by regular subdivision of a low-resolution mesh that can be irregular.

The overview in Table 2.1 shows that certain mesh characteristics of semi-regular meshes are closer to the ones of 2D images than those of general surface meshes.

Note that the iterative subdivision of the low-resolution mesh automatically defines a multi-scale structure. For this reason, semi-regular meshes are well suited for multi-resolution analysis, coarse-to-fine surface modeling, and of interest to adapt wavelets to surfaces and geometry compression [Mal89, KSS00, PRS15, MLDH15, LKC+20]. Later on, this hierarchical structure allows for a definition of a local pooling operation on the semi-regular meshes that can be applied to the piecewise regular structures.

### 2.1.2 Remeshing to Semi-regular Meshes

The authors of [PRS15, KPF+20] compare different semi-regular remeshing algorithms to each other. At first, all algorithms build a base mesh that is a coarse approximation of the irregular mesh. Classic incremental mesh simplification algorithms are based on edge collapses [Hop96, GH97]. Given the coarse base mesh, the semi-regular mesh is generated by iteratively refining the base mesh and adjusting it to fit the shape of the irregular mesh. The refinement level $rl$ states the number of times each face of the coarse base mesh is iteratively subdivided. The number of faces in the final semi-regular mesh is then

$$n_F^{semireg} = 4^{rl} \cdot n_{\mathcal{F}}^{c}, \tag{2.2}$$

with $n_{\mathcal{F}}^{c}$ being the number of faces describing the coarse base mesh. The iteratively refined faces of the irregular base mesh are called patches of the semi-regular mesh. All the vertices of the patch, but the three corner vertices and possibly boundary vertices, have six neighbors.

Some algorithms for subdivision surfaces, such as Loop [Loo87] and Butterfly [DLG90], create meshes that fulfill the definition of semi-regular meshes. The definition of subdivision surfaces also includes a deterministic, recursive subdivision mechanism that creates the upsampled surface meshes. Their limit surface then refers to the surface that is defined by an infinite number of upsampling/subdivision iterations. There is a rich theory that connects these limit surfaces to traditional splines [Zor07]. Nevertheless, since the goals of these algorithms are general convergence and smoothness properties, the positions of the vertices are calculated by taking a weighted average based on the local neighborhood. Therefore, the resulting meshes are overly smooth, see Figure 2.2.

Some remeshing algorithms build on the traditional algorithms for subdivision surfaces and aim to solve the over-smoothing. MAPS [LSS$^+$98] builds a self-parameterization of the original mesh over a semi-regular base domain. The non-linear subdivision method Neural Subdivision [LKC$^+$20] builds on MAPS and predicts the positions of new vertices using a neural network. While MAPS can handle surfaces with boundaries, it depends on the local mesh density of the irregular mesh, see the remeshing results for the mesh representing a human in Figure 2.2. Also, MAPS fails if there are non-manifold vertices in the interior of the surface. Neural Subdivision [LKC$^+$20] uses an adapted mesh decimation algorithm that prevents edge collapses causing poor quality of triangles. However, this restricts the lowest coarse mesh size that the algorithm reaches for training. Also, the algorithm is restricted to meshes without boundaries and non-manifold edges.

## 2.2 Distances for and on Surface Meshes

When comparing reconstructed or predicted surface meshes to each other and when evaluating remeshing results, measures determine how similar two meshes are. This section defines distances for comparing two surface meshes representing the same underlying surface and introduces the geodesic distance on a surface mesh.

### 2.2.1 Distances between Surface Meshes

To compare a remeshed surface mesh to a target mesh or a reconstructed mesh from an autoencoder to the input mesh, we need to calculate a distance or a measure of dissimilarity between the two surface meshes.

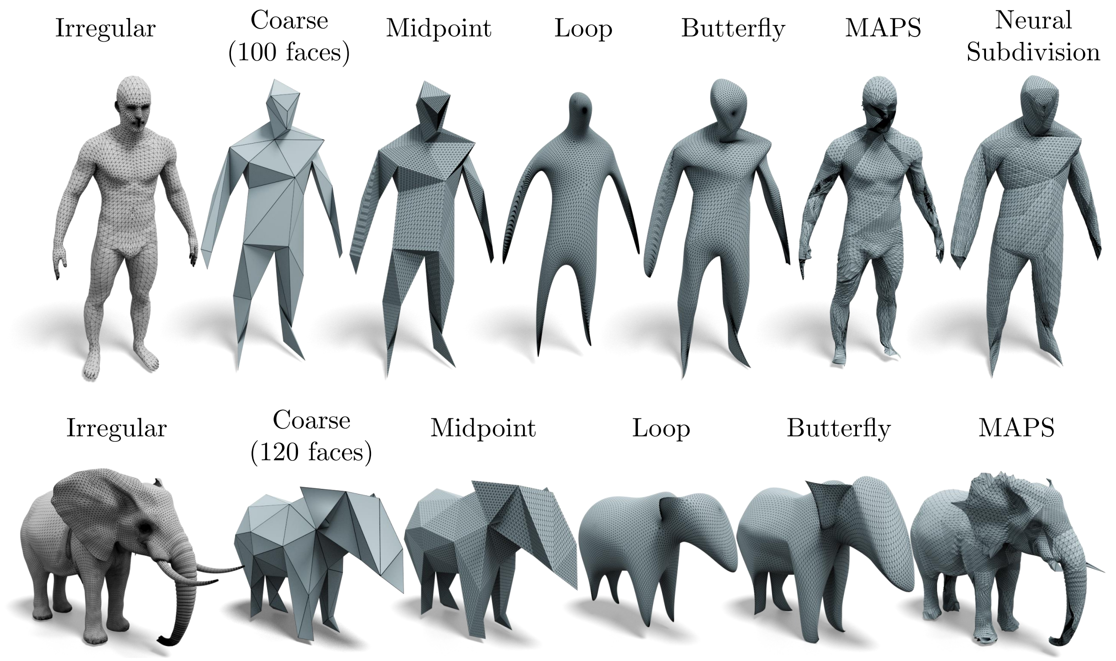


Figure 2.2: Different approaches for remeshing to semi-regular meshes of refinement level 4. The irregular mesh is the original mesh. The coarse mesh is input to the Loop and Butterfly subdivision algorithms. The midpoint method inserts new vertices at the midpoints of the edges. MAPS creates its own coarse base mesh of the same number of faces. Neural Subdivision downsamples the human mesh to 250 faces for training and cannot handle the elephant. The human and elephant meshes are from the *FAUST* and *GALLOP* datasets, respectively, which are both introduced in chapter 7.

Two surface meshes $\mathcal{M}(V, \mathcal{F})$ and $\mathcal{M}'(V', \mathcal{F})$, which share the same mesh connectivity, whose vertices are in the same order, and which represent the same underlying surface, can be compared using a simple **vertex-wise mean squared error**

$$d_V(\mathcal{M}, \mathcal{M}') = \frac{1}{|V|} \sum_{v \in V, v' \in V'} \|v - v'\|_2^2. \tag{2.3}$$

Nevertheless, we require a distance measure that compares two mesh representations $\mathcal{M}(V, \mathcal{F})$ and $\mathcal{M}'(V', \mathcal{F}')$ to each other, which do not share the same mesh connectivity, for example, when comparing an irregular representation of a surface to a semi-regular one. If a parametrization or point-to-point (p2p) map exists between the two meshes, we can map one mesh to the other, and calculate a vertex-wise mean squared error. Often, this parametrization between the meshes is not given, and in order to calculate a distance, the meshes are registered or somehow put in correspondence [SFC+21, WPZ+21].

The **average chamfer distance** relies on nearest neighbor projections [Bor84] between $\mathcal{M}(V, \mathcal{F})$ and $\mathcal{M}'(V', \mathcal{F}')$. It randomly samples points $S$ from $\mathcal{M}$ and $S'$ from $\mathcal{M}'$. Then, following the definition in [ADMG18], the chamfer distance measures the average squared distance between each point in set $S$ to its nearest neighbor in $S'$ and vice versa:

$$\begin{aligned} d_C(\mathcal{M},\mathcal{M}') &= d_{samp,C}(S,S') \\ &= \frac{1}{|S|}\sum_{x\in S}\min_{y\in S'}\|x-y\|_2^2 + \frac{1}{|S'|}\sum_{y\in S'}\min_{x\in S}\|x-y\|_2^2 \end{aligned} \tag{2.4}$$

Another metric that relies on nearest neighbor calculation is the **Hausdorff distance** [HKR93]

$$d_H(\mathcal{M},\mathcal{M}') = \max\{\max_{x\in V} d(x,\mathcal{M}'), \max_{y\in V'} d(y,\mathcal{M})\}, \tag{2.5}$$

where $d(x,\mathcal{M}) = \min_{z \text{ sampled from } \mathcal{M}}\|x-z\|_2^2$. It outputs the maximum distance between any vertex from $\mathcal{M}$ and its nearest projection to $\mathcal{M}'$ or vice versa. We use it to evaluate the remeshing results, but not as a training loss since it only considers the distance between the two most distant points. Therefore, when calculating the gradient at all the other vertices of the mesh, it is zero.

The **Optimal Transport (OT) distance** between two sets is the solution of a transportation problem that attempts to transform one set to the other one in the shortest way [PC19]. Interpreted as the cost to move one pile of earth to another, it is referred to as "earth mover's distance" [RTG00]. Mathematically, it is the problem of comparing two probability distributions. Each surface mesh is interpreted as a probability distribution, and the distance measures the transportation cost between them [FCVP17, FSV+19]. The surface represented by $\mathcal{M}$ is sampled at the mesh vertices, and to every vertex $v_i$ a weight $\alpha_i \geq 0$ is assigned that corresponds to the area of its surrounding faces. Given this, we define a discrete, positive measure as a weighted sum of Dirac masses $\alpha = \sum_{i=1}^{|V|}\alpha_i\delta_{v_i}$ and $\alpha'$ for $\mathcal{M}'$ respectively. Taking $C(x,y) = \frac{1}{2}\|x-y\|^2$ as cost function for the feature space $\mathbb{R}^3$, the OT distance recalls the standard Monge-Kantorovitch transportation problem:

$$\begin{aligned} d_{OT}(\mathcal{M},\mathcal{M}') &= \mathrm{OT}(\alpha,\alpha') \\ &= \min_{\pi\in\mathbb{R}_{\geq 0}^{|V|\times|V'|}} \sum_{i,j}\pi_{i,j}C(v_i,v_j') \qquad \text{s.t. } (\pi\mathbf{1})_i = \alpha_i, (\pi^T\mathbf{1})_j = \alpha_j'. \end{aligned} \tag{2.6}$$

Note that the OT distance is, similar to the chamfer distance, also a nearest neighbor projection but with the global constraint of bijectivity, which leads to a more consistent matching [SFC+21].

### 2.2.2 Geodesic Distance on a Surface Mesh

Given two points $v, w$ on the surface mesh $\mathcal{M}$, the Euclidean distance calculates the distance between them in $\mathbb{R}^3$ by ignoring the surface.

The geodesic distance $d_{geod}(v,w)$, on the other hand, is calculated by passing the shortest path between $v$ and $w$ on the surface defined by the mesh $\mathcal{M}$ between the two points, see Figure 2.3 for a visualization. The path is constrained to lie on the surface and typically cuts across faces in the mesh. Therefore, calculating the shortest path

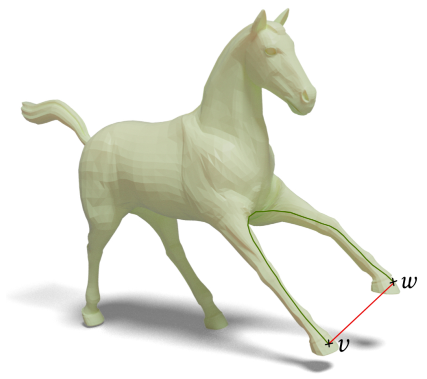


Figure 2.3: Two points $v, w$ with low Euclidean distance in red but higher geodesic distance in green on the surface on which they lie.

cannot be solved by applying the Dijkstra algorithm, which finds the shortest path only passing through vertices and edges of $\mathcal{M}$.

[MMP87] present the MMP algorithm for determining the exact shortest path between a source point $v$ and all destinations on an arbitrary polyhedral surface. The exact MMP algorithm has a worst-case time complexity of $\mathcal{O}(n^2 \log n)$. Nevertheless, [SSK$^+$05] demonstrate that, in practice, the exact algorithm runs in sub-quadratic time. An early approximation of the MMP algorithm applies Dijkstra and finds the shortest path through the vertices of a refined version of the initial surface mesh $\mathcal{M}$ [LMS97]. Among others, [KS98, SSK$^+$05, BK07] present more recent approximations of the exact algorithm with a lower runtime. My remeshing pipeline calculates geodesic distances between all pairs of vertices of a surface mesh $\mathcal{M}$ and applies the recent approach introduced in [CWW13]. They present a fast approach to compute the geodesic distances based on heat flow on the surface. The algorithm is rooted in the idea that a hot particle traveling from $v$ to $w$ in little time also had little time to deviate from the shortest possible path.

## 2.3 Laplacians for Surface Meshes

We begin by considering a 3D shape, represented as a triangular mesh $\mathcal{M}(V, \mathcal{F})$ comprising $n$ vertices $V \subset \mathbb{R}^3$ and additionally a feature function $f : V \to \mathbb{R}^F$ defined on the vertices. A Laplace operator $\mathcal{L}$ generalizes the second derivative of functions to other domains and, therefore, characterizes the domain smoothness. The Laplace-Beltrami operator is a special form for functions defined on surfaces or, more generally, manifolds. Nevertheless, all surface meshes are also graphs, so one can use the simpler graph Laplacian operators that only consider the connectivity between the vertices given by the faces $\mathcal{F}$. Section 2.3.1 introduces several discrete Laplacians for Graphs and Surfaces.

When decomposing the Laplace matrix, its orthonormal eigenvectors define a basis, and a function defined on $\mathcal{M}$ can be represented as a linear combination of these basis vectors. By only considering the most important basis vectors, $f$ can be represented

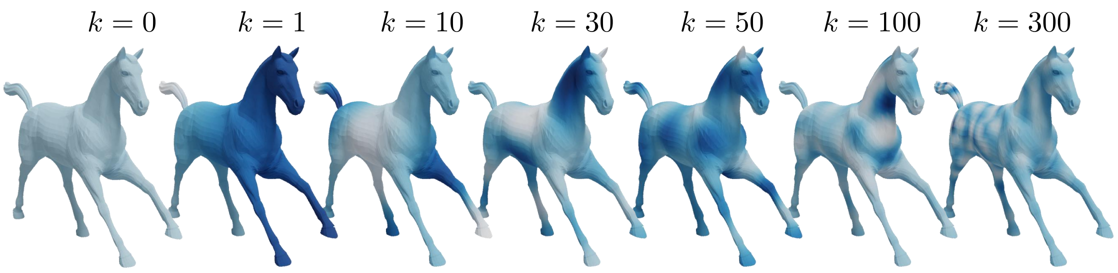


Figure 2.4: Visualization of selected $k$-th eigenfunctions of the discrete Laplace-Beltrami operator of a surface mesh representing a horse.

in lower dimension, and following processing tasks incorporating the function $f$ can be performed in lower dimensionality [VL08]. Section 2.3.2 details the decomposition of the Laplacian and how to define the low-dimensional basis.

### 2.3.1 Discrete Laplacians for Graphs and Surfaces

Graph Laplacians play a crucial role in analyzing graph signals in spectral graph theory. They act as a counterpart to the Laplace operator for functions in the Euclidean space. Essentially, the graph Laplacian quantifies the smoothness of a graph signal by considering the variations between the signal at a particular vertex and its neighboring vertices. In that way, this measure provides valuable information about the connectivity of the graph. The combinatorial and normalized graph Laplacians are two of the most widely utilized graph Laplacians and consider only the adjacency matrix.

**Definition 2** (Combinatorial Graph Laplacian)**.** *The combinatorial graph Laplacian $\mathcal{L}_C \in \mathbb{R}^{n \times n}$ is given as the difference of the diagonal degree matrix $D$ and the adjacency matrix $A$:*

$$\mathcal{L}_C := D - A$$

**Definition 3** (Normalized Graph Laplacian)**.** *The normalized graph Laplacian $\mathcal{L}_N \in \mathbb{R}^{n \times n}$ is defined as:*

$$\mathcal{L}_N := D^{-0.5} L_C D^{-0.5} = D^{-0.5}(D - A)D^{-0.5} = I - D^{-0.5} A D^{-0.5}$$

The discretization of the Laplace-Beltrami operator is a generalization of the Laplace operator to surface meshes. We apply the definition of the symmetrized cotangent Laplace-Beltrami operator $\mathcal{L}_B \in \mathbb{R}^{n \times n}$ from [SC20] for triangular meshes that can have non-manifold edges. Compared to the combinatorial graph Laplacians, it also considers vertex areas and cotan weights [MDSB03, PP93]. [SC20] provides a detailed definition of the Laplace-Beltrami Operator $\mathcal{L}_B$. Note that because its definition considers vertex areas and cotan weights, this Laplacian is not only dependent on the mesh connectivity but also changes under non-isometric deformation of the mesh.

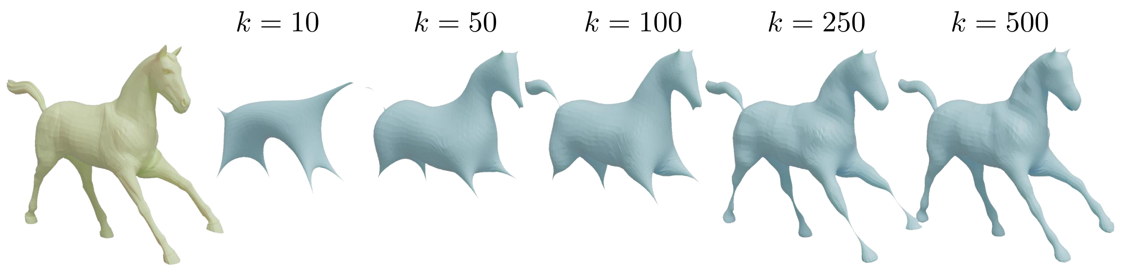


Figure 2.5: Reconstructed surface meshes from the representation of the vertex 3D coordinates $V$ in the $k$-dimensional spectral basis, which is obtained from the decomposition of the discrete Laplace-Beltrami operator. The original surface mesh is visualized in green on the left.

### 2.3.2 Decomposition of the Laplacians and Definition of the Spectral Basis

The introduced Laplace matrices $\mathcal{L}$ to surface mesh $\mathcal{M}$ are real, symmetric, and positive semidefinite. Therefore, when decomposing $\mathcal{L}$, its orthonormal eigenvectors $\mathbf{v}_i$ with corresponding real and nonnegative eigenvalues $\lambda_i$, $i = 1, \ldots, n$

$$\lambda_i \mathbf{v}_i = \mathcal{L} \mathbf{v}_i \tag{2.7}$$

define a basis, and a function $f$ defined on $\mathcal{M}$ can be represented as a linear combination of these basis vectors. This basis is called spectral basis because it possesses properties similar to the classical Fourier basis functions. If all eigenvalues are different, the decomposition of the Laplace operator and the representation of $f$ in the spectral basis is unique.

The eigenvectors $\mathbf{v}_i$, which in the interpretation as a spectral basis resemble eigenfunctions, are ordered by ascending absolute value of the corresponding eigenvalues $\lambda_i$, which resemble the frequencies of the eigenfunction. In the spectral domain, the eigenfunctions that correspond to the lowest eigenvalues and, therefore, the lowest frequencies are considered the most relevant ones to describe a function $f$ because higher frequencies are related to higher noise. Eigenfunctions corresponding to lower eigenvalues, on the other hand, capture the features of the lowest frequency of functions defined on the surface. Therefore, these eigenfunctions resemble coarser patterns, see Figure 2.4. Hence, the projection matrix $\Phi \in \mathbb{R}^{n \times k}$ is defined using the first $k$ eigenvectors $\mathbf{v}_i$ as the columns of $\Phi$.

Now we can project $F$-dimensional vertex-wise features $\mathbf{F} \in \mathbb{R}^{n \times F}$ defined by function $f : V \to \mathbb{R}^F$ from the mesh representation to the spectral representation by multiplying

$$\mathbf{A} = \Phi^{\dagger} \mathbf{F}, \tag{2.8}$$

where $\bullet^{\dagger}$ refers to the (left) Moore-Penrose pseudo-inverse. The resulting spectral features $\mathbf{A}$ lie in $\mathbb{R}^{k \times F}$. Note how this projection disentangles vertex-wise features from the representation on a mesh defined by a list of faces because it is captured by the eigenvectors resembling functions on the surface.

When setting $\mathbf{F}$ to the vertex 3D coordinates $V$, we can evaluate the significance of the low-dimensional spectral representation $\mathbf{A} = \Phi^{\dagger}V$ by reconstructing it to the vertex-space via $\Phi\Phi^{\dagger}V$ and visualizing the reconstructed surface mesh. Figure 2.5 visualizes a reconstructed horse for different numbers $k$ of considered eigenvectors to define the spectral basis. Note how finer structures of the mesh are only reconstructed for higher numbers of eigenvectors to define the spectral basis.

## 2.4 Functional Maps

Point-to-point (p2p) maps and functional maps describe a mapping from one shape to another shape. They have various applications, such as shape matching, animation, and texture or feature transfer [BBK09]. A point-to-point map between two shapes puts points on the surface meshes in correspondence, while a functional map maps real-valued functions from one shape to another. Additionally, functional maps are generally represented in a spectral basis obtained from decomposing the Laplace-Beltrami operator, which allows for a compact representation when selecting only the most important basis vectors. This chapter introduces both p2p maps (section 2.4.1) and functional maps (section 2.4.2) between shapes. Given a diverse collection of shapes, every shape has a different spectral basis since the shape-wise Laplace-Beltrami operators depend on the deformation, shape, and discretization of the shapes. If functional maps connect these shapes to each other, we can define the canonical consistent latent basis (CCLB) in section 2.4.3. It uses the correspondence information provided by the functional maps to define the common latent basis CCLB. This common basis then allows the joint analysis of the entire shape collection.

### 2.4.1 Point-to-point Maps Between Shapes

Point-to-point (p2p) maps are mappings from the set of vertices of one shape to the set of vertices of another shape. They describe the correspondence or alignment between individual points on the shapes. Let us consider two shapes represented by triangular surface meshes $\mathcal{M}_1$ and $\mathcal{M}_2$, a source and target shape, respectively. Then, a vertex-wise point-to-point map $T_{12} : \mathcal{M}_1 \to \mathcal{M}_2$ is defined as the function that maps each vertex in $\mathcal{M}_1$ to a corresponding vertex in $\mathcal{M}_2$. The sparse matrix $\Pi_{12} \in \mathbb{R}^{n_2 \times n_1}$, which takes the value 1 if $T_{12}(i) = j$, and 0 otherwise, represents the p2p map $T_{12}$. Then, to transfer vertex-wise features from $\mathcal{M}_2$ to $\mathcal{M}_1$, we can multiply $\Pi_{12}^T$ from the left to the feature vector defined on the vertices of $\mathcal{M}_2$. Therefore, $\Pi_{12}^T$ is the pullback of the vertex-wise map $T_{12}$ to the functional space.

This work only considers vertex-wise p2p maps describing exact correspondences. But it is also possible to encode correspondence in soft maps that assign to each point on $\mathcal{M}_1$ a probability distribution representing likely matches on $\mathcal{M}_2$.

### 2.4.2 Functional Maps

The direct estimation of the p2p map $\Pi_{12}$ is computationally infeasible since the number of possible point correspondences grows quadratically with an increasing number of vertices in the shapes. To address this issue, [OBCS+12] propose the functional map paradigm and reduce the dimensionality of $\Pi_{12}^T$ by representing it in the spectral bases of rank $k$ defined by the first $k$ eigenvectors of the Laplacians $\mathcal{L}_1$ and $\mathcal{L}_2$ of surface meshes $\mathcal{M}_1$ and $\mathcal{M}_2$. They construct the functional map $C_{21} = \Phi_1^\dagger \Pi_{12}^T \Phi_2$, which maps functions defined in the spectral basis of $\mathcal{M}_2$ to the spectral basis of $\mathcal{M}_1$. The functional map has a small size of $(k \times k)$, with $k$ usually below 50 in this work.

The functional map can either be obtained from a vertex-wise map $\Pi_{12}$ or is optimized directly in the low-dimensional spectral basis. To this end, two $F$-dimensional feature functions $\mathbf{F}_1$ and $\mathbf{F}_2$ defined on $\mathcal{M}_1$ and $\mathcal{M}_2$ respectively ($\mathbf{F}_i \in \mathbb{R}^{n_i \times F}$) are represented in the reduced basis $\mathbf{A}_i = \Phi_i^\dagger \mathbf{F}_i$. Next, the solution to the following optimization problem

$$\arg\min_C \|C\mathbf{A}_2 - \mathbf{A}_1\|_F^2 \tag{2.9}$$

is the sought-after functional map $C_{21}$.

### 2.4.3 Canonical Consistent Latent Basis

Given a collection of related 3D shapes represented by the meshes $\mathcal{M}_1, \dots, \mathcal{M}_n$, and a set of functional maps between some shape pairs, we build a functional map network (FMN) on the collection as follows. We construct a graph $\mathcal{G} = (V_\mathcal{G}, \mathcal{E}_\mathcal{G})$, where the $i$-th vertex represents the functional space of $\mathcal{M}_i$, and the edge $(i, j)$ exists if the functional maps $C_{ij}$ and $C_{ji}$ are given, in which case, the graph is symmetric. We assume that the graph $\mathcal{G}$ is connected, which means a path exists between any two shapes in the collection.

With this construction in hand, we can translate functions between any mesh representations $\mathcal{M}_i$ and $\mathcal{M}_j$ in the shape collection by multiplying the functional maps corresponding to the edges on the path through the graph $\mathcal{G}$. Nevertheless, there is no common basis where features from different shapes can be compared to each other. [WHG13] solve this by using a limit shape construction, which provides a consistent latent basis $Y_i$ for the shape features of the collection, such that $C_{ij}Y_i \approx Y_j, \forall i, j$. They obtain $Y_i$ by solving the optimization problem

$$\min_Y \|C_{ij}Y_i - Y_j\|_F \quad \text{s.t.} \quad \sum_i Y_i^T Y_i = I \tag{2.10}$$

These consistent latent basis vectors $Y_i \in \mathbb{R}^{k_1 \times k_1}$ ($k_1$ is the dimensionality of the functional maps) can be interpreted as functional maps from a latent shape to each mesh $\mathcal{M}_i$. If the shapes in the shape collection are in 1-to-1 vertex correspondence, [HAGO19] prove that $\sum_i Y_i^T \Lambda_i Y_i = \Lambda$ is a diagonal matrix, and its diagonal corresponds to the eigenvalues of the latent shape. Also, the eigenbasis $\Phi_0$ of the latent shape can be recovered as $\Phi_0 = \Phi_i Y_i$ for any $i$.

```
ALGORITHM 1: Constructing a Canonical Consistent Latent Basis (CCLB)
Input  : A shape collection of meshes $\mathcal{M}_1, \dots, \mathcal{M}_n$ and associated connected
         FMN $\mathcal{G}$ with functional maps of size $k_1 \times k_1$.
Output: A set of canonical consistent latent basis $\{\widetilde{Y}_i\}_{i=1}^n \subset \mathbb{R}^{k_1 \times k_2}$ with
         $k_1 \leq k_2$, the eigenbasis $\Phi_0$ and spectrum $\Lambda_0$ for the latent shape.
(1) Obtain the consistent latent basis $Y_i$ by optimizing (2.10).
(2) Compute the eigen-decomposition of $E = \sum_i Y_i^T \Lambda_i Y_i$ so that $EU = U\Lambda$ and let
    $\widetilde{Y}_i = Y_i U$.
(3) Let $\Phi_0 = \Phi_i \widetilde{Y}_i$ for an arbitrary $i$, and $\Lambda_0 = \Lambda$ from the previous step.
```

Nevertheless, [HAGO19] observe significant instabilities in the extracted latent basis $Y_i$ if the shapes are not in 1-to-1 correspondence, which is the case for the studied datasets in this thesis. They propose to further enhance the stability of this construction by introducing a normalization that forces $\sum_i Y_i^T \Lambda_i Y_i$ to be a diagonal matrix. Algorithm 1 summarizes the calculation of the more stable canonical consistent latent basis (CCLB) $\widetilde{Y}_i \in \mathbb{R}^{k_1 \times k_2}$.

[HAGO19] apply the CCLB for improving functional maps and for understanding variability within and across different shape classes. In comparison to selecting a template or reference shape for the definition of a common basis, they show that the CCLB enables an unbiased comparison of shapes using compact and meaningful representations in the common basis.

In section 5.3, I use the projection of shape features $\mathbf{F}_i$ to the latent shape in the encoder and, for the first time, the projection from the latent shape to a template shape for reconstruction in the decoder of the autoencoder network. [HAGO19] calculate characteristic shape differences following [ROA+13] that are linear operators acting on the function space of the limit shape to compare shapes in a common basis to each other. Nevertheless, they did not present any results on using the pseudo inverse of the matrices $\widetilde{Y}_i^\dagger$ as a projection matrix to the common basis. Similarly to (2.8), where vertex-wise features $\mathbf{F}_i$ are projected to the spectral representation expressed in the eigenbasis $\Phi_i$ of shape $i$ represented by $\mathcal{M}_i$, we can use $\widetilde{Y}_i^\dagger$ to project features to the CCLB

$$\mathbf{A}_i^{CCLB} = \widetilde{Y}_i^\dagger \Phi_i^\dagger \mathbf{F}_i. \tag{2.11}$$

Figure 2.6 evaluates the stability of the projection to the CCLB and back to the vertex representation by visualizing the reconstructed 3D coordinates $\Phi_i \widetilde{Y}_i \widetilde{Y}_i^\dagger \Phi_i^\dagger V$. For this experiment, I select the three animals in different poses from the *GALLOP* dataset (section 7.1.1), construct p2p maps between the three animals, build a fully connected FMN, and apply algorithm 1 to calculate the CCLB. There is no perfect p2p map between the animals since the trunk, tusks, and ears of the elephant do not correspond perfectly to horse and camel. This leads to unstable reconstructions for a higher dimension of

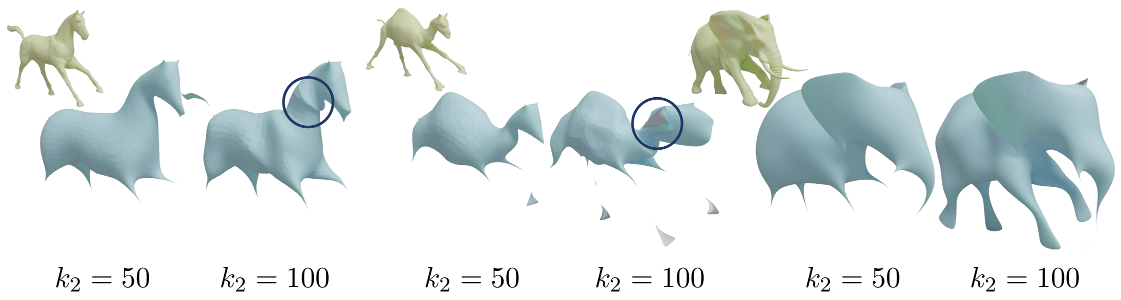


Figure 2.6: Reconstructed surface meshes from the *GALLOP* dataset from the representation of the vertex 3D coordinates $V$ in the $k_2$-dimensional CCLB, which is obtained following algorithm 1. We chose $k_1 = k_2 + 5$. The original surface meshes are visualized in green on the left, and the circles highlight reconstruction artifacts for $k_2 = 100$.

$k_2 = 100$ of the CCLB. For $k_2 = 50$, we observe a similar reconstruction quality as for the spectral reconstruction in Figure 2.5.

## 2.5 3D Shape Feature Descriptors

Methods for learning global 3D shape features build on local, for example, vertex-wise shape features that are summarized into one descriptor for the entire shape. Section 4.1 compares methods that learn vertex-wise features. This section focuses on hand-crafted 3D shape feature descriptors. The vertex-wise descriptors are used in many higher-level tasks, including finding shape correspondences [WBBP11, OBCS$^+$12] and segmentation algorithms based on clustering [SOCG10] or neural networks [QYSG17, SACO22]. Also, the additional experiments on 3D shape segmentation in section 9.1 use vertex-wise shape descriptors as input.

The choice of the local shape descriptor depends on the task and the desired invariance properties. For the segmentation of 3D shapes, one generally focuses on features that are invariant under rigid motion (global Euclidean transformations) and isometric deformations, which means inelastic bending transformations. Earlier descriptors used geodesic distances on the surfaces [HSKK01, EK03, GSCO07]. However, geodesic distances are sensitive to noise in the mesh topology because adding or removing small connections can lead to strong changes in the geodesic distances on the surface [SOG09, LB14].

More recent intrinsic 3D shape feature descriptors that are also invariant to isometric deformation are based on the diffusion distance proposed by [Laf04] also applied in the dimension reduction method Diffusion Maps [CL06]. These include the common heat kernel signature (hks) [SOG09] and wave kernel signature (wks) [ASC11]. The features use the spectral decomposition of the Laplace–Beltrami operator $\mathcal{L}_B$, see section 2.3.2, to compare and analyze geometric shapes. Therefore, they are isometry-invariant by construction.

The vertex-wise hks is based on heat diffusion over a surface. It applies the heat

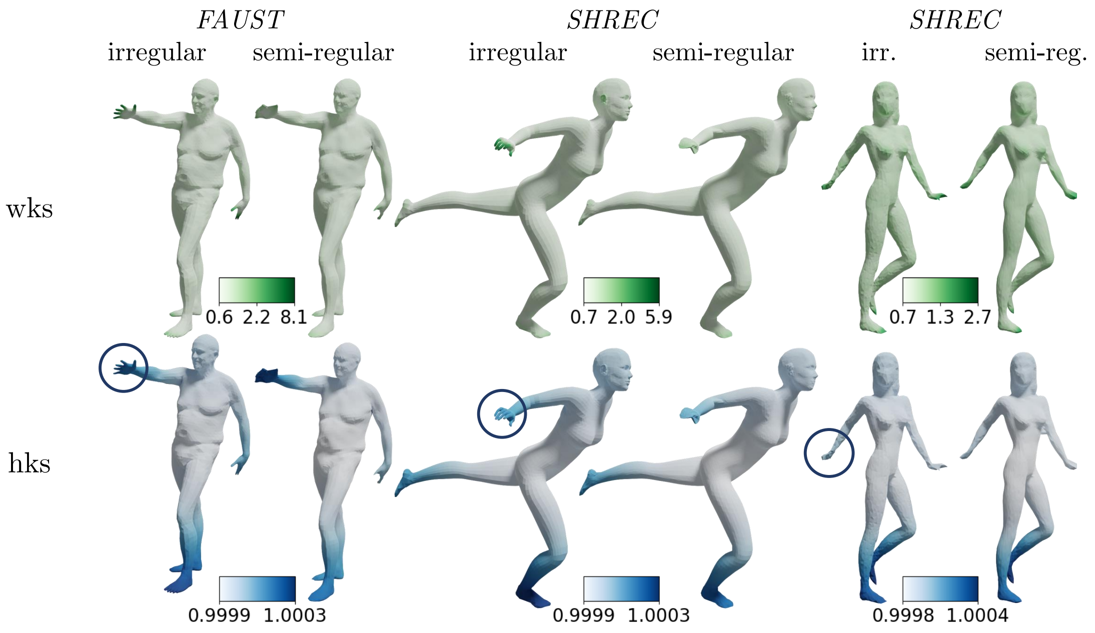


Figure 2.7: Three human meshes represented by irregular and semi-regular surface meshes with highlighted wks and hks descriptors on the surface. For two meshes representing the same underlying surface, the same color range is used to highlight the hks or wks feature descriptors. Note how the hks assigns different signature values to the hands of the different shapes. The vertex-wise wks is colored in log-scale.

kernel $k_t : \mathbb{R}^+ \times \mathcal{M} \times \mathcal{M} \to \mathbb{R}$, which can be interpreted as the amount of heat $k_t(x, y)$ transferred from point $x$, where a heat source is placed, to point $y$ (both on the surface $\mathcal{M}$) after time $t$. The heat kernel of a shape is computed using the eigendecomposition of the corresponding Laplace-Beltrami operator. For the analysis of different shapes, the hks descriptor restricts the heat kernel to the temporal domain by calculating $k_t(x, x)$ for several timesteps $t$ and a $x$ on the surface described by $\mathcal{M}$. The timesteps are selected from a time range that depends on the largest and smallest non-zero eigenvalue of the Laplace-Beltrami operator.

The wks follows a similar concept to the hks but replaces the heat equation with the Schrödinger wave equation. This change makes the wks more discriminative than the hks for shape matching [ASC11].

Figure 2.7 visualizes hks and wks signatures corresponding to the largest considered time $t$ on different human mesh representations. Since hks and wks are not invariant to scale, all analyzed shapes are scaled to unit area. Note how the hks is higher on the limbs than on the trunk, and both signatures are higher for hands and feet. Nevertheless, the signature values lie in different ranges for different meshes, because they are not invariant to changes in the triangle mesh structure [OBBG09, WBBP11, DSO20].

# 3 Machine Learning and Neural Networks

The verb *learn* is defined as "to acquire knowledge or skills as a result of study, experience, or teaching" in the Oxford English Dictionary [lea23]. Machine learning focuses on using data and algorithms to imitate how humans learn. To this end, we define a computer program that aims to learn the measurable performance of a clearly defined task by experience [Mit97]. Additionally, the user passes knowledge to the program by how it is defined and the kind of data presented to the program.

This work uses programs based on neural networks (NN) motivated by connected neurons in the human brain that are structured in one or more successive layers [GBC16]. A neural network with multiple layers between the input and output is called a deep neural network. Their high approximation power makes them a powerful tool for various learning tasks.

To successfully learn from experience, we must provide data to the network. We refer to the given input data by $x_i \in \mathbb{R}^n, i = 1, \ldots, d$, and the corresponding output data that should be predicted is $y_i \in \mathbb{R}^m, i = 1, \ldots, d$. The neural network then resembles a function

$$f_\theta : \mathbb{R}^n \to \mathbb{R}^m \quad \text{s.t.} \quad f(x_i) \approx y_i, \tag{3.1}$$

whose parameters $\theta$ are learned. To measure the performance of the neural network, a loss function $l : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ compares the predicted output $\hat{y}_i$ to the expected output $y_i$. The loss $l(y_i, \hat{y}_i)$ for data samples $x_i, i = 1, \ldots, d$ is then gradually minimized, generally by stochastic gradient descent.

This chapter summarizes related work regarding machine learning and neural networks, starting with an overview of different types of learning algorithms, loss functions, and how they are optimized in section 3.1. Section 3.2 introduces different layers of neural networks, while section 3.3 introduces autoencoders, the networks I apply to learn low-dimensional shape features. Finally, section 3.4 gives an overview of approaches to analyzing simulations using machine learning.

## 3.1 Learning Algorithms

This section gives an overview of different types of learning and summarizes the training of neural networks by gradient descent.

### 3.1.1 Supervised and Unsupervised Learning

The three main categories of learning algorithms are supervised, unsupervised, and reinforcement learning [Mur22]. This work focuses mainly on unsupervised approaches and conducts selected supervised experiments.

In supervised learning, the neural network is trained on labeled data, meaning that the correct output or "label" $y_i$ is provided for each input $x_i$ for $i = 1, \dots, d$. This allows the neural network to learn to create outputs on new, unseen data. A typical application is the classification of the input data into different categories [LJB+89]. In the case of image classification, the network predicts one class for the entire image that describes what the image is showing, or in the case of spam classifiers, a network predicts a spam or non-spam label for messages. Another application is the segmentation of regions of the input into different categories, where in the case of image segmentation, the network predicts a class for each pixel of the image.

In unsupervised learning, the neural network is not provided with labeled data, so there are no corresponding outputs. Instead, the goal is to find patterns or structures in the input data space. Dimension reduction algorithms are trained in an unsupervised way, and their performance is evaluated on the reconstruction of the input based on a latent low-dimensional feature representation also calculated by the network. This latent representation is expected to be meaningful if the reconstruction quality is high. Autoencoders are neural networks that can be used for dimension reduction and are explained in more detail in section 3.3. Other common applications are clustering algorithms where the input data should be clustered into significant subsets [Mur22].

### 3.1.2 Loss Functions

The loss function $l : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ evaluates how good the prediction $f_\theta(x) = \hat{y}$ is compared to the true label $y$, and it depends highly on the task and the data.

For regression-type problems, where $f$ outputs a real-valued quantity, the most common loss function is the mean squared error (MSE) [Mur22]. It is calculated as the average of the squared differences between the predicted $\hat{y}$ and actual values $y$

$$l_{MSE}(y, \hat{y}) = \frac{1}{m} \sum_{j=1}^{m} (y_j - \hat{y}_j)^2. \tag{3.2}$$

The training of autoencoders is a typical regression-type problem, where this work also applies the mean squared error. Nevertheless, the data handled in this work are surface meshes, which is why section 2.2.1 evaluates several loss functions based on the mean squared error that compare surface meshes to each other.

For classification-type problems, the function $f$ predicts one of $c$ class labels. Every sample $x$ is provided with a class label $y$ that is encoded in a one-hot vector $y \in [0, 1]^c$, which is only one for the single true class index. This is a special kind of regression, where we want the output $\hat{y}_j$ to resemble the probability to belong to class $j$, for $j = 1, \dots, c$. Here, the most common loss function is the cross-entropy [Mur22]. It calculates a score for the predicted $\hat{y}$ and true label $y$ that summarizes the average difference between the actual and predicted probability distributions for the $c$ classes

$$l_{CE}(y, \hat{y}) = -\sum_{j=1}^{c} y_j \log(\hat{y}_j). \tag{3.3}$$

Generally, before the loss calculation, a softmax activation function is applied to the network output $\hat{y}$, which ensures that the output resembles probabilities in range $[0,1]$ summing up to one

$$softmax(\hat{y})_j = \frac{e^{\hat{y}_j}}{\sum_{k=1}^{c} e^{\hat{y}_k}} \quad \text{for } j = 1, \dots, c. \tag{3.4}$$

### 3.1.3 Training using Stochastic Gradient Descent

Given the function $f_\theta$ resembling the neural network with trainable parameters $\theta$, the data $x_i \in \mathbb{R}^n, i = 1, \dots, d$, and the loss function $l$, we minimize the loss function with respect to the parameters $\theta$ of the neural network

$$\min_\theta \sum_{i=1}^{d} l(y_i, f_\theta(x_i)). \tag{3.5}$$

Neural networks are generally trained by methods based on first-order gradient descent that can scale to large datasets and models with lots of parameters [PPGS21, SCZZ20]. Consequently, successful training of neural networks requires a high amount of training data. The validation with unseen data is inevitable, and the datasets are separated into training and test data. The test data is then only presented to the network for evaluation of the training result [Mur22].

In practice, the network weights are optimized by stochastic gradient descent using randomly selected subsets (batches) of the data to calculate the gradient. This reduces the update time and, combined with adaptive learning rate and momentum methods, allows for a significant acceleration of the calculation [SCZZ20, KB15, SMDH13]. The calculation and tracking of the gradients are done automatically using, for example, PyTorch [PGM+19]. The gradients are calculated by iterating backward through the neural network, which is why it is also called backpropagation. Note that training using stochastic gradient descent requires the neural network function $f_\theta$ and the loss function $l$ to be differentiable with respect to the network parameters $\theta$.

## 3.2 Neural Networks and Convolutional Layers

Neural Networks (NN) are motivated by the structure and function of interconnected neurons in the human brain. They consist of layers, frequently considered to resemble neurons, that process and transmit information [Mur22]. In the case of feed-forward networks, the information flows in one direction, from the network input $x_i$ to its output $y_i$, without looping back. Each layer processes and transforms the input and passes it on to the next layer. Most of the layers are made up of a linear function $l_j : \mathbb{R}^{n_j} \to \mathbb{R}^{n_{j+1}}$ for each layer $j$, that combines the input from several neurons from the previous layer in a weighted sum. It is followed by an activation function $a_j : \mathbb{R} \to \mathbb{R}$, which generally introduces non-linearity to the network. The parameters of $l_j$ are optimized during the training.

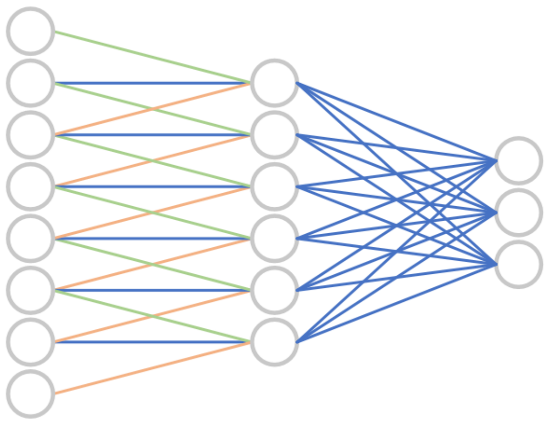


Figure 3.1: Feed-forward network. Every line resembles a connection between two nodes in subsequent layers (in grey). The first layer resembles a convolutional layer, and the second layer is fully connected.

We introduce different types of feed-forward layers for neural networks: fully connected layers and convolutional as well as pooling layers.

### 3.2.1 Fully-Connected Layers

In a fully connected layer, each node in layer $j + 1$ is connected to every node in the previous layer $j$, and each connection has its own weight. The output $\mathbf{o} \in \mathbb{R}^{n_{j+1}}$ of a fully connected layer is computed by taking the dot product of the input $\mathbf{x} \in \mathbb{R}^{n_j}$ with a weight matrix $\mathbf{W}_j \in \mathbb{R}^{n_{j+1} \times n_j}$, adding a bias term $\mathbf{b}_j \in \mathbb{R}^{n_{j+1}}$, and applying the activation function $a_j$ element-wise.

$$\bar{\mathbf{o}} = l_j(\mathbf{x}) = \mathbf{W}_j \mathbf{x} + \mathbf{b}_j \tag{3.6}$$

$$\mathbf{o} = a_j(\bar{\mathbf{o}}). \tag{3.7}$$

The weight matrix values $\mathbf{W}_j$ and the bias term $\mathbf{b}_j$ are optimized during training. A network made of multiple fully connected layers is also called a multilayer perceptron. The universal approximation property of deep neural networks states that this basic network architecture has a high approximation power, which refers to their ability to approximate any function, given enough data and capacity [LL20]. However, the dense weight matrix $\mathbf{W}_j$ uses $n_{j+1} \times n_j$ parameters per layer, which motivates lighter layer architectures, for example, convolutional layers.

### 3.2.2 Convolutional and Pooling Layers

Convolution for images was first introduced in [LJB+89] for digit recognition. Every neuron connects to one pixel of the rectangular image. Instead of fully connecting all pixels to each other, we only connect them to the locally close ones because they are

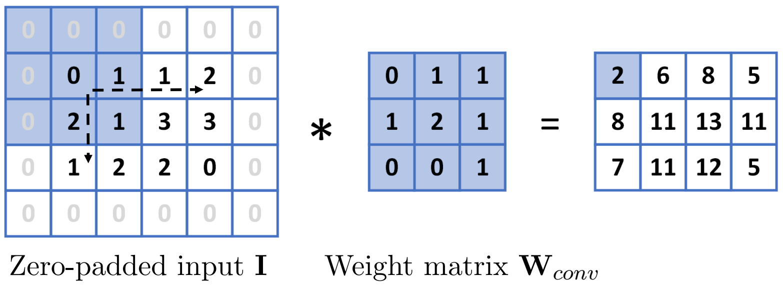


Figure 3.2: Application of a convolutional filter that slides vertically and horizontally along the dashed lines.

expected to be correlated. Additionally, the weights are shared for all positions to extract the same features for every neighborhood. The small weight matrix $(\mathbf{W}_{conv})_j \in \mathbb{R}^{k\times k}$ is referred to as filter or kernel, which slides over the image $\mathbf{I}$ and at every position, a dot product is computed with the local neighborhood. Figure 3.2 visualizes a convolutional layer when the stride is 1 and no positions are skipped when sliding the filter over the input. If the input is zero-padded, the image size remains the same. Again, a non-linear activation function $a_j$ is applied element-wise [Mur22]

$$\mathbf{o} = a_j(\mathbf{I} * (\mathbf{W}_{conv})_j). \tag{3.8}$$

Generally, a convolutional layer combines several convolutional filters whose pixel-wise outputs are concatenated and which calculate multi-dimensional feature vectors per pixel in the output image. The feature dimensions are also referred to as channels. For $c_{in}$ input and $c_{out}$ output feature channels, the resulting weight matrix is of size $c_{out} \times c_{in} \times k \times k$. Neural networks that are made of mainly convolutional layers are referred to as convolutional neural networks (CNN).

If we want to reduce the spatial size of the image, applying a pooling layer is common. Common types of pooling are max pooling, which selects the maximum value in a small window and replaces the entire window with that value, and mean pooling, which takes the average. Pooling layers are used to reduce the computational complexity of the network, make it more robust to small translations of the input, and reduce overfitting [GBC16].

Note that in the case of classical representation learning for 2D images with convolutional layers, one has a fixed-size grid along which the pixels align; in fact, all samples are in 1-to-1 correspondence. The convolutional filters with stride 1 calculate vertex-wise features, then pooling summarizes many vertex-wise features, reducing the number of pixels. This is done symmetrically for all the images, and the features from different samples are comparable because of the 1-to-1 correspondence.

## 3.3 Autoencoders

Autoencoders (AE) are a type of unsupervised learning algorithm that use neural networks to learn feature representations $h_i \in \mathbb{R}^{hr}$ of input data $x_i$ for $i = 1, \dots, n$

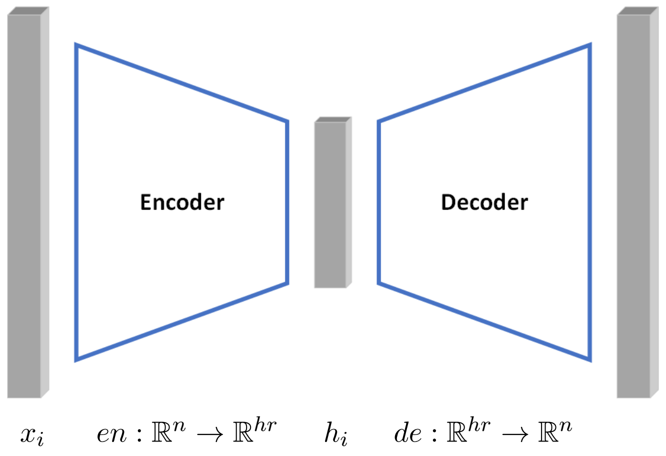

Encoder
Decoder
$x_i$
$en : \mathbb{R}^n \to \mathbb{R}^{hr}$
$h_i$
$de : \mathbb{R}^{hr} \to \mathbb{R}^n$

Figure 3.3: Autoencoder.

[Mur22, GBC16, BK88]. This is achieved by setting $y_i = x_i$. Then, the principle purpose is that the learned features of the input allow its recreation.

Autoencoders consist of two parts: an encoder network $en : \mathbb{R}^n \to \mathbb{R}^{hr}$ that calculates a low-dimensional representation $h_i$ of the input $x_i$, and a decoder network $de : \mathbb{R}^{hr} \to \mathbb{R}^n$ that recreates the input from the $hr$-dimensional representation, see Figure 3.3. The encoder and decoder are trained together to approximate the identity function

$$f(x_i) = de(en(x_i)) \approx x_i \text{ for } i = 1, \ldots, d. \tag{3.9}$$

In case $hr < n$, the feature representation is of a lower dimension than the input data. Therefore, we expect that the low-dimensional representation $h_i$, also called hidden representation, captures the most prominent features of the training data and allows distinction between the samples. This makes autoencoders an alternative to dimension reductions like the linear Principal Component Analysis (PCA) [Pea01], Isomap [TdSL00], or Diffusion Maps [CL06].

In practice, the user is generally not interested in the output of the decoder, the reconstructed input, but in the low dimensional feature representation, which captures the most relevant aspects that distinguish the input from the rest of the data.

As autoencoders learn an encoder function $en$, an advantage over standard dimension reduction methods is their improved generalization performance. The function $en$ is quickly evaluated and describes the manifold. Therefore, embeddings for new data points do not have to be obtained by interpolation.

## 3.3.1 Evaluation

The evaluation of autoencoders for any data type can be separated into different steps [GBC16]. Autoencoders are typically evaluated based on their ability to reconstruct the input data from the encoded representation, which is also the loss used during training. This reconstruction error measures the difference between the original data and its reconstruction and allows us to compare different autoencoder architectures.

The capacity of the autoencoder to learn a good representation of the data is evaluated by analyzing the encoded representations. Generally, they are visualized in a 2D or 3D space using, for example, the linear dimension reduction method PCA [Pea01] or the non-linear t-SNE [vdMH08]. In these low-dimensional visualizations, we expect to detect latent features of the dataset.

Since the goal of an autoencoder is to learn the underlying distribution of the input data, we also test to generate new samples from the latent space for the evaluation of the learned model. In the first step, new samples are created by interpolating between the latent representation of samples from the train or test data. The decoder is then evaluated, generally visually, on how realistic the generated samples are to understand if the model is overfitting to the training data.

The special and more challenging case of autoencoders for surface meshes and their evaluation is described in more detail in section 4.2 in the related work and in chapter 5 when introducing the proposed surface mesh autoencoders.

### 3.3.2 Autoencoders and Generative Models

Autoencoders learn a compact representation of the input data, which makes them relevant for many generative approaches since the low-dimensional embedding space is then used for data generation and manipulation. Autoencoders are an essential building block for variational autoencoders and generative adversarial networks [Mur23]. These two generative architectures are introduced shortly.

#### Variational Autoencoders

In a variational autoencoder (VAE) [KW14, RMW14], the encoder maps the input data to a probability distribution in the latent space, and the decoder maps a sample from this distribution back to the original data space. The goal of training is, similar to the original autoencoder, to minimize the difference between the original and decoded data. Additionally, the distribution of the samples in the latent space is encouraged to follow a fixed prior distribution (generally Gaussian), allowing the user to generate new samples from random noise after successful training. This is achieved by introducing an additional component to the optimizer, the Kullback–Leibler (KL) divergence, which measures the difference between the learned latent space and the prior distribution. In addition to using the Gaussian distribution as a prior distribution, there has been work on Gaussian mixture variational autoencoders [DMG$^+$16], who use this approach for clustering with a predefined number of Gaussian kernels that correspond to the clusters after successful training.

#### Generative Adversarial Networks

Generative adversarial networks (GAN) [GPAM$^+$20] aim to generate new data samples that are indistinguishable from real data. A generator produces new data and a discriminator classifying the data into real and generated samples are trained simultaneously. In this context, autoencoders can be used as the generator part of the GAN to produce the new data samples by sampling from the low-dimensional embedding space.

## 3.4 Machine Learning from Simulations

A simulation imitates a process or system that could exist in the real world, generally using the finite element method. A model and a configuration, for example, a car represented by several meshes and a crash scenario, are provided, and the simulation represents the evolution of this system over time. A simulation is generally cheaper and/or safer than conducting the actual experiment. Therefore, the goal of simulating a process is acquiring information about the process and understanding the behavior and relation between the input model and the simulation outcome before experimenting in the real world. To this end, several setups and model configurations are simulated, which creates a data pool of simulation results.

This paves the way to analyze the growing datasets using machine learning [vRMS$^+$20], for example, via clustering [BGIT$^+$13], anomaly detection [KDSG23], data-based mesh adaptation [BF21, YDP$^+$23], or surrogate modeling [LBD$^+$18]. If the dynamics of a simulation setup are understood, simulation results can be anticipated, and desired simulation results correlated to certain model parameters.

One research direction aims to learn the dynamics from a collection of simulations to predict the outcome of a simulation. [BRFF18] use convolutional networks emulating a fluid dynamics simulator to optimize the aerodynamic properties of a shape. [KGE$^+$21] learn low-dimensional features of a workpiece and, based on these features, predict the axial crushing response using recurrent neural networks. [HITG20], on the other hand, provide the first few timesteps as an input to predict a coarse approximation of the outcome of simulation results using LSTM modules. All these methods are generally tailored to a specific simulation setup and model.

Nevertheless, predicted simulation results using machine learning might not be physically correct. This issue led to physics-informed machine learning that enforces certain physical laws by incorporating them into the objective function for training or designing specialized network architectures [JKK20, KKL$^+$21]. Because domain knowledge is incorporated into the network, better results with less training data are expected.

Generally, simulations result in high-dimensional data sequences because, per timestep, the model state and local features are calculated. Therefore, data-based dimensional reduction techniques are common. A popular technique for data sequences is the dynamic mode decomposition (DMD) that was initially presented to analyze flow data [Sch10]. The dimension reduction into dynamic modes is not learned. The modes are, in fact, obtained by an eigen decomposition [HWM$^+$14, Sch22], see section 9.2.1. Recently, DMD has been combined with machine learning to learn a representation of the state space, for example, by using a fully connected network [TKY17] or autoencoders [LKB18, OR19, PD20]. Then, the DMD is applied to the learned features.

# 4 Learning on Surface Meshes

Neural Networks can learn significant local features for data and global features representing the data they handle. This is also the case for surface meshes, which this work explores. Many architectures excel at calculating vertex-wise features by applying convolution, see section 4.1, which finds application in mesh segmentation or correspondence learning.

Representation learning for surface meshes focuses on learning shape features, which represent the entire shape in one feature vector and often use learned vertex-wise features. These features facilitate the comparison of different shapes, speed up following analysis tasks on the shapes, and simplify shape editing. Section 4.2 introduces autoencoders calculating 3D surface mesh features. Many of the autoencoders presented in this section will serve as a baseline for the surface mesh autoencoding methods presented in this work.

Feature learning is also crucial for mesh generation, to which I provide an overview in section 4.3. Finally, section 4.4 introduces recent works in shape correspondence learning.

## 4.1 Convolutional Networks for Learning Vertex Features

When learning vertex-wise features, convolutional neural networks have proven to be good feature learners in many domains, including image and video analysis and natural language processing. This is because their combination of local weight sharing and considering neighborhoods of vertices allows the network to detect common 'local' features.

The convolutional filters applied to vertices of surface meshes can be distinguished into isotropic and anisotropic filters. Anisotropic filters have increased expressiveness because they are direction-dependent and take into account the direction on the surface from which we consider information. While this is a trivial task in the case of convolution on images since they are aligned along a global grid, surface meshes lack this global grid. Therefore, most earlier convolutional methods [BZSL14, MBBV15, DBV16, GSR$^+$17, KW16, MBM$^+$17] are isotropic because information from neighboring vertices is averaged and, therefore, rotation-invariant.

Since surface meshes can be viewed as graphs, earlier, graph-based convolutional methods were often applied to meshes. Generally, convolutional networks on graphs or surfaces can be separated into spectral and spatial ones [BBCV21, BBL$^+$17, WPC$^+$21].

Spatial convolutional methods for graphs aggregate features based on a node's spatial relations, for example, in its $r$-ring neighborhood. These methods allow generalization across different domains, and since they follow a common definition as message passing neural networks, their implementations are flexible and efficient [GSR$^+$17, WPC$^+$21].

The spatial methods use different ways to orient the local filters. The authors of [MBBV15, MBM$^+$17] calculate isotropic, rotation-invariant features by averaging the result of different anisotropic kernels that are sensitive to orientation. [LDCK18] introduces the spiral convolutions SpiralNet, which move around the neighborhood of the vertices in spirals. They enforce the network to learn rotation-invariant, anisotropic features by randomly selecting vertices for the spiral to start. In [BMR$^+$16, HJZS20], the kernels are aligned with the principal curvature direction, whereas [WGS$^+$18] sort neighboring vertices by similarity. [TPZ18], on the other hand, project neighborhoods onto tangent planes, and [HAGO19] align the filters along a 4-rotational symmetric field. Triangular polygonal meshes with highly regular meshing can be represented in hexagonal grids in almost all areas of the surface. Therefore, [HPCW18] apply hexagonal 2D-convolutional kernels that take advantage of the regular meshing and align the filters along the grid defined by regular connectivity. SubdivNet [HLG$^+$22] calculates face features for input meshes that are remeshed to semi-regular mesh connectivity. They define a spatial convolutional layer that takes advantage of the local regular meshing in their aggregation function with variable kernel size, stride, and dilation.

On the other hand, spectral approaches interpret information on the vertices as a signal propagating along the vertices. They exploit the connection of the graph Laplacian and the Fourier basis (see section 2.3), and vertex features are projected to the low-frequency Laplacian eigenvector basis, where filters and a non-linearity are applied [BZSL14]. Instead of explicitly computing Laplacian eigenvectors, the authors of [DBV16] use truncated Chebyshev polynomials, while [KW16] only use first-order Chebyshev polynomials. These spectral methods require fixed connectivity of the graph. If not, the adjacency matrix and, consequently, the Laplacian eigenvector basis changes. Note that these graph convolutional networks are also isotropic because the graph Laplacian is isotropic.

Furthermore, network architectures take into account the geometry of the surface when calculating vertex-wise features. They have been proposed to improve results on surfaces for classification, mesh segmentation, and shape correspondence. The anisotropic convolutional networks DiffusionNet [SACO22] and DeltaConv [WNEH22] are motivated by diffusion on the surfaces. DiffusionNet is discretization agnostic and learns the diffusion using Laplace–Beltrami operators of the local neighborhoods and directional features from gradients. DeltaConv, on the other hand, uses a more extensive set of operators and processes directional vector-valued features. Similarly, HodgeNet [SS21] constructs an operator based on a parameterized class of learnable operators. An alternative approach to building intrinsic rotation-equivariant convolutional networks for surface meshes is to use local parametrizations. [CWKW19, dHWCW21, WEH20, WFVW21] are independent of the choice of bases in the tangent spaces because they apply rotation- or gauge-equivariant kernels in the parameter domain. Generally, these methods track how the basis changes between the local parametrizations at the vertices, which can be costly.

The methods introduced in the previous paragraph, as well as the spectral approaches, are intrinsic convolutional methods because they operate directly on the tangent spaces around the vertices. Since the tangent spaces are two-dimensional, intrinsic convolutions can be more efficient, and additionally, they are robust to rigid- and non-rigid deformations [BMR$^+$16].

## 4.2 Shape Representation Learning via Mesh Autoencoder

While there are many approaches to calculating vertex-wise features, these architectures cannot be implemented directly into autoencoders. Surface mesh autoencoders calculate a relevant low-dimensional representation of the input shape and then reconstruct it only given this low-dimensional representation, see section 3.3. Nevertheless, the methods for calculating vertex-wise features for surface meshes are missing approaches to significantly combine these vertex features into one feature vector representing the entire mesh. For images, pooling operators successfully downsample the image resolution by combining vertex features from local neighborhoods. This approach cannot be directly transferred to meshes since the vertex neighborhoods do not have a fixed connectivity, see Table 2.1. Therefore, there is a need for different solutions and mesh-specific approaches.

The simplest approach is to average all vertex features to obtain a shape representation, for example, in PointNet [QSMG17] for point clouds. We refer to this averaging as global average pooling. A weighted local averaging that depends on the point density improves this slightly in [QYSG17] and, more recently, in [QLP$^+$22]. Although the global pooling loses all information about the mesh connectivity and locality of certain features in the learned shape representations, it is applied to 3D data because it is simple and avoids handling the geometry of the data. Note that the learned features are not localized. This makes the reconstructing by an autoencoder more challenging, as evident in the experiments in section 8.2.3.

Because of the disadvantages of global pooling of local features, other approaches calculate localized mesh features by downsampling the mesh to a target number of vertices or faces. In [LBBM18] and [RBSB18] (CoMA), some of the first mesh autoencoders have been introduced. The authors of CoMA, the Neural3DMM network [BBP$^+$19], SpiralNet++ [GCBZ19], [YLY$^+$20], MeshConv [ZWL$^+$20], and [ZCAK23] utilize mesh downsampling and mesh upsampling layers for pooling and unpooling. CoMA combines this with spectral convolutional layers, whereas Neural3DMM uses adapted anisotropic SpiralNet layers from [LDCK18]. [GCBZ19] slightly improves the autoencoder reconstruction results by introducing SprialNet++ layers. By manually choosing latent vertices maintained during downsampling into the embedding space, MeshConv [ZWL$^+$20] defines an autoencoder that allows interpolating in the latent space. They learn a global kernel weight basis from which they sample a convolutional kernel for every vertex, which accounts for irregular sampling and connectivity. [ZCAK23] again use SpiralNet layers but improve the upsampling in the unpooling operations. The spectral autoencoder (SAE) [LDLD22] projects all meshes of the same mesh connectivity to the possibly reduced eigenbasis of the constant graph Laplacian, where the autoencoder learns low-dimensional features. In contrast to the latter models, all pooling and unpooling operations take place in the spectral space.

All the above-mentioned mesh convolutional autoencoders work only for collections of meshes with the same connectivity because the pooling, convolutional layers, and/or graph Laplacian eigenbasis depend on the adjacency matrix. The MeshCNN architecture [HHF$^+$19] can be implemented as an encoder and decoder that handles different mesh topologies. Nevertheless, the pooling is feature-dependent, so the embeddings can be of different significance.

An earlier line of research maps surface meshes in $\mathbb{R}^3$ to $\mathbb{R}^2$ or some regular surface [BWS+18, MGA+17, SUHR17, SBR16]. In these representations, the definition of convolution and pooling can be defined regularly and more efficiently. Nevertheless, a successful projection can not be guaranteed, and the projection results are generally distorted and affected by artifacts [ZCAK23].

For surfaces represented as signed distance functions and in other implicit representations, [GYH+20] and [PFS+19] achieve good results in shape reconstruction and completion. Nevertheless, their generalization and scalability are often limited to a small set of deformations and require big training datasets as these approaches represent entire shapes using a single fixed-length vector.

Another parallel line of work is representation learning on point clouds [ADMG18, YFST18, QYW+19, ZBDT19, PWT+22, XHG+23]. Theoretically, these methods can handle surface meshes when disregarding the faces defining the surface mesh. However, these methods only reconstruct and generate point clouds, which is a different and more straightforward task than my work aims for because point sets are permutation invariant. This means that the reconstructed 3D coordinates do not need to fit a mesh topology and vertex arrangement; hence, the chamfer distance or OT loss measures the reconstruction loss appropriately [FSG17], which is not the case for surface meshes, see chapter 6.

## 4.3 Generative Models for Meshes

Generating new meshes or features for a given mesh connectivity is an important additional application of autoencoding because it requires compact representations of the shapes that lie on a smooth embedding manifold from which one can sample. To evaluate the smoothness of the learned embedding spaces using the proposed autoencoders, I also conduct generative experiments by interpolating in the embedding space and generating shapes in new positions. While generating deformed meshes from (random) sampling from the embedding space is not one of the main contributions of this work, this section briefly summarizes related work.

For generative models, low-dimensional features are combined linearly to generate shapes in positions that the user controls or the features are randomly sampled from an embedding space to create new samples.

[FKSC21, HRA+19, RBSB18] show mesh generative results by sampling from the mesh feature space of an autoencoder or variational autoencoder. These methods generally require a given mesh connectivity, which has to be the same for the entire dataset and predict the 3D positions of the vertices. For meshes of different connectivity, we expect the low-dimensional features to be arranged into clusters corresponding to the different mesh topologies. This impedes the application of standard variational autoencoders to most of the considered datasets because VAEs expect the set of embedded features to be normally distributed. Although there is research on Gaussian mixture VAEs, they are unstable when training and the number of clusters has to be predefined because it corresponds to the number of considered Gaussians [DMG+16].

Various methods enhance or replace the learned latent space with calculated and non-learned features. [YGT+23] generate shapes using a multiscale approach in combination with attention-based autoencoders. Their latent space for meshes of fixed mesh connectivity relies on a non-learned deformation representation [GLY+21]. The authors of [TGLX18] present a variational autoencoder for deforming 3D meshes that also handles a similar feature representation of the deforming meshes of fixed mesh connectivity. [BPAD23] presents a generative model that uses one template mesh for learned reconstruction and generation in combination with an encoder that calculates mesh-invariant features. They define a dissimilarity measure based on geometric measure theory to compare the generated mesh to the input mesh for optimization.

An alternative line of work uses generative adversarial networks (GANs) for mesh generation [BBP+19, CBZ+19, BHMK+18], which randomly sample from the embedding space. While GANs generally generate more diverse 3D shapes than autoencoders, they are more unstable, difficult to train, and require big training datasets.

Attention modules or transformers [VSP+17] have been applied in mesh generative experiments. [SWL+20] and [NGEB20] use an autoregressive generative model for 3D point clouds or meshes respectively and predict the 3D positions of vertices sequentially. They use transformers to capture spatial dependencies and [NGEB20] predict the faces using a second attention-based network. The recent work in [SAA+19], on the other hand, use transformers to create sequences of triangles. In an active parallel line of work, transformer models are used to generate human meshes from images [LWL21a, LWL21b, CYO22, Yos23] or from images and text [FLD+23], for example, for applications in virtual reality or human-computer interaction. Here, the transformer architectures model spatial interactions between vertices and joints from the 3D mesh representation.

## 4.4 Learning Shape Correspondences

The research area of correspondence learning or shape matching has the goal of predicting the point-to-point (p2p) map $T_{12}$ for two shapes represented by meshes $\mathcal{M}_1$ and $\mathcal{M}_2$, [BBL+17, GBS+16, GWH+21]. Functional Maps allow a representation of these p2p maps in a lower dimension. See section 2.4 for a definition of $T_{12}$ as well as the derived Functional Map $C_{21}$ derived from $T_{12}$. Instead of optimizing a p2p map directly, one can optimize the functional map [DSO20, MRR+19, NO17, OCB+17, SO20, HSA+23], which is more feasible since the size of the p2p map is quadratic in the number of vertices and the functional map is of reduced size $k \times k$ with $k$ smaller than the number of vertices. Here, [DSO20] calculate vertex-wise features $\mathbf{F}_1$ and $\mathbf{F}_2$ for $\mathcal{M}_1$ and $\mathcal{M}_2$ using DiffusionNet [SACO22] and project them onto the corresponding reduced Laplacian basis $\mathbf{A}_i = \Phi_i^\dagger \mathbf{F}_i$. Then the functional map $C_{21}$ can be estimated by

$$\arg\min_{\mathbf{C}} \|\mathbf{C}\mathbf{A}_2 - \mathbf{A}_1\|_F^2 + \lambda \|\mathbf{C}\Delta_2 - \Delta_1 \mathbf{C}\|_F^2. \tag{4.1}$$

The second term is a differentiable regularization that promotes the isometry of the maps, as described in [OBCS+12]. To train the DiffusionNet, they define a loss function that captures the quality of the functional maps.

While the above works depend on ground truth p2p maps for this optimization, there are recent works that focus on unsupervised learning of p2p maps [AO22, ETLTC20, HLRK19, RSO19]. Since there are no ground truth correspondences, these works impose structural properties such as bijectivity and orthonormality on functional maps in the reduced spectral basis, penalize the geodesic distortion of the predicted maps, or combine intrinsic and extrinsic shape alignment. Additionally, [HAGO23] apply the refinement method ZoomOut [MRR$^{+}$19] to improve the quality of the maps. This method navigates between the spectral and spatial domains while progressively increasing the number of spectral basis functions.

# 5 Proposed Methods for Mesh Autoencoding

This chapter first expresses the reason for creating an autoencoder to process meshes with varying connectivities in section 5.1. This is a shift from traditional autoencoders, which mainly work with point clouds and meshes with fixed connectivities.

After the motivation, I introduce two novel approaches and the resulting model architectures for shape representation and generation, resolving the challenges motivated in the first section. Section 5.2 introduces the **convolutional semi-regular mesh autoencoder** (CoSMA) and followed by the **canonical consistent latent basis autoencoder** (CCLB-autoencoder) in section 5.3.

Notice the elephant in the lower right corner?[1]

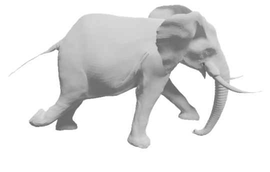
## 5.1 Motivation

Conventional autoencoders for 3D shape feature learning deal primarily with point clouds and surface meshes with fixed mesh connectivity, see section 4.2. Nevertheless, point clouds do not capture the shape surface, and many datasets are not limited to one fixed mesh topology. Therefore, I aim to develop autoencoders that handle meshes with different connectivities representing diverse shapes.

When learning shape features, the crucial step is to reduce the dimensionality of local features to one feature vector with a low dimension that describes the entire shape. In the case of 3D point clouds, autoencoders have been successfully applied by reducing point-wise features to a compact representation. Point clouds are unordered sets of 3D points and, therefore, invariant to the permutation of the points. Generally, point-cloud autoencoders take advantage of this and are also defined as permutation invariant, which allows effective processing. This means that the learned low-dimensional feature and the decoder output do not depend on the order or permutation of the points in the cloud. Generally, this property is enforced by applying a permutation invariant global pooling function on the entire set, like averaging or maximizing.

Triangular meshes are not permutation invariant in their vertex order because they encode detailed geometry and topology information of the surface, which is crucial for applications, for example, in computer graphics. The order of vertices in triangular meshes is essential for the definition of mesh connectivity, which is given by a face or edge list pointing to the indices of the connected vertices. Modifying the order of

[1] On this page starts the flipbook animating the galloping sequence of the elephant from the *GALLOP* dataset introduced in section 7.1.1. All shapes are reconstructed by the CCLB-autoencoder, and subsequent timesteps have been interpolated in the embedding space for a smoother animation.



vertices results in the disturbance and loss of connectivity information, diminishing the representational utility of the mesh.

Therefore, point cloud autoencoders cannot be applied straight-forward to surface meshes without disregarding the given connectivity information.

Existing approaches assume a strict 1-to-1 correspondence between meshes [LBBM18, RBSB18, BBP$^+$19, GCBZ19, ZWL$^+$20], see section 4.2. This allows for fixed mesh down and upsampling operations in the encoder and decoder that calculate hierarchical features. As a side effect, all the meshes share the same embedding space for the learned mesh features.

However, many meshes do not have fixed mesh topology. One could force them into the same connectivity via remeshing. Nevertheless, remeshing to achieve a 1-to-1 correspondence can introduce distortions and undermine the quality of the original mesh. Additionally, it might not be feasible. For example, meshes with different genus or boundaries cannot be remeshed to have the same mesh connectivity.

Another possibility to enforce a fixed mesh representation is to map the meshes to a common space, for example, a 2D space or a regular surface like a sphere or a torus [HSBH$^+$19, MGA$^+$17]. Projection to 2D allows for the application of 2D autoencoders. Noticeably, this can introduce high distortions. Also, there might not be a projection function back to the mesh representation, which limits the usability of this approach.

Therefore, this work aims to develop mesh autoencoder pipelines that handle arbitrary triangular surface meshes without needing 1-to-1 correspondence.

## 5.2 Convolutional Semi-regular Mesh Autoencoder (CoSMA)

As a first approach, I propose to remesh the given surface meshes to a semi-regular structure instead of enforcing fixed mesh connectivity for the entire mesh. It results in mesh patches that are of fixed and regular mesh connectivity. The proposed autoencoder pipeline takes advantage of these regularly meshed regional patches. For the first time, it can analyze arbitrary shapes represented by a semi-regular mesh, including different shape categories.

This section introduces the novel handling of semi-regular meshes for autoencoding It defines the Convolutional Semi-Regular Mesh Autoencoder (CoSMA) for patch-based shape representation learning for semi-regular meshes. The CoSMA makes use of tailored convolutional and pooling functions as well as padding of the patches, which are introduced in the following section. The two resulting network architectures using spatial and spectral convolution are introduced in section 5.2.2, followed by the introduction of the loss calculation.

Since the CoSMA handles meshes of semi-regular connectivity, I present a flexible remeshing pipeline for remeshing arbitrary irregular surface meshes to a semi-regular mesh representation that is presented in chapter 6. The entire pipeline consisting of remeshing and feature learning is visualized in Figure 5.1.

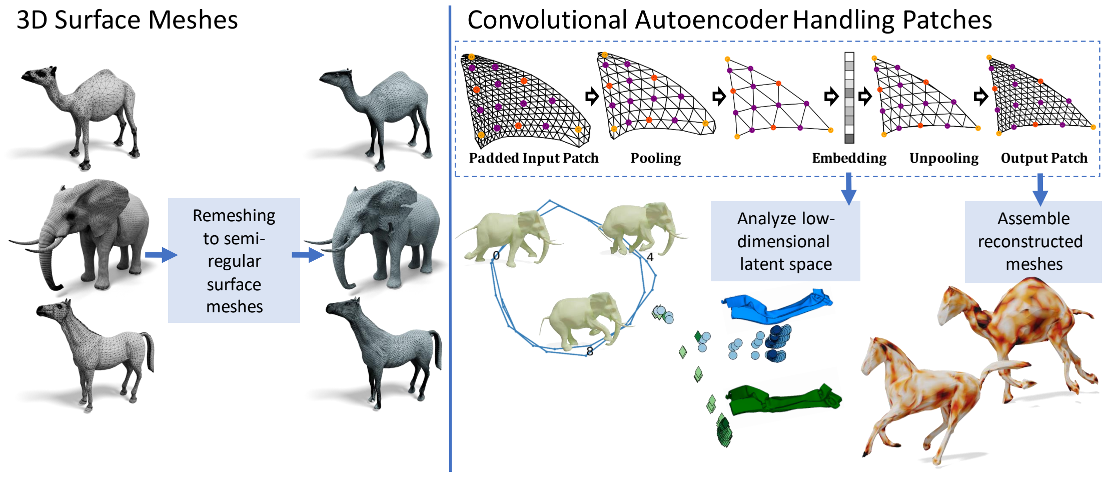

3D Surface Meshes
Remeshing to semi-regular surface meshes
Convolutional Autoencoder Handling Patches
Padded Input Patch
Pooling
Embedding
Unpooling
Output Patch
Analyze low-dimensional latent space
Assemble reconstructed meshes
0
4
8

Figure 5.1: CoSMA pipeline and method overview: The method consists of two stages. In the first stage (left), we calculate semi-regular mesh representations for the underlying surfaces. In the second stage (right), we train an autoencoder that handles the regularly meshed and padded patches of the surface meshes and calculates low-dimensional representations.

### 5.2.1 Method: Autoencoding Semi-Regular Meshes

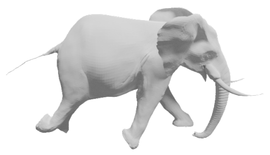
The semi-regular mesh structure, see section 2.1.1, results in regional mesh patches that are of fixed and regular mesh connectivity. I propose an autoencoder to handle these regular substructures of the mesh, leveraging that the local surface deformation follows similar patterns on the entire shape. Therefore, I apply patch-wise parameter sharing to learn translation-invariant localized features. Also, semi-regular meshes have, by definition, a multi-scale structure that provides mesh representations in different resolutions. This can be exploited for an efficient definition of pooling.

Because of the patch-wise handling, the pipeline can handle any shapes represented by semi-regular meshes that are not required to share the same global mesh connectivity. In fact, the shapes can be from different shape categories.

Note that remeshing the polygonal mesh only changes the representation of the objects. The considered surface embedded in $\mathbb{R}^3$ is the same but now represented by a different discrete approximation.

A similarly motivated line of research uses the patch approach for learning regional features that are then used for denoising and regularizing images [Pey08, Pey09, XLH+21]. The authors assume the patches lie on a patch manifold with a low-dimensional structure characterized by patch image features. This relates to my patch-wise mesh autoencoding approach, where I motivate the patch-wise handling by common local and regional deformation behavior.

Because all the regional patches share the same meshing and the convolutional neural networks learn local features, the patches are input separately into the network. For a semi-regular surface mesh with $j$ patches of regular connectivity, we refer to the vertex



**ALGORITHM 2:** Calculating the patch-wise padding of size $PS$.

```
Input  : Vertex coordinates $v_p$ of patch $p$ and $V$ of the entire shape, padding size $PS$.
Output: Vertex coordinates $\mathbf{Pad}_{PS}(v_p)$ of the patch $p$ with padding of size $PS$.
for $l = 0;\ l < PS;\ l = l + 1$ do
    for vertex $(v_p)_i$ on the patch boundary do
        if $(v_p)_i$ a vertex on the boundary of the entire shape then
            pad with boundary vertices $(v_p)_i$
        else
            if $N_1((v_p)_i) = 6$ then
                // vertex with regular degree
                use the coordinates of the vertices in $N_1((v_p)_i)$
            else if $N_1((v_p)_i) < 6$ then
                interpolate the values of the vertices in $N_1((v_p)_i)$
            else
                chose the closest vertices in $N_1((v_p)_i)$ in both cyclic rotations
            end
        end
    end
end
```

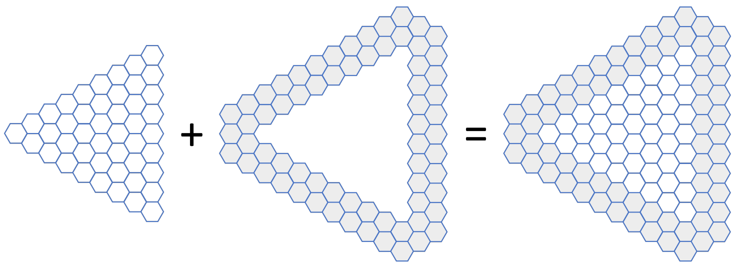


Figure 5.2: Visualization of the padding of a patch of refinement level $rl = 3$ using padding size $PS = 2$ .

coordinates of a patch as $v_p \in \mathbb{R}^{m\times 3}$ for $p = 1, \dots, j$. To consider the global context when handling patches, it is fed to the network via padding. Now, we define the padding, different convolutional functions, and a pooling function that are applied to the patches.

### Padding

The padding is crucial for the network to consider the regional patches in a larger context. Since the network handles the patches separately, I consider the features of the neighboring patches in a padding of size $PS$. Algorithm 2 outputs for every input patch $v_p$ the padded patch vertices $\mathbf{Pad}(v_p)$. Since the vertices where two patches meet are of regular degree, the padding is the regular $PS$-ring neighborhood of the vertices. At the corner vertices of possibly irregular degrees and the boundary of the entire shape, the algorithm approximates the padding. Figure 5.2 shows the application of the padding operation to a regularly meshed patch of refinement level $rl = 3$.

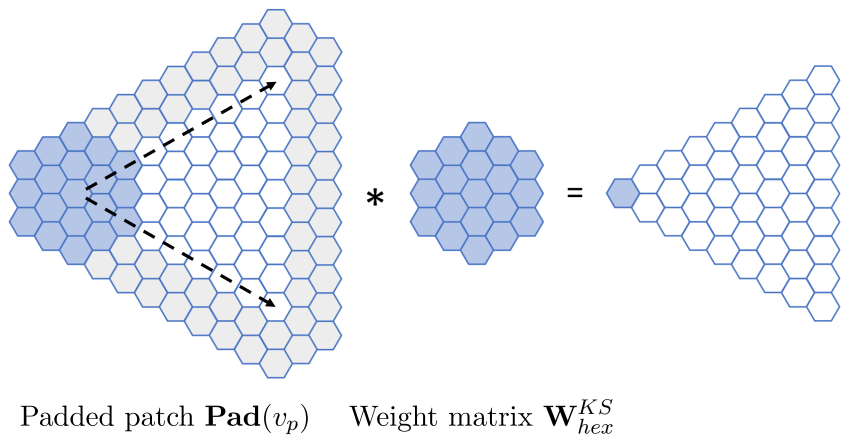

*
=
Padded patch $\mathbf{Pad}(v_p)$
Weight matrix $\mathbf{W}_{hex}^{KS}$

Figure 5.3: Application of a hexagonal convolutional filter of kernel size $KS = 2$ slided along the dashed lines to a padded input patch $\mathbf{Pad}(v_p)$.

### Spatial Convolution

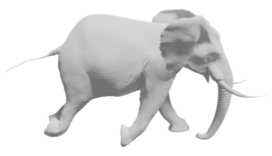
The regional patches are of the same regular structure. All vertices have exactly six neighbors; only the three corners can be irregular, but we project their neighborhood to a regular one with the padding. Additionally, the patches are intrinsically two-dimensional and represent a surface. Therefore, the application of a 2D-convolutional kernel is possible. Since the regional patches are represented in hexagonal grids, the application of hexagonal 2D-convolutional kernels has been shown to yield better results [HPCW18, SH19]. Similarly to [BRFF18], the consistent degree of the vertices results in better runtimes since similar calculations at these vertices can be moved to GPU.

On the local regular structure, the translation of the convolutional kernels is well-defined. Therefore, the kernels preserve the orientation of the neighborhood and are anisotropic. The padded patch-based approach assures gauge equivariance of the network. The authors of [CWKW19, dHWCW21] show how anisotropic kernels that preserve orientation significantly improve the expressivity of models.

This leads, in comparison to rectangular filters for images in Figure 3.2, to filters $\mathbf{W}_{hex}^{KS}$ of hexagonal shape, see Figure 5.3. Their kernel size $KS$ is given by the $KS$-ring neighborhood considered by a filter $\mathbf{W}_{hex}^{KS}$ with $1 + \sum_{k=1}^{KS} 6k$ trainable parameters. The hexagonal convolution on one-channel input features $\mathbf{x}$ on a hexagonal grid is defined as

$$\mathbf{o} = \text{HexConv}_{KS}(\mathbf{x}) = \mathbf{x} * \mathbf{W}_{hex}^{KS}. \tag{5.1}$$

I use the implementation of hexagonal convolution in Pytorch from [SH19]. Note that the network does not correct differences in the distances to neighbors or angles between neighbors. Nevertheless, the edge lengths of the semi-regular meshes are stable because of the edge length regularization during the remeshing, see section 6.2.2.



### Spectral Convolution

For the spectral convolution, I apply fast Chebyshev filters [DBV16], which [RBSB18] also apply to surface meshes, with the distinction that I perform them on the regional patches instead of the entire mesh. Spectral convolutions perform spectral decomposition using spectral filters and apply convolutions directly in frequency space. The Cheyshev convolution approximates the spectral filters by truncated Chebyshev polynomials, which avoids explicitly computing the Laplacian eigenvectors and, by this means, reduces the computational complexity.

The Chebyshev polynomials of order $k \in \mathbb{N}$ for input features $\mathbf{x}$ are defined as

$$T_1 = \mathbf{x} \tag{5.2}$$

$$T_2 = \hat{\mathcal{L}}\mathbf{x} \tag{5.3}$$

$$T_k = 2\hat{\mathcal{L}}T_{k-1} - T_{k-2}, \tag{5.4}$$

where $\hat{\mathcal{L}}$ denotes the scaled and normalized Laplacian $\frac{2\mathcal{L}_N}{\lambda_{max}} - 1$. Finally, the Chebyshev spectral graph convolutional function of filter size $K$ approximated by truncated Chebyshev polynomials is defined as

$$\mathbf{o} = \text{ChebConv}_K(\mathbf{x}) = \sum_{k=1}^{K} T_k(W_{Ch})_k, \tag{5.5}$$

with $W_{Ch} \in \mathbb{R}^K$ the trainable parameters for one-channel input $\mathbf{x}$ and output $\mathbf{o}$ vectors.

Note that the $k$th-order polynomials of the Laplacian are $k$-localized. Therefore, they encode information from the $k$-ring neighborhoods of the vertices. Since a large spectral filter size $K$ leads to fewer trainable parameters than a large spatial kernel size $KS$, $K$ is generally greater than $KS$. Also, in the experiments, the spectral convolutions consider bigger neighborhoods and, therefore, more general characteristics of the patch deformations.

### Pooling

The piecewise regular form of the semi-regular meshes has a multi-scale structure created by the iterative subdivision of the faces of the low-resolution mesh. I take advantage of this structure that all semi-regular meshes have in common and define an average pooling function that undoes the subdivision of one into four faces. This reduces the dimensions of the features and the number of network parameters, as pooling layers for 2D convolution do as well [GBC16, Mur22]. The vertices kept during this pooling operation take the average of their value and the values of the six neighboring vertices in the one-ring neighborhood that are discarded. The resulting pooling function $\mathbf{P}_{hex}$ is defined as

$$\mathbf{P}_{hex}(v_p)_{i^P} = \frac{1}{7}\Big((v_p)_i + \sum_{w \in N_1((v_p)_i)} w\Big), \tag{5.6}$$

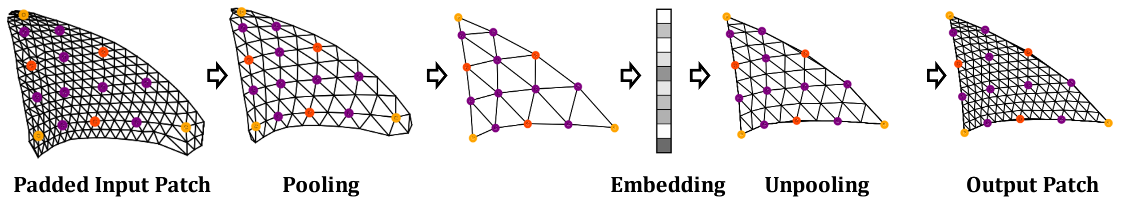


Figure 5.4: Resolution of a padded regularly meshed patch $\mathbf{Pad}(v_p)$ with refinement level $rl = 4$ inside the CoSMA. The encoder pools the patches twice by undoing the subdivision. In the decoder, the unpooling increases the resolution again by subdivision. The orange vertices are the vertices from the irregular base mesh. Red and purple vertices were created in the 1st and 2nd refinement steps.

with $v_p$ being the vertex coordinates of patch $p$ and the index of vertex $i$ is mapped to $i^P$ by reducing the refinement level.

To increase the resolution of the mesh patches in the decoder, the unpooling function $\mathbf{UP}_{hex}$ recreates the multi-scale structure of the semi-regular mesh. Every face is subdivided into four faces, and the newly created vertices are assigned the average value of neighboring vertices from the lower-resolution mesh patch. This corresponds to one midpoint subdivision of the mesh, see Figure 2.2.

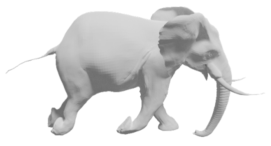
A similar pooling and unpooling is also applied by [HLG+22], where the information is saved on the faces. Figure 5.4 illustrates how the pooling and unpooling layers undo the subdivision of the regular patch or increase its resolution, respectively.

## 5.2.2 Network Architecture

I define a general structure for a mesh autoencoder using the convolution and pooling layers that are defined above. It is inspired by [RBSB18] but processes the padded regular patch coordinates $\mathbf{Pad}(v_p)$ with padding size $PS = 2$ of a semi-regular mesh. The autoencoder compresses every padded patch, which corresponds to one face of the low-resolution mesh, from $\mathbb{R}^{276\times 3}$ (dimensions for padded input at refinement level $rl_{in} = 4$) to an $hr_p$ dimensional latent vector and reconstructs the original padded patch from the latent vector.

The **encoder** $en_{CoSMA}$ consists of two blocks containing a convolutional layer followed by an average pooling layer. The output of the second encoding block is mapped to the latent space by a fully connected layer. The resulting $hr_p$-dimensional embedding is referred to as $h_p = en_{CoSMA}(\mathbf{Pad}(v_p))$.

The **decoder** $de_{CoSMA}$ mirrors the structure of the encoder by first applying a fully connected layer, which transforms the latent space vector back to a regular triangle representation with refinement level $rl_{in} - 2$. Afterward, two decoding blocks consisting of an unpooling layer followed by a convolutional layer transform the coarse triangle representation back to the original padded patch representation. Finally, another convolutional layer is applied without activation function to reconstruct the original patch



| | **Spatial CoSMA** | | | | **Spectral CoSMA** | | | |
|---|---|---|---|---|---|---|---|---|
| | Layer | Output Shape | $KS$ | Param. | Layer | Output Shape | $K$ | Param. |
| Encoder | Input | $(\bullet,\ 3, 267)$ | | 0 | Input | $(\bullet,\ 3, 267)$ | | 0 |
| | HexConv | $(\bullet, 2^4, 267)$ | 2 | 912 | ChebConv | $(\bullet, 2^4, 267)$ | 6 | 304 |
| | Pooling | $(\bullet, 2^4,\ 78)$ | | 0 | Pooling | $(\bullet, 2^4,\ 78)$ | | 0 |
| | HexConv | $(\bullet, 2^5,\ 78)$ | 1 | 3,584 | ChebConv | $(\bullet, 2^5,\ 78)$ | 6 | 3,104 |
| | Pooling | $(\bullet, 2^5,\ 15)$ | | 0 | Pooling | $(\bullet, 2^5,\ 15)$ | | 0 |
| | Fully Conn. | $(\bullet, 10)$ | | 8,010 | Fully Conn. | $(\bullet, 10)$ | | 4,810 |
| Decoder | Fully Conn. | $(\bullet, 2^5,\ 15)$ | | 8,800 | Fully Conn. | $(\bullet, 2^5,\ 15)$ | | 5,280 |
| | Unpooling | $(\bullet, 2^5,\ 78)$ | | 0 | Unpooling | $(\bullet, 2^5,\ 78)$ | | 0 |
| | HexConv | $(\bullet, 2^4,\ 78)$ | 1 | 3,584 | ChebConv | $(\bullet, 2^5,\ 78)$ | 6 | 6,176 |
| | Unpooling | $(\bullet, 2^4, 267)$ | | 0 | Unpooling | $(\bullet, 2^5, 267)$ | | 0 |
| | HexConv | $(\bullet, 2^4, 267)$ | 2 | 4,864 | ChebConv | $(\bullet, 2^4, 267)$ | 6 | 3,088 |
| | HexConv | $(\bullet,\ 3, 267)$ | 1 | 336 | ChebConv | $(\bullet,\ 3, 267)$ | 6 | 291 |
| | Total number of parameters | | | 30,090 | | | | 23,053 |

Table 5.1: Structure of the autoencoders spatial CoSMA and spectral CoSMA for refinement level $rl = 4$, and hidden representation of size $hr_p = 10$. The table also lists the number of trainable parameters for selected spatial convolution kernel size $KS$ and spectral convolution filter size $K$. The bullets $\bullet$ reference the corresponding batch size. The last dimension of the data is the number of vertices considered for each padded patch.

coordinates by reducing the number of features to three dimensions.

This general **CoSMA** architecture can handle all surface meshes remeshed into a semi-regular mesh representation of the same refinement level $rl_{in}$. This workflow is independent of the original irregular mesh connectivity and size, thanks to the remeshing and patch-wise handling. Note that this approach can handle non-manifold edges of the coarse base mesh because the patches, whose interiors by construction have only manifold edges, are fed separately.

Depending on the applied convolutional layers, we define two versions of the CoSMA. The **spectral CoSMA** uses the spectral convolutional layers with Chebyshev polynomials in combination with an exponential linear unit (ELU) as an activation function [CUH16]. The **spatial CoSMA**, on the other hand, uses the spatial convolutional layers followed by a biased ReLU activation function.

Table 5.1 defines and compares these two architectures. If not stated differently, the experiments use $K = 6$ Chebyshev polynomials for the spectral CoSMA, and for the spatial CoSMA, the kernel sizes $KS$ listed in the Table. Figure 5.4 illustrates the patch sizes inside the CoSMA autoencoder for $rl_{in} = 4$.

### 5.2.3 Surface-aware Loss Calculation

I input the 3D coordinates of the padded patch to the autoencoders, and both the spatial and spectral CoSMA output the reconstructed patch, including the padding. Nevertheless, for the reconstruction of the entire shape, the padding of the patch is not relevant. Therefore, during the model training, I optimize the mean square error (MSE) without considering the padding of the reconstructed patch. Let $v_p$ be the ground truth 3D coordinates of the patch $p$ and

$$\hat{v}_p = de_{CoSMA}(en_{CoSMA}(\mathbf{Pad}(v_p))) \tag{5.7}$$

their reconstructions. The patches have $m$ vertices without considering the padding. Then, the MSE between $v_p$ and $\hat{v}_p$ is

$$\text{MSE}(v_p, \hat{v}_p) = \frac{1}{m}\sum_{i=1}^{m}((v_p)_i - (\hat{v}_p)_i)^2. \tag{5.8}$$

Nevertheless, a patch-wise application of a mean squared error as the training loss does not keep track of multiple appearances of the vertices in the patch boundaries. Because the patches are considered separately, the boundary errors are weighted higher than in the interior of the patches. Therefore, it is not surface-aware and does not consider the patches as part of the entire mesh but separately. By weighting the vertex-wise error in the training loss with one divided by the number of appearances of the vertices in the different patches, I employ a surface-aware error for training.

Let us consider a semi-regular mesh with $n$ vertices that is made up of $j$ patches, which have $m$ vertices without considering the padding. For all vertices, $\mathcal{P}_i$ is the set of patches in which vertex $i$ appears. Then, I calculate the patch-wise surface-aware training loss between the ground truth 3D coordinates $v_p$ of the patch $p$ and their reconstructions $\hat{v}_p$ as follows:

$$\text{MSE}_{SA}(v_p, \hat{v}_p) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{|\mathcal{P}_i|}((v_p)_i - (\hat{v}_p)_i)^2. \tag{5.9}$$

When considering the MSE for the entire mesh, it holds

$$\begin{aligned}
\frac{1}{j}\sum_{p=1}^{j}\text{MSE}_{SA}(v_p, \hat{v}_p) &= \frac{1}{j}\sum_{p=1}^{j}\frac{1}{m}\sum_{i=1}^{m}\frac{1}{|\mathcal{P}_i|}((v_p)_i - (\hat{v}_p)_i)^2 && (5.10)\\
&= \frac{1}{jm}\sum_{p=1}^{j}\sum_{i=1}^{m}\frac{1}{|\mathcal{P}_i|}((v_p)_i - (\hat{v}_p)_i)^2 && (5.11)\\
&= \frac{1}{jm}\sum_{i=1}^{j}\sum_{p\in\mathcal{P}_i}\frac{1}{|\mathcal{P}_i|}((v_p)_i - (\hat{v}_p)_i)^2 && (5.12)
\end{aligned}$$

and the reconstructions of all vertices have the same weight, taking the average if there are multiple reconstructions. This improves the calculation of the reconstruction error and avoids artifacts and errors due to overemphasizing the patch boundaries.

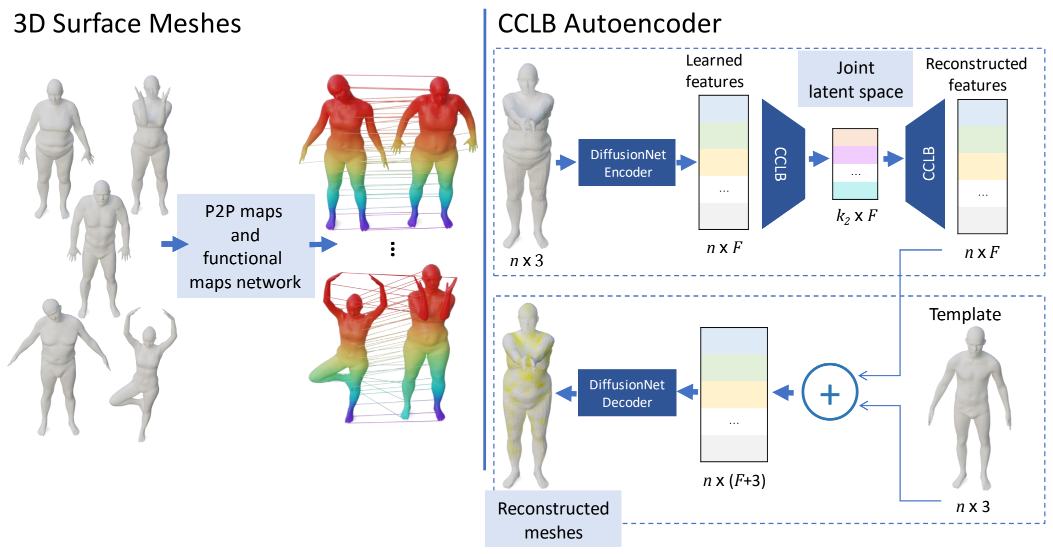


Figure 5.5: CCLB autoencoder pipeline: The method consists of two stages. In the first stage (left), we define a functional maps network using p2p maps between the 3D surface meshes. These p2p maps are used to calculate the canonical consistent latent basis (CCLB), which is used as the basis of the joint embedding space of the mesh autoencoder in stage 2 (right).

## 5.3 Mesh Autoencoder using a Canonical Consistent Latent Basis (CCLB-AE)

As a second approach, I propose to use shape correspondences to overcome different mesh connectivities when calculating shape features. For that objective, I introduce a novel spectral mesh pooling.

Note that all previously introduced approaches require the same mesh connectivity to situate the learned mesh features within a shared embedding space that allows for both comparative and manipulative operations on the shapes. Using mesh correspondences for autoencoding allows, for the first time, to calculate shape features of differently meshed shapes that lie in a joint embedding space.

As a first stage of my approach, the required mesh correspondences are calculated to construct point-to-point (p2p) maps between a collection of shapes. The second stage of the model is the autoencoder, where I make use of the novel spectral mesh pooling. This pipeline is visualized in Figure 5.5.

This section first introduces the proposed spectral mesh pooling, and then I define the network architecture and explain the applied p2p map extraction. Finally, I define the loss function and the regularizer that improves the performance when using unsupervised, imprecise learned p2p maps between the shapes.

### 5.3.1 Method: Spectral Mesh Pooling

I propose a spectral mesh pooling method to reduce the dimensionality of the meshes in the spectral domain to handle meshes of different connectivity and represent them in a joint low-dimensional embedding space.

In the case of classical representation learning for 2D images with convolutional networks, all image samples have a fixed size and are in 1-to-1 correspondence. The convolutional filters with stride 1 calculate vertex-wise features, then pooling summarizes many vertex-wise features, going from $n$ pixels to $k$. This is done uniformly for all the images in correspondence; hence, features from different samples are comparable to each other because of the 1-to-1 correspondence. Let us consider average pooling for images of size $n = 4 \times 4$ and the pooling size $k = 2 \times 2$. Then, these four matrices are a common basis for all the images with $2 \times 2$ pixels:

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}. \tag{5.13}$$

The projection vector from $n$ pixels towards one of low-dimensional basis has ones in the corresponding corner. For the first common basis, the resulting average pooling operation is

$$\frac{1}{4}\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \tag{5.14}$$

displayed in the shape of the image. Therefore, pooling in 2D can also be interpreted as a projection from $n$ dimensions to a set of common basis functions in $k$ dimensions. This projection reduces the dimensionality of the data while the pixel-wise feature dimensionality stays the same.

A similar pooling operation cannot be constructed for meshes with different mesh connectivities. Only p2p maps between the shapes are available, allowing a function to be projected from one shape to another.

To solve the pooling for meshes, I propose to adapt the CCLB method (initially developed for deformation detection) and introduce a novel intrinsic **spectral mesh pooling $\mathbf{P}_{Spec}$** $: \mathbb{R}^{n \times F} \to \mathbb{R}^{k_2 \times F}$. I project vertex-wise features $\mathbf{F}_i$ that are calculated for every surface mesh $\mathcal{M}_i$ separately to the common CCLB basis, reducing the dimension from the number of vertices to the size of the limit shape

$$\mathbf{P}_{Spec}(\mathbf{F}_i) = \widetilde{Y}_i^{\dagger} \Phi_i^{\dagger} \mathbf{F}_i. \tag{5.15}$$

At the same time, the local features are disentangled from the given surface mesh to global shape features. I calculate the limit shape basis CCLB as described in algorithm 1 in section 2.4.3. It has dimension $k_2$ and uses eigendecompositions of the Laplacians of size $k_1 \geq k_2$. It is the common basis for the low-dimensional embedding space.

For the spectral mesh unpooling, I project the features from the limit shape basis back to the vertex representation of a template shape $\mathcal{M}_t$. The resulting unpooling function $\mathbf{UP}_{Spec} : \mathbb{R}^{k_2 \times F} \to \mathbb{R}^{n_t \times F}$ is defined as

$$\mathbf{UP}_{Spec}(\mathbf{A}_i^{CCLB}) = \Phi_t \widetilde{Y}_t \mathbf{A}_i \tag{5.16}$$

with $\mathbf{A}_i^{CCLB}$ being the spectral mesh features and $n_t$ the number of vertices of the template shape used for reconstruction.

If the dimensionality of the CCLB is 1 ($k_1 = k_2 = 1$), the projection of the shape features into the CCLB corresponds to a global $\pm$ average pooling for all the shapes in the collection. Furthermore, the inverse of this operation duplicates the average feature to the mesh vertices, similar to upsampling in the 2D case. Also, the sign of the resulting global average pooling function from all shapes in the shape collection to the CCLB is the same, making the different low-dimensional representations comparable to each other. I formally state this observation in the following lemma.

**Lemma 1.** *If $k_1 = k_2 = 1$, there are only two possible solutions for the projection $\widetilde{Y}_i^\dagger \Phi_i^\dagger$ from the vertex-wise features $\mathbf{F}_i$ to the CCLB for all surface meshes $\mathcal{M}_i, i = 1, 2, \ldots$ and the projection from the CCLB to a template shape $\Phi_t \widetilde{Y}_t$. Either*

$$\boldsymbol{P}_{Spec}(\mathbf{F}_i) = \widetilde{Y}_i^\dagger \Phi_i^\dagger \mathbf{F}_i = \mathrm{mean}(\mathbf{F}_i) \ \forall i \ \textit{and} \ \Phi_t \widetilde{Y}_t = \mathbf{1}_{n_t} \tag{5.17}$$

*or*

$$\boldsymbol{P}_{Spec}(\mathbf{F}_i) = \widetilde{Y}_i^\dagger \Phi_i^\dagger \mathbf{F}_i = -\mathrm{mean}(\mathbf{F}_i) \ \forall i \ \textit{and} \ \Phi_t \widetilde{Y}_t = -\mathbf{1}_{n_t} \tag{5.18}$$

*with* mean : $\mathbb{R}^{n_i \times d} \to \mathbb{R}^d$ *the vertex-wise average function, $\mathbf{1}_{n_t}$ the column-vector with only ones in $\mathbb{R}^{n_t}$, and $n_t$ being the number of vertices of the template shape.*

*Proof.* At first, we proof that $\widetilde{Y}_i^\dagger \Phi_i^\dagger \mathbf{F}_i = \pm\mathrm{mean}(\mathbf{F}_i)$ for a fixed $i$. If $k_1 = k_2 = 1$, it holds

$$\Phi_i = \pm \mathbf{1}_{n_i} \tag{5.19}$$

being the eigenvector corresponding to the smallest eigenvalue $\Lambda_i = 0$ because the sum of all values in each row of the Laplacian $\mathcal{L}_i$ is 1. Therefore,

$$\Phi_i^\dagger \mathbf{F}_i = \frac{1}{n_i} \Phi_i^T \mathbf{F}_i = \pm\mathrm{mean}(\mathbf{F}_i). \tag{5.20}$$

The functional map

$$C_{ij} = \Phi_j^T \Phi_i \in \mathbb{R}^{1 \times 1} \tag{5.21}$$

is 1 or -1, projecting only constant functions from shape $j$ to shape $i$. It holds $C_{ij} = C_{ji}$. If $k_1 = 1$, the optimization problem to compute the Consistent Latent Basis

$$\min_Y \|C_{ij} Y_i \text{ - } Y_j\| \ \text{ s.t. } \sum_i Y_i^T Y_i = I \tag{5.22}$$

(see also equation (2.10)) has the solutions:

$$\begin{aligned} \text{if } C_{ij} = C_{ji} = 1 &\Rightarrow Y_i = Y_j \in \{-1, 1\} \\ \text{else } C_{ij} = C_{ji} = -1 &\Rightarrow Y_i = -Y_j \in \{-1, 1\}. \end{aligned} \tag{5.23}$$

Since $\Lambda_i = 0$, the matrix E in algorithm 1 in section 2.4.3 is 0. Therefore, the possible solutions for its eigenvector $U$ are -1 and 1. For the calculation of the CCLB, this leads to

$$\widetilde{Y}_i = Y_i U \in \{1, -1\}. \tag{5.24}$$

For the inverse it holds

$$\widetilde{Y}_i^{\dagger} = \widetilde{Y}_i. \tag{5.25}$$

From (5.20) and (5.25) follows

$$\widetilde{Y}_i^{\dagger} \Phi_i^{\dagger} \mathbf{F}_i = \pm \text{mean}(\mathbf{F}_i) \tag{5.26}$$

and all entries of the matrix have the same sign.

In a second step, we prove by contradiction that the non-zero entries of the matrix products $\widetilde{Y}_i^{\dagger} \Phi_i^{\dagger}$ have the same sign for all $i = 1, 2, \dots$ .
Assume that the sign of $\widetilde{Y}_i^{\dagger} \Phi_i^{\dagger}$ is different from the sign of $\widetilde{Y}_j^{\dagger} \Phi_j^{\dagger}$ for $i \neq j$. Without loss of generality, assume the sign of $\widetilde{Y}_i^{\dagger} \Phi_i^{\dagger}$ to be positive. Therefore, the sign of $\widetilde{Y}_i^{\dagger}$ is the same as the sign of $\Phi_i^{\dagger}$. Then, either $\widetilde{Y}_j^{\dagger}$ or $\Phi_j^{\dagger}$ has a different sign.
If $\widetilde{Y}_j^{\dagger} = -\widetilde{Y}_i^{\dagger}$, then $Y_j = -Y_i$ and therefore $C_{ij} = C_{ji} = -1$ because $Y_i$ and $Y_j$ solve (5.22). From (5.21) follows that $\Phi_j$ and $\Phi_i$ have different signs, which is a contradiction to $\Phi_i^{\dagger}$ having the same sign as $\Phi_j^{\dagger}$.
If in the other case $\Phi_i^{\dagger}$ has a different sign than $\Phi_j^{\dagger}$, $C_{ij} = C_{ji} = -1$ because of (5.21). It follows $Y_i = -Y_j$, which is a contradiction to $\widetilde{Y}_j^{\dagger}$ having the same sign as $\widetilde{Y}_i^{\dagger}$.

Finally, the entries of the matrix product $\Phi_t \widetilde{Y}_t$, which projects the features from the CCLB representation to the template shape, have the same sign as $\widetilde{Y}_i^{\dagger} \Phi_i^{\dagger}$.

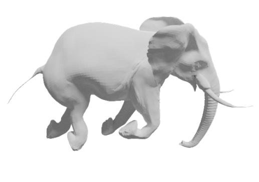
□

### 5.3.2 Network Architecture

Given a shape collection of meshes that can have different connectivity and p2p maps, we calculate functional maps between the shapes and then construct the functional map network, as well as the limit shape basis CCLB as described in 2.4.3 using only the shapes in the train set. The CCLB basis has dimension $k_2$ and uses eigendecompositions of the Laplacians of size $k_1 \geq k_2$. It will be the common basis for the low-dimensional embedding space. In addition, we chose a set of template meshes from the collection for the different types of meshes whose mesh connectivity will be used for the reconstructions.

The **encoder** $en_{CCLB} : \mathbb{R}^{n \times 3} \to \mathbb{R}^{k_2 \times F}$ takes as an input the vertex 3D coordinates of mesh $\mathcal{M}_i$. Four trainable DiffusionNet Blocks [SACO22] are applied to calculate $F$-dimensional vertex-wise features $\mathbf{F}_i \in \mathbb{R}^{n \times F}$. Then, I apply the proposed spectral mesh pooling and project the features to the CCLB $\mathbf{A}_i^{CCLB} = \mathbf{P}_{Spec}(\mathbf{F}_i)$. This low-dimensional representation of dimension $k_2 \cdot F$ is now independent of the mesh connectivity of $\mathcal{M}_i$ because it is represented in the common CCLB basis.

The **decoder** $de_{CCLB} : \mathbb{R}^{k_2 \times F} \to \mathbb{R}^{n_t \times 3}$ projects the features represented in the CCLB to the template shape $\mathcal{M}_t$ by applying the spectral mesh unpooling $\mathbf{UP}_{Spec}(\mathbf{A}_i^{CCLB})$.



| | **CCLB Mesh Autoencoder** | | |
|---|---|---|---|
| | Layer | Output Shape | Param. |
| Encoder | Input | $(n, 3)$ | 0 |
| | 4 DiffusionNet Blocks | $(n, F)$ | 466,472 |
| | Projection to CCLB | $(k_2, F)$ | 0 |
| Decoder | Projection to template shape | $(n, F)$ | 0 |
| | Append template 3D coordinates | $(n, F+3)$ | 0 |
| | 4 DiffusionNet Blocks | $(n, 3)$ | 466,819 |
| | Total number of parameters | | 933,291 |

Table 5.2: Structure of the CCLB mesh autoencoder calculating a hidden representation of size $hr = k_2 \cdot F$. The given parameter counts correspond to a model calculating $F = 40$ DiffusionNet features and are independent of the number of mesh vertices $n$ and embedding dimensions $k_2$.

At this point, I concatenate the vertex-wise 3D coordinates of the template shape to the projected features to provide more information for reconstructing the input shape. Finally, four trainable DiffusionNet Blocks reconstruct the 3D coordinates of the input shape on the template mesh vertices, see Table 5.2 for network details.

The proposed autoencoder applies the surface-based convolutional network DiffusionNet. Following the default segmentation configuration of DiffusionNet, 128 eigenvectors approximate the diffusion in the encoder and decoder.

For the test meshes, I estimate the projection matrix $\widetilde{Y}_i^\dagger$ by first projecting the features to the template shape $\mathcal{M}_t$ via the corresponding functional map and then applying the CCLB projection matrix of the template shape. This results in $\widetilde{Y}_t^\dagger C_{it} \Phi_i^\dagger \mathbf{F}_i$ for the spectral mesh pooling of test mesh $\mathcal{M}_i$.

### 5.3.3 Extraction of Point-to-point Maps

The calculation of the common latent basis using the CCLB requires point-to-point (p2p) maps or functional maps between enough shapes to construct a fully connected functional maps network $\mathcal{G}$. Also, the vertex-wise loss calculation takes advantage of given true p2p maps because it takes advantage of the shared mesh connectivity.

Many datasets come with true p2p maps between the shapes since the dataset or subsets of the dataset share the same mesh connectivity. Nevertheless, the CCLB-AE is not limited to datasets with given true p2p maps between the shapes.

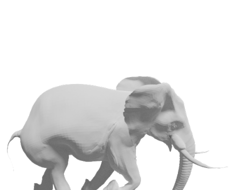
Obtaining supervised p2p maps is challenging and time-consuming since it requires significant labeling effort. However, there are different methods to calculate approximate p2p maps in an unsupervised way. We chose the method introduced in [HAGO23], see section 4.4. Since the unsupervised maps might be of lower quality, we apply an additional regularizing loss function to rectify the defaults in the maps.

### 5.3.4 Loss Calculation

The CCLB-autoencoder is fully differentiable, and we denote the 3D coordinates of the input mesh as $V$; the reconstruction is defined as $V_{rec} = de_{CCLB}(en_{CCLB}(V))$. It is trained using a combination of two losses.

**Point-to-point (p2p) loss**: Given a p2p map $\Pi$ (either ground truth or learned in an unsupervised way) between the template and the input mesh, the p2p loss is defined as $L_1 = \|\Pi V - V_{rec}\|_F^2$. However, in the case of unsupervised learned maps, this loss may provide inaccurate signals as the p2p map is often faulty and not entirely correct. An additional loss addresses this issue.

**Reconstruction loss**: Given the reconstructed coordinates $V_{rec}$, we construct the matrix $D^V$ such that $D_{i,j}^{V,rec} = \|V_{rec,i} - V_{rec,j}\|_F^2$. We similarly create the matrix $D^V$ for $\Pi V$. The reconstruction loss is formulated as $L_2 = \|D^V - D^{V,rec}\|_F^2$. This loss computes the cumulative reconstruction error, and each point receives reconstruction feedback from the other $n-1$ points. Thus, even if the p2p map is faulty in some places, the faulty points receive signals from the non-faulty ones [HAGO23]. Due to the possibly large size of the matrices $D^V$ and $D^{V,rec}$, the mesh vertices $V_{rec}$ and $\Pi V$ are resampled to 20,000 vertices during the loss computation, if the mesh has more than 20,000 vertices. As this loss is rotation invariant, it cannot be used alone. Therefore, when using unsupervised learned p2p-maps, the final loss combines the two losses: $L = L_1 + \lambda_{rec} L_2$.



# 6 Semi-Regular Remeshing

The convolutional semi-regular mesh autoencoders introduced in section 5.2 require the surface to have a semi-regular mesh representation. Section 2.1.2 introduced existing remeshing algorithms for semi-regular meshes and highlighted their limitations, see Figure 2.2. Also, the existing algorithms have strict conditions on the mesh characteristics that, for example, the mesh of the camel and horse cannot fulfill since they have non-manifold vertices and edges. Therefore, there is a need for a new pipeline for semi-regular remeshing.

The pipeline has two building blocks: At first, the irregular mesh $\mathcal{M}_{IR}$ is coarsened to a coarse representation $\mathcal{M}_{coarse}$; then I iteratively refine this coarse base mesh via subdivision of the faces and fit the resulting mesh $\mathcal{M}_{SR}$ to the original irregular mesh $\mathcal{M}_{IR}$. This chapter introduces the coarsening algorithm in section 6.1, followed by the subdivision and the mesh fitting in section 6.2.

Section 7.2 in the next chapter sets these steps together to a remeshing pipeline and evaluates the remeshing results after introducing the datasets.

## 6.1 Mesh Coarsening

To coarsen the surface mesh, I employ an adapted Garland-Heckbert algorithm for surface simplification using quadric error metrics [GH97]. It simplifies the mesh by collapsing edges until the target number of faces $k^*$ is reached, constantly contracting the pair of edges with the lowest cost, see Figure 6.1. The algorithm starts with a surface mesh $\mathcal{M}(V, \mathcal{F})$, which is coarsened iteratively $\mathcal{M}_{coarse,k}$ with $k \in \mathbb{N}$ describing the actual number of faces until $k \leq k^*$.

The cost per edge $c : \mathcal{E}_{coarse,k} \rightarrow \mathbb{R}$ measures the shape changes that are introduced by contracting the edge. For the edge $(v_1, v_2)$ in the initial mesh, the cost function adds up the square distances between the newly created vertex $\bar{v}$ to the faces adjacent to $v_1$

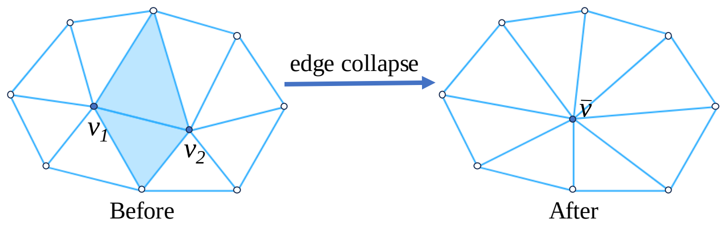
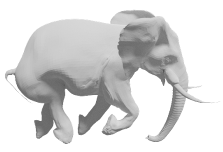


Figure 6.1: The edge $(v_1, v_2)$ is contracted into the newly created vertex $\bar{v}$. This degenerates the shaded triangles.



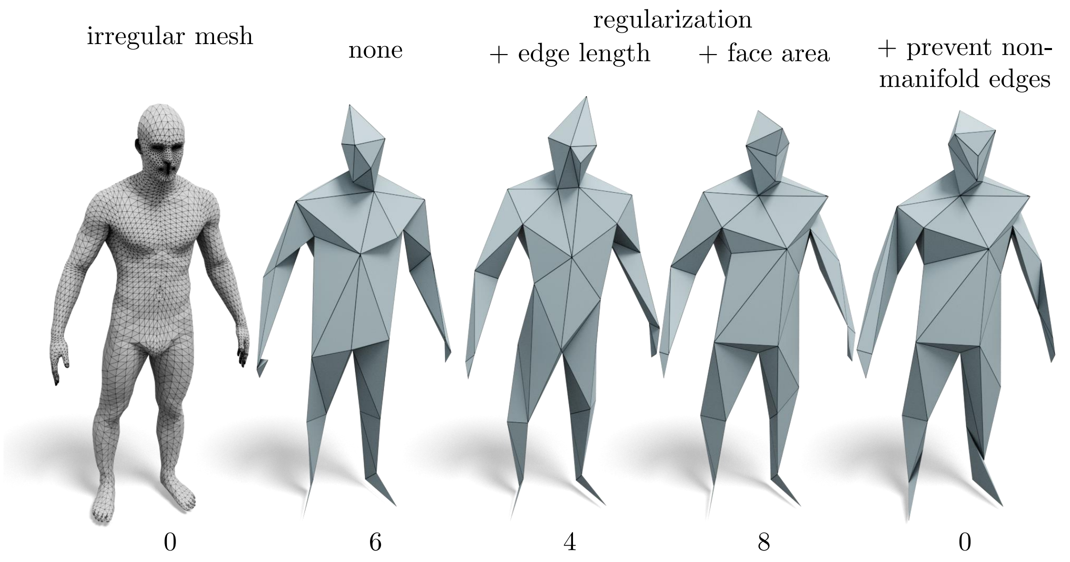


Figure 6.2: Mesh coarsening results from 13,776 to 100 faces and the number of non-manifold edges in the visualized mesh.

and $v_2$

$$c((v_1, v_2)) = \Delta(\bar{v}) = \sum_{p \in \mathbf{planes}(\bar{v})} \mathrm{dist}_{p2pl}(\bar{v}, p)^2, \tag{6.1}$$

where $\mathrm{dist}_{p2pl}$ is the distance between a point and a plane, and $\mathbf{planes}(\bar{v}) = \{\text{plane } p \text{ defined by } f \in \mathcal{F} | f \text{ adjacent to } v_1 \text{ or } v_2\}$. The coordinates of the newly created vertex $\bar{v}$ are chosen to minimize $\Delta(\bar{v})$.

The faces adjacent to $v_1$ or $v_2$ are now considered adjacent to $\bar{v}$. This union of adjacent faces is approximated to reduce the runtime [GH97].

### Regularization

If necessary, this cost is regularized. I regularize the edge lengths of the resulting coarse mesh [YLY+20] by adding the maximum length of the neighboring edges

$$r_{max,\mathcal{E}}((v, w)) = \max_{\substack{e \in \mathcal{E} \\ v \in e \text{ or } w \in e}} \|e\|_2 \tag{6.2}$$

to the edge-wise contraction cost $c((v, w))$. For regularizing the face areas, I add the maximum area of the neighboring faces

$$r_{max,\mathcal{F}}((v, w)) = \max_{\substack{f \in \mathcal{F} \\ v \in f \text{ or } w \in f}} \mathrm{area}(f). \tag{6.3}$$

Both regularizers are weighted by $\lambda_{\mathcal{E}} \geq 0$ and $\lambda_{\mathcal{F}} \geq 0$ respectively and added to the cost of contracting edge $(v, w)$. This increases the cost of an edge contraction that would make coarsely sampled mesh areas even coarser.

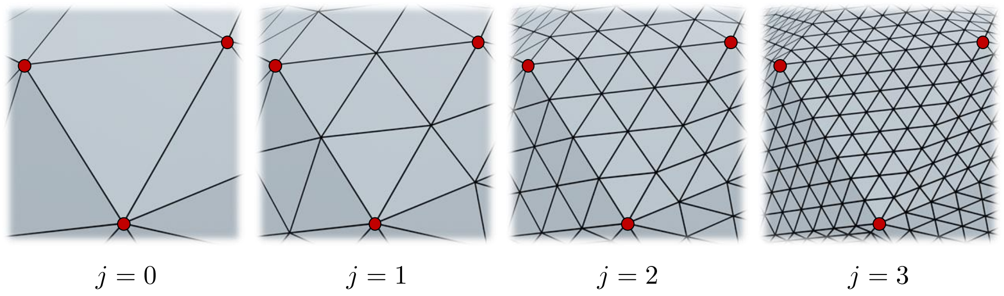


Figure 6.3: Iterative subdivision by splitting every face into four faces. $j$ indicates the refinement level.

Additionally, I prevent contractions of edges that lead to non-manifold edges and vertices in the mesh and disconnected components. This is only done towards the end of the coarsening procedure when the number of remaining faces $k$ is smaller than a selected $k_{nm}$ because otherwise, finer mesh structures of the irregular mesh cannot be coarsened, and the algorithm does not terminate. If $k < k_{nm}$ edges, whose contraction would create non-manifold structures in the mesh, are not considered, although their cost might be low. Figure 6.2 visualizes the regularization effect on the resulting coarse mesh.

## 6.2 Iterative Subdivision and Mesh Fitting

The subdivision is generally done iteratively, alternating the subdivision and the adaption of the vertex locations [LSS+98, PRS15].

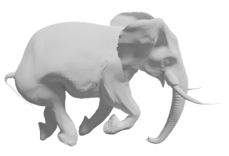
A 1-to-4 subdivision splits every face into four faces, which creates a new vertex for every existing edge. When splitting every face into four faces, all newly created vertices that do not lie on the mesh boundary have six neighbors, which creates the desired semi-regular mesh structure, see Figure 6.3. To calculate the initial 3D position of the newly created vertex, the positions of the two vertices that define the subdivided edge are averaged. This corresponds to one midpoint subdivision, see Figure 2.2.

The resulting 3D positions of the vertices of the semi-regular mesh of refinement level $j \in \mathbb{N}, j > 0$ are referred to as $V_{SR,j} \in \mathbb{R}^{n_j \times 3}$, with $n_j$ the number of vertices.

The semi-regular mesh $\mathcal{M}_{SR,j}$ has to be fit to the original irregular mesh $\mathcal{M}_{IR}$ to describe the surface well. The fitting is described by a deformation vector $\boldsymbol{\zeta}_j \in \mathbb{R}^{n_j}$ that contains 3D translations for the vertices $V_{SR,j}$ of the semi-regular mesh. I optimize the deformation vector $\boldsymbol{\zeta}_j$, such that the surface mesh $\mathcal{M}_{SR,j}$ with vertex positions $V_{SR,j} - \boldsymbol{\zeta}_j$ describes the underlying surface of $\mathcal{M}_{IR}$. The final vertex positions $V^*_{SR,j}$ of the semi-regular mesh at refinement level $j$ are then calculated by

$$V^*_{SR,j} = V_{SR,j} - \boldsymbol{\zeta}^*_j. \tag{6.4}$$

I apply different loss functions comparing $\mathcal{M}_{SR,j}$ to $\mathcal{M}_{IR}$ and additionally regularizers to $\mathcal{M}_{SR,j}$ in order to fit the created semi-regular mesh to the irregular one. Since I



optimize with respect to the translation vector $\boldsymbol{\zeta}_j$, the loss function and regularizers must be differentiable with respect to $\boldsymbol{\zeta}_j$.

The resulting objective function is composed of a selected loss function and regularizers and optimized using stochastic gradient descent. Therefore, it needs to be efficient to compute the forward and backpropagation of the loss functions and regularizers. Because the refinement level $j$ is fixed during the optimization of $\boldsymbol{\zeta}_j$, I omit the subscript $\bullet_j$.

### 6.2.1 Loss Functions

The loss functions measure the distance between the semi-regular mesh representation $\mathcal{M}_{SR}$ and the target irregular mesh representation $\mathcal{M}_{IR}$. In section 2.2.2, I have introduced several distance measures between surface meshes and point clouds that I slightly adapt to use as loss functions for remeshing.

#### Average Chamfer Distance

The average chamfer distance is calculated between sampled points $S_{IR}$ from the surfaces described by the original mesh $\mathcal{M}_{IR}$ and sampled points $S_{SR}$ from the iteratively deformed semi-regular mesh $\mathcal{M}_{SR}$ respectively, as explained in more detail in section 2.2.1:

$$d_C(\mathcal{M}_{IR}, \mathcal{M}_{SR}) = d_{samp,C}(S_{IR}, S_{SR}) \tag{6.5}$$
$$= \frac{1}{|S_{IR}|} \sum_{x \in S_{IR}} \min_{y \in S_{SR}} \|x - y\|_2^2 \tag{6.6}$$
$$+ \frac{1}{|S_{SR}|} \sum_{y \in S_{SR}} \min_{x \in S_{IR}} \|x - y\|_2^2. \tag{6.7}$$

#### Chamfer Distance weighted by Geodesic Distance

The chamfer loss handles point clouds sampled from the surface meshes. Therefore, their distance is minimized when two points on the surface meshes are nearest neighbors in the 3D space. They are brought closer together, disregarding that their geodesic distance on the remeshed surface might be high. This is why we want to prevent these cases, which generally occur when two convex regions of the surface mesh are close to each other, see Figure 6.4.

Note, how minimizing the chamfer distance $d_C(\mathcal{M}_{IR}, \mathcal{M}_{SR})$ fits the meshes in two directions: $\mathcal{M}_{IR}$ is projected to $\mathcal{M}_{SR}$ (6.6) and $\mathcal{M}_{SR}$ is projected to $\mathcal{M}_{IR}$ (6.7). Also, in convex regions of the shape surface, the semi-regular mesh that is built from a coarse approximation generally lies in the interior of the irregularly meshed shapes because the vertices of the coarse mesh lie on the surface. Therefore, the vertices created by subdivision lie in the interior. So, the described case appears when erroneously projecting vertices from $\mathcal{M}_{IR}$ to an unwanted area of $\mathcal{M}_{SR}$, see Figure 6.4.

Let $x_{IR}$ be a vertex on $\mathcal{M}_{IR}$ and its closest projection to $\mathcal{M}_{SR}$ and back to $\mathcal{M}_{IR}$ is $x_{IR,proj}$, which lies again on $\mathcal{M}_{IR}$. Now, I weigh the corresponding error term from

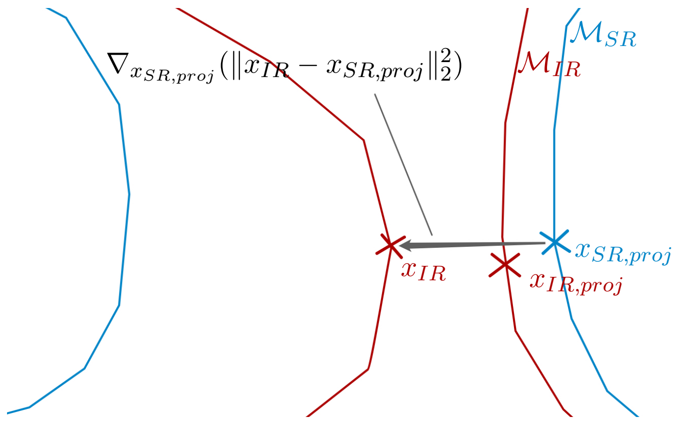


Figure 6.4: $x_{IR}$ is erroneously projected to $x_{SR,proj}$ on $\mathcal{M}_{SR}$ and the gradient $\nabla_{x_{SR,proj}}(\|x_{IR} - x_{SR,proj}\|_2^2)$ pulls improperly at $x_{SR,proj}$. Therefore, this projection is regularized depending on the geodesic distance $d_{geod}(x_{IR}, x_{IR,proj})$ on $\mathcal{M}_{IR}$.

equation 6.6 by a factor $< 1$, if the geodesic distance $d_{geod}(x_{IR}, x_{IR,proj})$ is higher than a threshold $t_{geod}$:

$$\begin{aligned} d_{C,geod}(\mathcal{M}_{IR}, \mathcal{M}_{SR}) = & \frac{1}{|S_{IR}|} \sum_{x \in S_{IR}} \min_{y \in S_{SR}} \frac{t_{geod}}{\max(t_{geod}, d_{geod}(x, x_{IR,proj}))} \|x - y\|_2^2 + \\ & \frac{1}{|S_{SR}|} \sum_{y \in S_{SR}} \min_{x \in S_{IR}} \|x - y\|_2^2. \end{aligned} \tag{6.8}$$

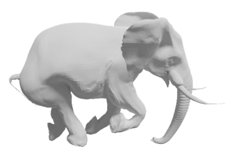
Only if $x_{IR}$ is projected to the intended area of the semi-regularly meshed surface, $t_{geod} > d_{geod}(x, x_{IR,proj})$ and the regularization factor $\frac{t_{geod}}{\max(t_{geod}, d_{geod}(x, x_{IR,proj}))}$ is equal to one.

### Boundary Loss

If the meshes have a boundary, we want the semi-regular mesh boundary $\partial\mathcal{M}_{SR}$ to follow the one of the irregular mesh $\partial\mathcal{M}_{IR}$. Nevertheless, since the coarse mesh lies in the interior of the irregular meshed shape, the random sampling of the chamfer distance leads to a semi-regular mesh representation that also lies in the interior of the irregular mesh.

Therefore, I define an additional loss that measures the distance between the two mesh boundaries. To this end, I use the chamfer distance between sampled points from the boundary of the irregular mesh $S_{\partial\mathcal{M}_{IR}}$ and the one of the semi-regular mesh $S_{\partial\mathcal{M}_{SR}}$:

$$\begin{aligned} d_B(\mathcal{M}_{IR}, \mathcal{M}_{SR}) &= d_C(S_{\partial\mathcal{M}_{IR}}, S_{\partial\mathcal{M}_{SR}}) \\ &= d_{samp,C}(S_{IR,B}, S_{SR,B}). \end{aligned} \tag{6.9}$$



### Optimal Transport (OT) Distance

Equation (2.6) in section 2.2.1 defines the Optimal Transport (OT) distance between two surface meshes $d_{OT}(\mathcal{M}_{IR}, \mathcal{M}_{SR})$ as the lowest transportation cost from one surface sampled at the vertices to the other one. To consider the vertex density on the surfaces, each vertex is assigned a weight corresponding to the area of the surrounding faces.

Since the calculation of the OT distance is computationally expensive, [FSV$^+$19] propose to approximate a regularized version of (2.6) using Debiased Sinkhorn divergences $d_{OT,\text{reg}}(\mathcal{M}_{IR}, \mathcal{M}_{SR})$. It is a computationally affordable approximation of the OT distance between $\mathcal{M}_{IR}$ and $\mathcal{M}_{SR}$. [FSV$^+$19] prove that it is a reliable loss function for machine learning applications because it is symmetric, smooth, and positive definite. Additionally, the authors provide a Pytorch implementation that allows its calculation and optimization on the GPU.

## 6.2.2 Regularizers

Note that both the chamfer distance and the OT distance do not fit the actual surfaces to each other but calculate distances between point clouds that are sampled from the mesh surfaces. This can lead to unwanted artifacts and non-smooth shapes, see Figure 7.10. Therefore, I add different shape regularizers to the loss functions to enforce smoothness.

### Edge Length

At first, I apply an edge length regularizer, similarly to the edge length regularizer applied during the mesh coarsening (6.2), to enforce similar lengths of the edges $\mathcal{E}$ of the semi-regular mesh $\mathcal{M}_{SR}$

$$r_{\mathcal{E}}(\mathcal{M}_{SR,j}) = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \|e\|_2^2 . \tag{6.10}$$

Since the filters of the CoSMA convolutional layers do not consider the edge lengths, they rely on stable edge lengths of the meshes for reasonable results and transfer learning experiments.

### Laplacian Smoothing

I also smooth the surface of the mesh $\mathcal{M}_{SR}$ by utilizing the Laplace Operator [DMSB99, NISA06]. I apply a discrete Laplace operator $\mathcal{L}$ to the vertex positions $V_{SR}$, measuring the smoothness of the mesh. Then, its norm is minimized

$$r_L(\mathcal{M}_{SR}) = \frac{1}{n} \sum_i \|(\mathcal{L}V)_i\|_2^2 . \tag{6.11}$$

I utilize the combinatorial graph Laplacian $\mathcal{L}_C$ (Definition 2), which compares a vertex value to all its neighbors, and divide by the number of contributing edges for point-wise evaluation $(\mathcal{L}_C V)_i$. It is evaluated as follows

$$(\mathcal{L}_C V)_i = \sum_{w \in N_1(v_i)} \frac{1}{|N_1(v_i)|}(w - v_i) \tag{6.12}$$

where $N_1(v_i)$ is the one-ring neighborhood of vertex $v_i$. Note how $(\mathcal{L}_C V)_i$ points to the centroid of the neighboring vertices of $v_i$. Alternatively, one could apply a cotangent Laplacian for the Laplacian smoothing, whose computation is more complex since it requires the calculation of cotangent weights. Then $(\mathcal{L}V)_i$ approximates the surface normal [DMSB99]. Nevertheless, I use the uniform combinatorial graph Laplacian $\mathcal{L}_C$ in the proposed remeshing pipeline because, at the same time, it regularizes the edge lengths of all neighboring edges by pointing to the centroid of the neighboring vertices.

### Normal Consistency

In addition, I enforce normal consistency for each pair of neighboring faces of $\mathcal{M}_{SR}$. This prevents neighboring faces from wrinkling and overlapping each other. The regularizer is defined as

$$r_N(\mathcal{M}_{SR,j}) = \frac{1}{|E_{interior}|} \sum_{e \in E_{interior}} 1 - \cos(n_{e,0}, n_{e,1}), \tag{6.13}$$

where $n_{e,0}$ and $n_{e,1}$ are the normals of the two neighboring faces of the interior edge $e$.

## 6.2.3 Optimization

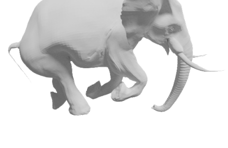
The regularization terms are weighted by the weights $\omega_E, \omega_L, \omega_N > 0$ and, if applicable, the boundary loss by $\omega_B > 0$. They are added to the chosen loss $d(\mathcal{M}_{IR}, \mathcal{M}_{SR})$. Since all the loss functions and regularizers are differentiable with respect to the deformation vector $\boldsymbol{\zeta}$, I apply stochastic gradient descent with momentum to approximate

$$\begin{aligned}
\boldsymbol{\zeta}^* = \arg\min_{\boldsymbol{\zeta}} \; & d(\mathcal{M}_{IR}, \mathcal{M}_{SR}) && && (6.14)\\
& + \omega_B \cdot d_B(\mathcal{M}_{IR}, \mathcal{M}_{SR}) && \text{boundary loss} && (6.15)\\
& + \omega_{\mathcal{E}} \cdot r_{\mathcal{E}}(\mathcal{M}_{SR}) && \text{edge length} && (6.16)\\
& + \omega_L \cdot r_L(\mathcal{M}_{SR}) && \text{Laplacian smoothing} && (6.17)\\
& + \omega_N \cdot r_N(\mathcal{M}_{SR}) && \text{normal consistency} && (6.18)
\end{aligned}$$

and fit the semi-regular mesh to the original irregular mesh. The chamfer loss, edge length regularization, Laplacian smoothing, and normal consistency are implemented in Pytorch3D [RRN+20], who motivated their application for fitting a sphere to a different mesh. The optimization of the losses is done using Pytorch [PGM+19].

## 6.2.4 Parametrization

If a dataset contains a shape in different positions or deformations while the mesh topology stays the same (for example, a shape deforming over time), I remesh a template mesh



that shares the same set of faces $\mathcal{F}$. The semi-regular remeshing result is parameterized and transferred to the meshes in different positions and deformations. After projecting the vertices of the semi-regular mesh to the closest face of the irregular template mesh $\mathcal{M}_{IR}$, I calculate the barycentric coordinates and obtain a parametrization. This parametrization of the remeshing result is then applied to the other deformed meshes, and semi-regular meshes discretize the complete sequence of the deforming shape. Note that this simplifies the overall workflow and allows visualization of a shape deformation sequence in a joint CoSMA embedding space. In principle, a semi-regular representation can be calculated for all the meshes, albeit sharing the same mesh connectivity, as done for the human body part segmentation in section 9.1.

Additionally, I use barycentric coordinates to calculate a parametrization of the irregular mesh based on the obtained semi-regular remeshing results. Then, I can project the reconstructed semi-regular meshes of the CoSMA architectures back to the irregular meshing. This allows the calculation of the reconstruction error in the irregular mesh representation, which is comparable to reconstructions of baseline methods.

# 7 Datasets and Remeshing Results

## 7.1 Datasets

I test the proposed surface mesh autoencoders on four categories of data containing various surface meshes from different domains and datasets.

### 7.1.1 GALLOP

The dataset contains triangular meshes representing a motion sequence with 48 timesteps from a galloping horse, elephant, and camel [SP04]. The authors of [SP04] transferred the surface deformation of a galloping horse to the camel and the elephant by using correspondences. Consequently, their deformation follows the same pattern, although elephants do not gallop [HFLK03]. Figure 7.1 visualizes example poses and, starting at page 35, a flipbook animates the galloping sequence of the elephant. While the galloping movement is similar, the meshes representing the surfaces of the three animals are different in connectivity and the number of vertices (horse: 8,431, camel: 21,887, elephant: 42,321). This is why mesh-dependent autoencoders have to be trained three times. I use the first 70% of the galloping sequences for training in the feature learning experiments. The architectures are tested on the remaining 30%.

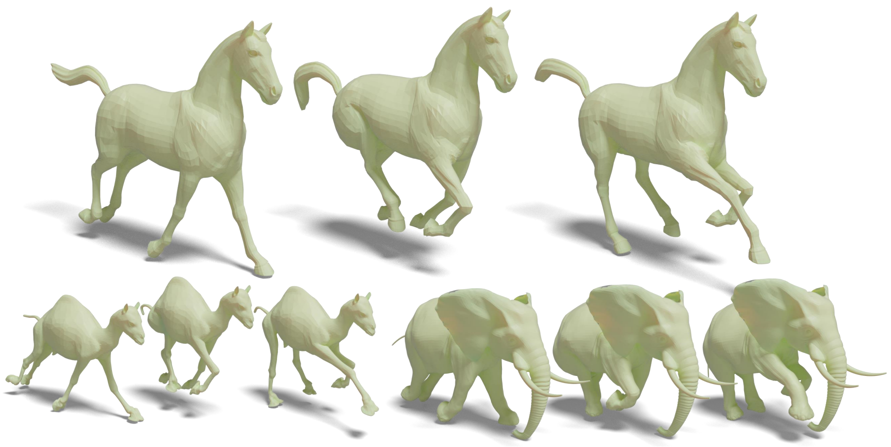
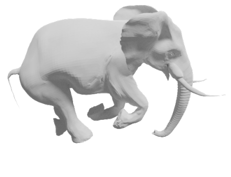


Figure 7.1: Animals from *GALLOP* dataset in three different poses from the galloping pattern.



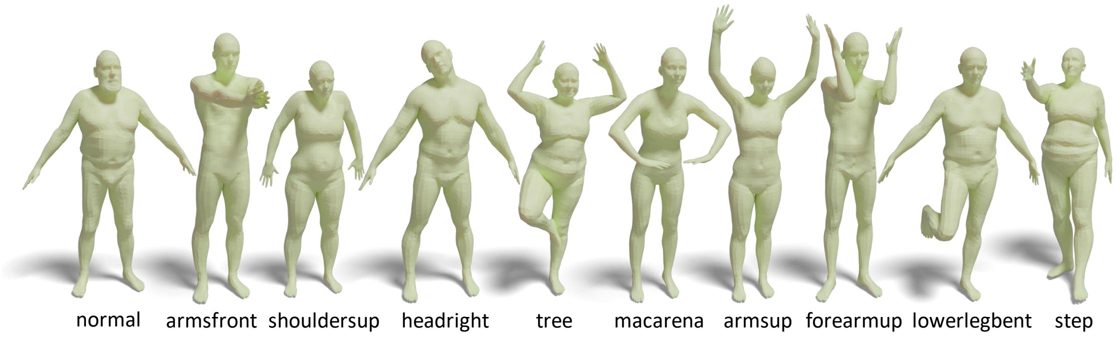


Figure 7.2: Ten different individuals in ten different poses from the *FAUST* dataset.

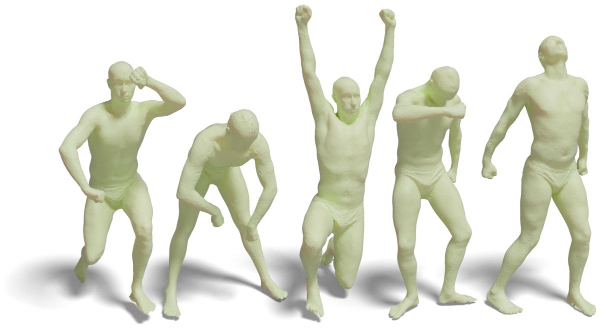


Figure 7.3: Individual from the *SCAPE* dataset in different poses.

### 7.1.2 Human Body Datasets

#### FAUST

The dataset contains 100 meshes [BRLB14], which are in 1-to-1 correspondence to each other, meaning that they share the same mesh connectivity. The dataset consists of ten different individuals in ten different poses, see Figure 7.2. To analyze the learned shape features later, I describe each position with a short name. The irregular surface meshes have 6,890 vertices, and I rotate the meshes around the vertical axis such that the individuals face in the same direction. Two different feature learning experiments are conducted: First, I consider the known poses of two unseen individuals in the testing set, referred to as "unknown individuals". Then, I consider two unknown poses of all bodies in the testing set, referred to as "unknown poses". In both cases, 20% of the data is included in the testing set.

#### SCAPE

The dataset contains 71 meshes [ASK$^+$05], which are in 1-to-1 correspondence. The dataset consists of one individual in different poses. Figure 7.3 visualizes a selection. The irregular surface meshes have 12,499 vertices, and they are also rotated around the

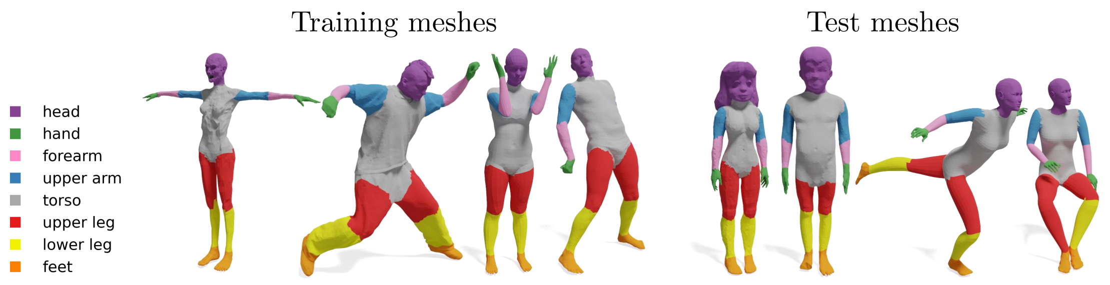


Figure 7.4: Human meshes with true segmentation labels for the human segmentation task.

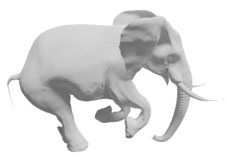
vertical axis such that the individuals face in the same direction. To demonstrate the reconstruction quality of the introduced methods on datasets that contain multiple mesh connectivities, I add the *SCAPE* dataset to the “unknown poses” experiment. For this, I use the same *FAUST* test shapes plus ten additional *SCAPE* shapes for testing.

### Human Body Part Segmentation Task

The human body dataset for segmentation contains 381 training shapes (*FAUST* [BRLB14], *SCAPE* [ASK+05], *ADOBE* [Ado23], and *MIT* [VBMP08]) and 18 test shapes from the *SHREC* dataset [GBP08]. The authors of [MGA+17] labeled the mesh faces into eight different segments corresponding to different body parts. The resulting learning problem predicting these face labels is a supervised one. Figure 7.4 visualizes meshes and their face-wise true segmentation labels from the different datasets.

## 7.1.3 TRUCK and YARIS

In a car crash simulation, the car components, which are generally represented by surface meshes, often deform in different patterns. A different surface mesh discretizes every component, while the same physical rules describe the local deformation. The *TRUCK* dataset contains 32 completed frontal crash simulations of 30 timesteps and six components of a Chevrolet C2500 pick-up truck. The components have between 696 and 1,736 vertices. 30% of the timesteps and two entire simulations are used only for testing. The *YARIS* dataset contains ten simulations of up to 26 timesteps and ten components of a detailed model of the Toyota Yaris (both models from NCAC [Nat]). It is only considered as a test set and the components have between 534 and 3,205 vertices. Figures 7.5 to 7.7 visualize the car models and the selected components. From simulation run to simulation run, model parameters are modified to achieve multiple design goals, for example, crash safety, weight, or performance. The car model often deforms in different patterns depending on the chosen model and simulation parameters, see Figure 7.6. Since the simulations nowadays contain detailed information for up to two hundred time steps and more than ten million nodes, their analysis is challenging and is generally



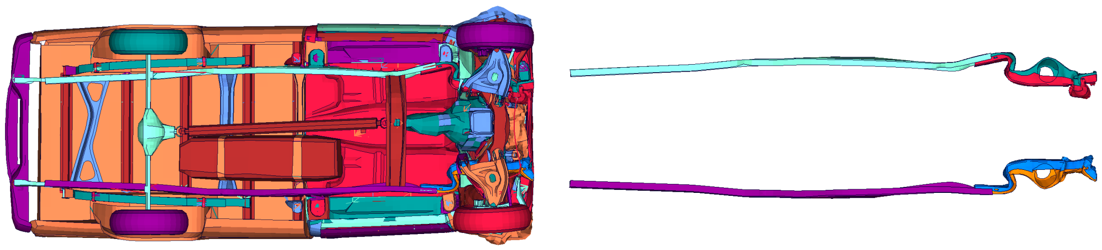


Figure 7.5: Bottom view of the *TRUCK* model after half of the simulation time. Left: the entire model, right: the six selected components.

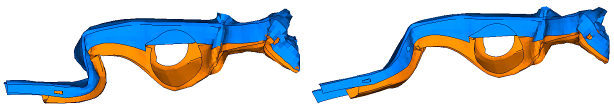


Figure 7.6: Two deformation patterns that manifest in the *TRUCK* front beams. Blue: Part 0; Orange: Part 1.

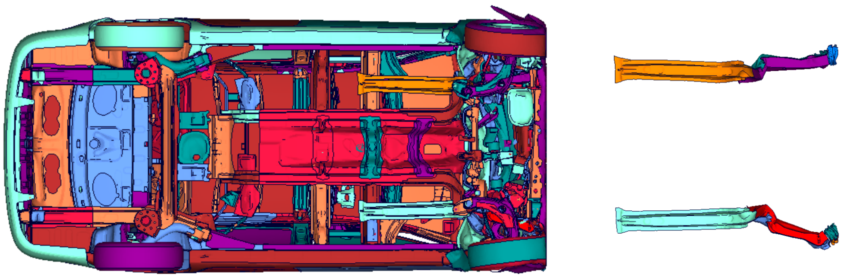


Figure 7.7: Bottom view of the *YARIS* model after half of the simulation time. Left: the entire model without the protective plate; right: the ten selected components.

assisted by dimension reduction methods. One goal is the detection of clusters corresponding to different deformation patterns in the embeddings of the components. This way, relations between model parameters and the deformation behavior are discovered more easily, and the analysis of car crash simulations is accelerated [BGIT+13, HITG20].

### 7.1.4 SYNthetic Dataset

The dataset *SYN* contains regularly meshed triangles with 153 vertices, whose three corners are bent up and down. Therefore, the deformation of the triangles is parametrized by three variables $\alpha, \beta, \gamma \in [-1, 1]$ that describe the deformation of the corners. I sample $25 \times 25 = 625$ noisy triples from a plane in 3D by selecting 25 equally distributed pairs $(\alpha, \beta) \in [-1, 1]^2$ and calculate $\gamma$ by

$$\gamma = 0.8 \cdot \alpha - 0.5 \cdot \beta + \mathcal{N}(0, 0.15^2). \tag{7.1}$$

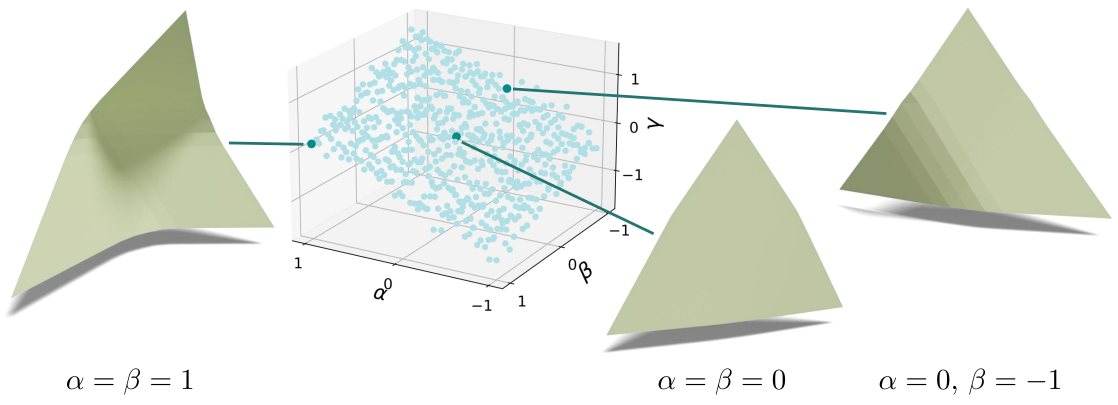


Figure 7.8: Triangular Meshes from the synthetic dataset *SYN* sampled from a plane in 3D.

Figure 7.8 illustrates sample triangles from the dataset. Since the deformation of the dataset is parametrized by three variables that lie approximately on a two-dimensional manifold, I expect to detect this structure in the embedding space. I include 9% of the triangles in the testing set to train the mesh autoencoders.

## 7.2 Remeshing to Semi-regular Meshes

The CoSMA networks, introduced in section 5.2, handle the regional patches of semi-regular surface meshes. Therefore, all shapes in the diverse collections need to be represented by a semi-regular mesh. For the different datasets, I remesh a template mesh for a set of meshes that share the same connectivity, applying and combining losses and regularizers proposed in section 6.2.

As explained in section 6.1, I calculate a coarse base mesh $\mathcal{M}_{coarse,k}$ with a given number of faces $k$ that approximates the surface of the shape. I choose $k$ as low as possible to increase the size of the patches that the CoSMAs handle while maintaining a high remeshing quality.

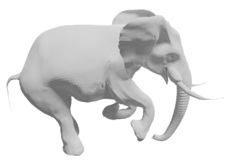
The semi-regular remeshing is done in two iterative and one projection step:

(i) The coarse base mesh is refined to level 3 using midpoint subdivisions. The resulting semi-regular mesh is fitted to the original irregular mesh using a regularized loss.

(ii) The resulting semi-regular (SR) mesh of refinement level 3 is refined once more to refinement level 4. This mesh is again fitted to the original irregular mesh.

(iii) Finally, I project the vertices of the semi-regular mesh to the surface of the irregular one. This final semi-regular mesh representation is referred to as the projected semi-regular (SR) mesh.

The remeshing pipeline for the *FAUST* template mesh is visualized in Figure 7.9.



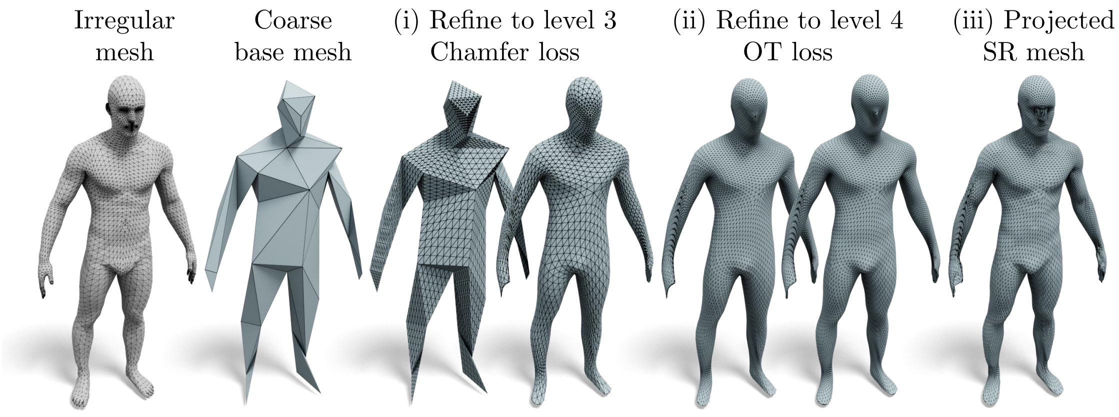

Irregular mesh
Coarse base mesh
(i) Refine to level 3 Chamfer loss
(ii) Refine to level 4 OT loss
(iii) Projected SR mesh

Figure 7.9: Remeshing pipeline for the *FAUST* template mesh to a semi-regular mesh representation of refinement level 4.

### 7.2.1 Selecting Loss Functions and Regularizers

Depending on the mesh structure, different loss functions fit the vertices of the semi-regular mesh $\mathcal{M}_{SR}$ to the irregular mesh vertices $\mathcal{M}_{IR}$.

Using the chamfer distance $d_C(\mathcal{M}_{IR}, \mathcal{M}_{SR})$ generally creates smooth meshes if adequately regularized. Nevertheless, its usage for higher refinement levels leads to artifacts in areas with finer deformations, for example, the face, see Figure 7.10. On the other hand, if the distance between the irregular mesh and the semi-regular mesh is higher, the OT loss $d_{OT}(\mathcal{M}_{IR}, \mathcal{M}_{SR})$ fails and its minimization creates artifacts as overlapping areas of the mesh.

Both the chamfer and the OT loss consider points sampled for the surface mesh as point clouds, making regularization of the semi-regular mesh surface crucial. Figure 7.10 shows how the meshes overlap locally and how the surface wrinkles if fitted without any regularizer. The figure visualizes how the remeshing results change if regularizing the edge length $r_E(\mathcal{M}_{SR})$, the normal consistency $r_N(\mathcal{M}_{SR})$, and smoothing the Laplacian $r_L(\mathcal{M}_{SR})$ with regularization weights $\omega_{\mathcal{E}}$, $\omega_N$, and $\omega_L$, respectively.

The above observations lead to a combined application of the chamfer distance and the OT loss. The first refinement step (i) is fitted using the regularized chamfer distance, and in the last refinement step (ii), the regularized OT loss is minimized since the distance between the surface meshes is lower now. I apply the resulting remeshing pipeline to shapes from the *GALLOP* and *SCAPE* datasets to semi-regular meshes of refinement level 4. For the *SCAPE* meshes, I did not change any regularization weights but used the same hyperparameters from the *FAUST* setup. Figure 7.11 shows the remeshing results. While the regularized chamfer distance leads to smooth surfaces, the OT loss in the last refinement step allows for a more detailed remeshing result, also describing smaller structures of the surface, for example, the ears.

| | OT Loss | | Chamfer Loss | | | Chamfer & OT loss |
|---|---|---|---|---|---|---|
| $\omega_{\mathcal{E}} =$ | 0 | 0.0001 | 0 | 0.1 | 1 | 1, 0.0001 |
| $\omega_L =$ | 0 | 0.01 | 0 | 0.01 | 0.1 | 0.1, 0.01 |
| $\omega_N =$ | 0 | 0.001 | 0 | 0.001 | 0.001 | 0.001 |

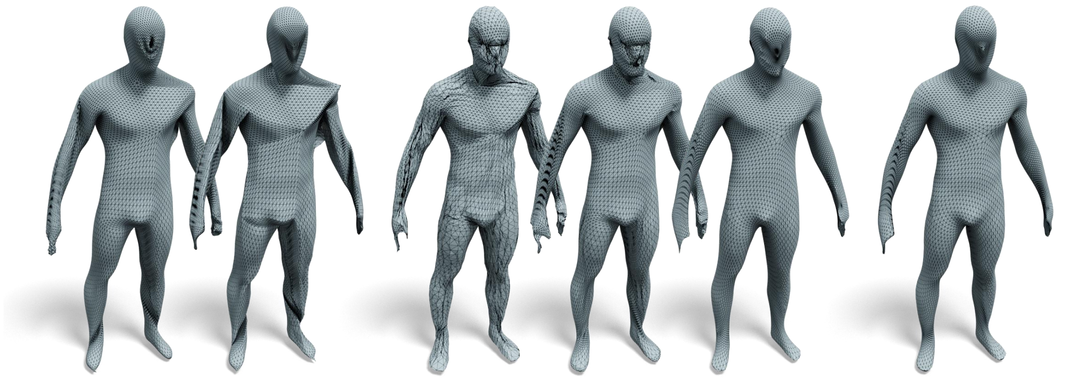


Figure 7.10: Remeshing the *FAUST* template mesh using different loss functions and regularization losses. Without the regularizers artefacts are visible in the mesh representing the head

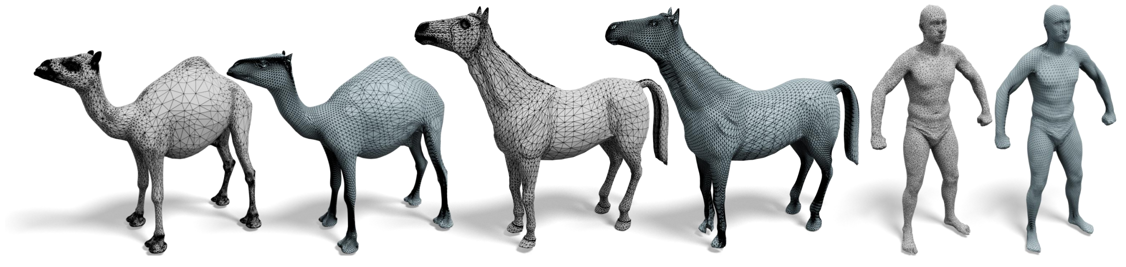
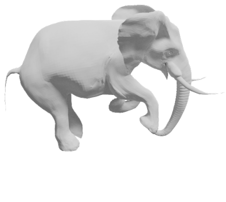


Figure 7.11: Selected remeshing results to semi-regular meshes of refinement level 4 for the *GALLOP* and *SACPE* datasets. Irregular meshes in gray, semi-regular meshes in light blue. 100 to 115 coarse faces.

### 7.2.2 Additional Regularizers

Some meshes require some of the additional loss functions or regularizers introduced in sections 6.2.1 and 6.2.2. Since the car components from the *TRUCK* and *YARIS* datasets have boundaries, I add the weighted boundary loss $\omega_B \cdot d_B(\mathcal{M}_{IR}, \mathcal{M}_{SR})$ to ensure that the semi-regular mesh covers the surface well. The remeshing results and the effect of the boundary loss are visualized in Figure 7.12.

The elephant template mesh requires the geodesic distance as an additional regularizer for the chamfer distance because otherwise, the ears are fitted to the shoulders, which leads to high remeshing and, therefore, reconstruction errors of the autoencoders. The effect, when applying the chamfer distance weighted by geodesic distance $d_{C,geod}(\mathcal{M}_{IR}, \mathcal{M}_{SR})$, is visualized in Figure 7.13.



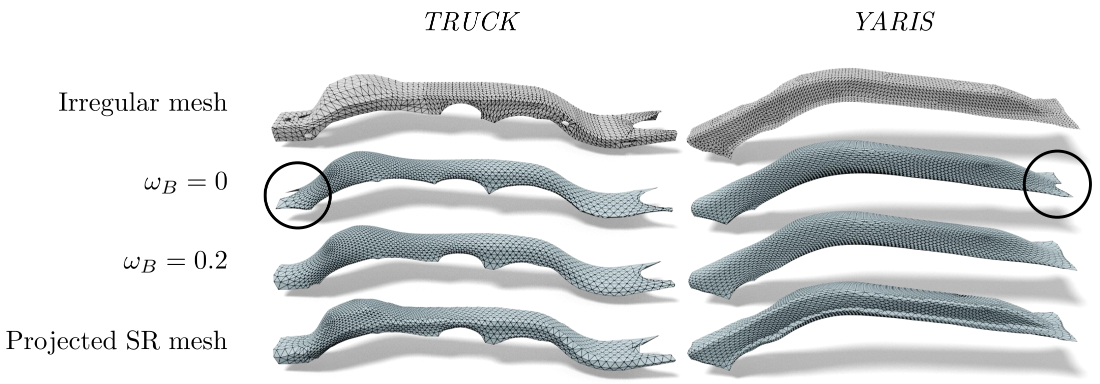


Figure 7.12: Selected remeshing results to semi-regular meshes of refinement level 4 for the *TRUCK* and *YARIS* datasets without and with weighted boundary loss $\omega_B \cdot d_B(\mathcal{M}_{IR}, \mathcal{M}_{SR})$. 15 to 20 coarse faces.

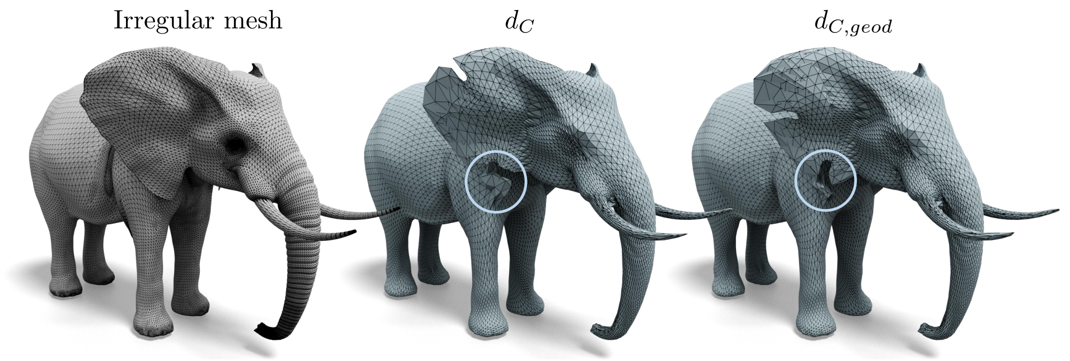


Figure 7.13: Remeshing results to semi-regular meshes of refinement level 4 for the elephant from the *GALLOP* dataset when using the chamfer distance $d_C(\mathcal{M}_{IR}, \mathcal{M}_{SR})$ or chamfer distance regularized depending on the geodesic distance $d_{C,geod}(\mathcal{M}_{IR}, \mathcal{M}_{SR})$. 120 coarse faces.

### 7.2.3 Remeshing Error

All surface meshes are remeshed to the semi-regular mesh representation for analysis by the CoSMA networks. Nevertheless, the final reconstruction error is computed on the irregular meshes after a second remeshing back to the irregular meshing using the parametrization calculated during the remeshing, see section 6.2.4. These two steps of remeshing result in an error that can be measured between the vertices of the original mesh and the projected semi-regular vertices onto the irregular mesh. Using the same error calculation for evaluating the reconstruction quality, I provide the remeshing errors in Table 7.1. Therefore, a perfect reconstruction of the CoSMA networks on the semi-regular meshes results in these reconstruction errors measured on the irregular meshes.

The table also lists the Hausdorff Distance $d_H(\mathcal{M}_{IR}, \mathcal{M}_{SR})$ between the irregular meshes and their remeshing results. To make this distance comparable to the above error,

| Dataset | Shape | $\lvert V_{IR}\rvert$ | Vertex-wise error | Hausdorff |
|---|---|---:|---|---:|
| | Horse | 8,431 | 0.03 ± 0.002 \| 0.03 | 128.0 |
| *GALLOP* | Camel | 21,887 | 0.08 ± 0.003 \| 0.09 | 359.9 |
| | Elephant | 42,321 | 0.41 ± 0.003 \| 0.41 | 1794.6 |
| *FAUST* | | 6,890 | 0.09 ± 0.058 \| 0.07 | 126.6 |
| *SCAPE* | | 12,499 | 0.08 ± 0.009 \| 0.08 | 270.2 |
| *TRUCK* | | 696 to 1736 | 0.06 ± 0.004 \| 0.07 | 56.0 |
| *YARIS* | | 534 to 3205 | 0.05 ± 0.010 \| 0.06 | 57.3 |

Table 7.1: Remeshing errors on all datasets. Average vertex-wise error, standard deviation, and median between the original and irregular meshes resulting from remeshing twice, and the average Hausdorff distance $d_H(\mathcal{M}_{IR}, \mathcal{M}_{SR})$ between the original and semi-regular remeshing results. For the car datasets, component-wise errors are averaged.

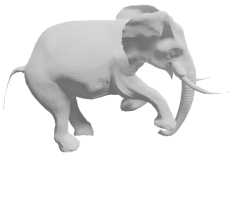
I multiply it by the number of vertices in the irregular mesh. The distance describes the highest distance between a vertex from one mesh to its closest projection on the other mesh, see section 2.2.1. Therefore, it can be interpreted as a measure of stability since it measures the worst closest projection.

Note that the errors between the different shapes are not comparable. The complexity of the remeshing task highly depends on the mesh structure, the quality of the irregular mesh representation, and the number of vertices in the irregular mesh. For more densely sampled vertices on the surface, the remeshing error is expected to be lower since the surface description is better. At the same time, adding up the vertex-wise squared errors for a shape (instead of averaging), on the other hand, leads to higher errors for a higher number of vertices. Table 7.1 lists the number of vertices and the shapes.

Nevertheless, we can compare the ratio between the Hausdorff distance and the vertex-wise error for the different meshes. It measures how much larger the error at the worst fitting surface area is compared to the average error. Note how it is the lowest for the car components, which are the most simple meshes, and how the remeshing result fits smoothly to the irregular mesh. The *FAUST* remeshing results also have almost no artifacts. Therefore, their ratio is low. It is the highest for the elephant remeshing result, where the regularization of the remeshing cannot impede all the artifacts.

### 7.2.4 Summary

The proposed semi-regular remeshing algorithm is flexible and remeshes all given surface meshes to a semi-regular mesh representation. The resulting semi-regular meshes resemble the irregular meshes more closely than baseline remeshing methods, whose results are visualized in Figure 2.2. Compared to the baseline methods, the regular surface meshes can have holes or boundaries, and not be watertight. This is due to the Garland-Heckbert algorithm that also coarsens non-manifold edges and holes and the flexible loss functions. The resulting semi-regular mesh depends highly on this coarse mesh representation. If



the coarse representation has unevenly distributed vertices or non-manifold edges, the refined mesh has the same unwanted characteristics. Nevertheless, the CoSMA networks only consider connectivity, so the edges should be of similar lengths. Also, non-manifold edges unavoidably lead to high remeshing errors. Therefore, the regularization of edge length and face areas, as well as the adaption avoiding non-manifold edges, are essential for semi-regular mesh representations that the neural networks can handle.

The vertices of the resulting semi-regular meshes are generally equally distributed over the surface as a consequence of the regularization. Only in areas with a high degree of detail, such as the fingers of the human meshes, the semi-regular surface meshes cannot capture all the details and simplify the details. At the elephant's thin ears, artifacts can not be entirely avoided. In general, the chamfer loss and the optimal transport-inspired loss, in combination with the regularizers, lead to stable remeshing results using gradient descent for optimization, which a GPU can speed up. The loss functions consider points on the surface meshes, which makes the pipeline flexible in the kind of surface meshes it can handle. Nevertheless, the hyperparameters and regularization weights must be carefully chosen for the different types of meshes.

# 8 Representation Learning Results

This chapter compares the representation learning results from the proposed surface mesh autoencoders spatial CoSMA, spectral CoSMA, and CCLB-autoencoder to several baseline networks. After detailing the experiment setup in section 8.1, I compare the reconstruction qualities of each network in section 8.2. Afterward, section 8.3 evaluates the learned low-dimensional features, including visualizations in two or three dimensions. Finally, section 8.4 provides generative results of my autoencoders by sampling from and interpolating in the learned embedding spaces.

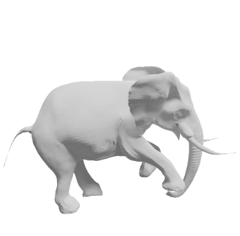
## 8.1 Experiment Setup

I compare the three introduced autoencoders, spatial CoSMA, spectral CoSMA, and CCLB-AE, to several baseline autoencoders. The three baseline mesh autoencoders CoMA [RBSB18], Neural3DMM [BBP+19], and MeshConv [ZWL+20] have been explained in more detail in section 4.2. Additionally, we built one baseline autoencoder based on the SubdivNet convolutional method for semi-regular meshes [HLG+22]. The architecture of the SubdivNet autoencoder is detailed in section 8.5. It also handles the semi-regular surface mesh representations. All baselines handle entire meshes and require them to share the same mesh connectivity, so I have to split some datasets into several subsets of meshes to process them using these methods.

For the CoSMA models, I normalize the 3D coordinates of each mesh to range $[-1, 1]$ relative to the coordinates' ratio and translate every input patch to zero mean. The CCLB-autoencoder handles meshes normalized to unit area and centered to zero concerning the average of the vertex areas, as proposed by DiffusionNet [SACO22]. For the original training of CoMA and Neural3DMM, every vertex was normalized to zero mean and a standard deviation of one. Nevertheless, a vertex-wise normalization is inappropriate since I aim to learn the mesh deformation and handle meshes with different connectivities. Therefore, to train the baseline methods, I normalize the entire mesh to zero mean and standard deviation of one.

The latent representation is a compressed representation of the input data. Therefore, its size $hr$ is essential when comparing different autoencoder architectures to each other. A higher latent dimension can capture more details but may lead to overfitting, while a lower dimension may result in lossy compression and, thus, loss of important information. For comparability of the results, the different architectures produce low-dimensional representations of similar size $hr$.

The CoSMA embedding dimensions $hr = hr_p \cdot \#(\text{patches})$ for every mesh category depend on the number of patches, which is equal to the chosen number of faces in the coarse mesh created during the remeshing procedure. All networks use a patch-wise



| Method | Learnable Parameters | *GALLOP* Horse | Camel | Elephant | *FAUST* | *TRUCK* | *YARIS* |
|---|---|---|---|---|---|---|---|
| CoMA* | 192,935 | 100 | 100 | 100 | 100 | - | - |
| Neural3DMM | 7,558,147 | 1024 | 1024 | 1024 | 1024 | - | - |
| SubdivNet | 19,675,855 | 1024 | 1024 | 1024 | 1024 | - | - |
| MeshConv | 4,076,917 | $9 \cdot 454$ | $9 \cdot 1122$ | - | $9 \cdot 400$ | - | - |
| spatial CoSMA | 30,090 | $10 \cdot 100$ to $10 \cdot 120$ | | | $10 \cdot 100$ | $10 \cdot 15$ to $10 \cdot 20$ | $10 \cdot 15$ to $10 \cdot 20$ |
| spectral CoSMA | 23,053 | | | | | | |
| CCLB-AE | 928,665 | $40 \cdot 26$ | | | $22 \cdot 45$ | $18 \cdot 11$ | - |

Table 8.1: Dimension $hr$ of the hidden representation calculated by the different models and number of learnable parameters when analyzing *FAUST*.
$^*$: no convergence for higher latent dimension.

embedding dimension of $hr_p = 10$. Table 8.1 lists the resulting shape-wise embedding dimensions. The embedding dimension of the CCLB-autoencoder depends on the number of features $F$ output by the DiffusionNet layer in the encoder and the dimensionality $k_2$ of the CCLB. $F$ and $k_2$ for all shape collections are chosen in a way that ensures that the embedding dimension $hr = k_2 \times F$ is similar to the CoSMA models, see Table 8.1. For the baseline architectures, I chose to keep the latent embedding dimensions as in the original papers or increase them if the original embedding dimensions were lower than those of the CoSMAs and CCLB-autoencoder. Additionally, Table 8.1 lists the number of learnable parameters of all autoencoder networks for the *FAUST* dataset.

The CoSMA networks (implemented in PyTorch [PGM+19]) are trained with the adaptive learning rate optimization algorithm [KB15] using a learning rate of 0.001 (spatial CoSMA) or 0.0001 (spectral CoSMA). To augment the data in the case of the *GALLOP* and the *FAUST* datasets, I rotate the regional patches by $0°, 120°$, and $240°$. Similarly to the CoSMA models, the CCLB-autoencoder is implemented in PyTorch [PGM+19] and trained using the Adam optimizer [KB15] with an initial learning rate of 0.001. When using unsupervised p2p maps, we weight the reconstruction loss using $\lambda_{rec} = 10$, see section 5.3.4, and apply dropout inside the DiffusionNet blocks.

## 8.2 Reconstruction Results

The analysis is initiated by conducting a conventional reconstruction experiment. The evaluation of the reconstruction quality of the proposed mesh autoencoders to the baseline methods is followed by additional experiments to analyze the characteristics of the specific models. Finally, I ablate the proposed model architectures.

To obtain reconstructed meshes, I first encode the vertex positions $V$ of mesh $\mathcal{M}$ from the test set, which was never seen during the training phase, into a latent representation $en(V)$. Subsequently, I decode the latent representation using my decoder $V_{rec} = de(en(V))$ and compare the reconstructed shape $\mathcal{M}_{rec}$ to the initial shape $\mathcal{M}$ to assess the reconstruction quality. To this end, I compute the squared Euclidean distance

| Method | *GALLOP* Horse | Camel | Elephant |
|---|---|---|---|
| CoMA | 3.2 ± 0.3 | 7.8 ± 1.4 | 24.3 ± 4.4 |
| Neural3DMM | 4.7 ± 0.1 | 12.4 ± 0.1 | 29.7 ± 3.5 |
| SubdivNet | 7.0 ± 1.8 | 7.9 ± 0.7 | 54.4 ± 8.7 |
| MeshConv | 7.3 ± 0.1 | 19.2 ± 0.4 | – |
| Spatial CoSMA | 1.5 ± 0.05 | 3.7 ± 0.03 | 16.8 ± 0.9 |
| Spectral CoSMA | 1.2 ± 0.01 | 3.3 ± 0.01 | 20.0 ± 0.3 |
| CCLB-AE | **0.1 ± 0.02** | **0.4 ± 0.05** | **1.4 ± 0.3** |

Table 8.2: Reconstruction errors on the *GALLOP* dataset. Only the proposed CoSMAs and the CCLB-autoencoder train one model on three animals together.

between the vertex coordinates of the input shape $V$ and its reconstruction $V_{rec}$ and add these vertex-wise errors up to determine the reconstruction error:

$$\text{RecErr}(\mathcal{M}, \mathcal{M}_{rec}) = \sum_{\substack{v \in V \\ v_{rec} \in V_{rec}}} \|v - v_{rec}\|_2^2 \,. \tag{8.1}$$

To obtain consistent results, the reconstructions of the remeshed semi-regular meshes are projected back to the irregular mesh using the parametrization obtained during the remeshing. For uniform results, I normalize all meshes into the range $[-1, 1]$. All reported reconstruction errors are mean errors over three randomly initialized runs, and $\pm$ denotes the standard deviation.

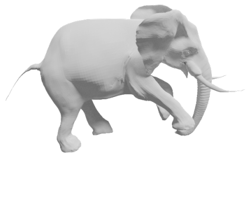
### 8.2.1 Evaluation of Reconstruction Qualities

I compare the reconstructed meshes from my proposed models and the baseline mesh autoencoders for each dataset.

#### GALLOP Dataset

The three introduced autoencoders can be trained on the three animals together, although they do not share the same connectivity. The baselines have to be trained separately for every animal. The MeshConv network for the elephant with 42,321 vertices has more than 20 million trainable parameters and could not be trained on a GPU with 40 GB. Table 8.2 lists the reconstruction errors.

The baseline reconstructions are worse, more unstable in the legs, and have incorrectly positioned limbs, as Figure 8.1 illustrates. Also, when considering only one animal, the datasets are small since they contain only 34 training shapes. This is also notable in the higher standard deviation concerning the reconstruction error for the baseline methods. On the other hand, the reconstructions from the CoSMAs and the CCLB-autoencoder are not only of higher quality but also more stable. In fact, the reconstruction errors



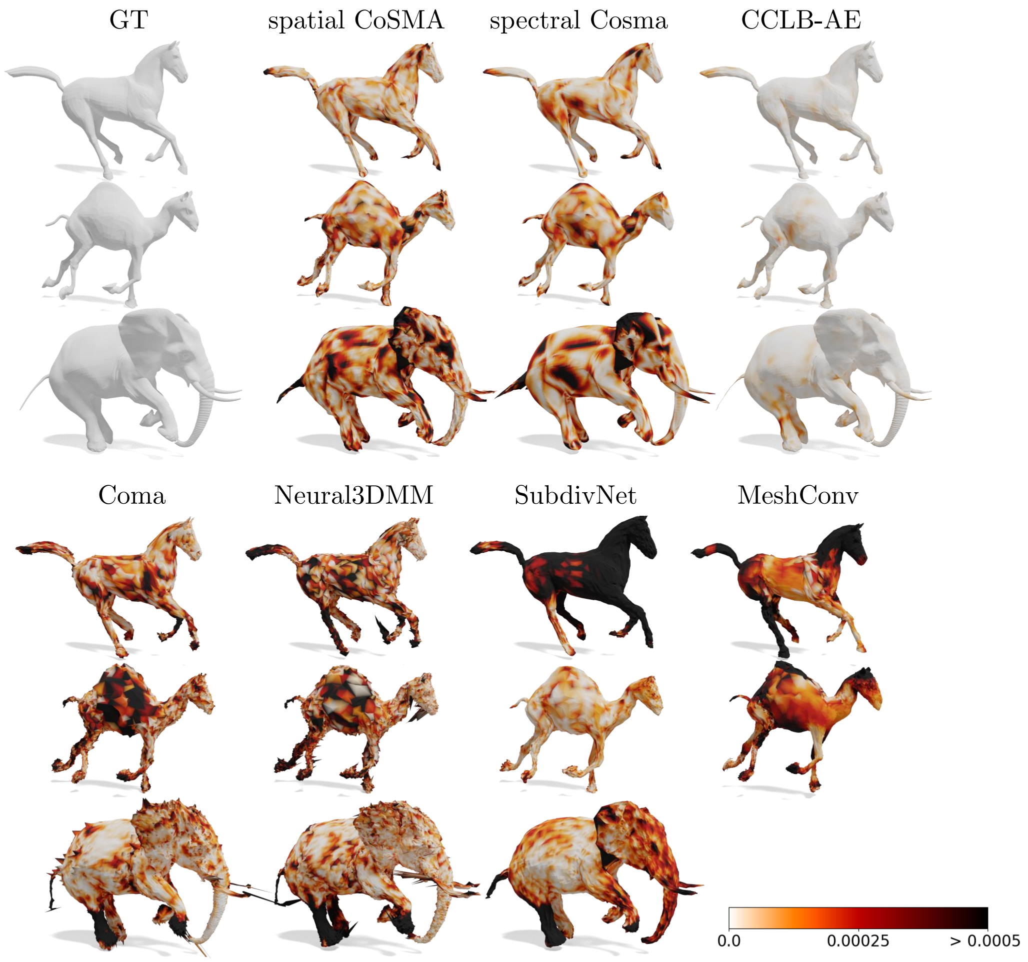


Figure 8.1: Ground truth (GT) and reconstructed test meshes from the *GALLOP* dataset. Vertex-wise reconstruction error is highlighted.

of the CoSMA models are at least 17% lower, and the ones of the CCLB-autoencoder are at least 94% lower than the reconstruction errors of the baselines. Qualitatively, the reconstructed meshes are smooth, deform naturally, and do not have any outlier vertices, which is not the case for some baseline methods, see Figure 8.1.

### Human Body Datasets

I conduct the "unknown individuals" and "unknown poses" experiments on the *FAUST* dataset and list reconstruction errors in Table 8.3. The reconstruction errors on the "unknown individuals" experiments are generally lower since the difference between the train and test samples solely originate from different body shapes and sizes. On the other hand, the test shapes in the "unknown poses" experiments are in poses that are not contained in the training set. It makes this a more challenging setup since the autoencoders need to generalize well to the test shapes.

| Method | Unkown poses *FAUST* | Unknown indiv. *FAUST* | Unkown poses *FAUST* | Unkown poses *SCAPE* |
|---|---|---|---|---|
| CoMA | 569.3 ± 203.1 | 28.3 ± 6.4 | | |
| Neural3DMM | 246.2 ± 5.4 | 10.4 ± 0.9 | | |
| SubdivNet | 783.7 ± 58.1 | 47.1 ± 22.8 | | |
| MeshConv | 18.2 ± 2.2 | 3.5 ± 0.4 | | |
| Spatial CoSMA | 2.5 ± 0.4 | 1.0 ± 0.03 | 1.55 ± 0.1 | 2.41 ± 0.2 |
| Spectral CoSMA | **1.0 ± 0.02** | 0.9 ± 0.01 | **1.01 ± 0.001** | **1.45 ± 0.01** |
| CCLB-AE | 2.8 ± 0.1 | **0.7 ± 0.02** | 2.97 ± 0.04 | 3.30 ± 0.17 |

Table 8.3: Reconstruction errors on the two experiment setups for the *FAUST* dataset and when training one model for *FAUST* and *SCAPE* shapes.

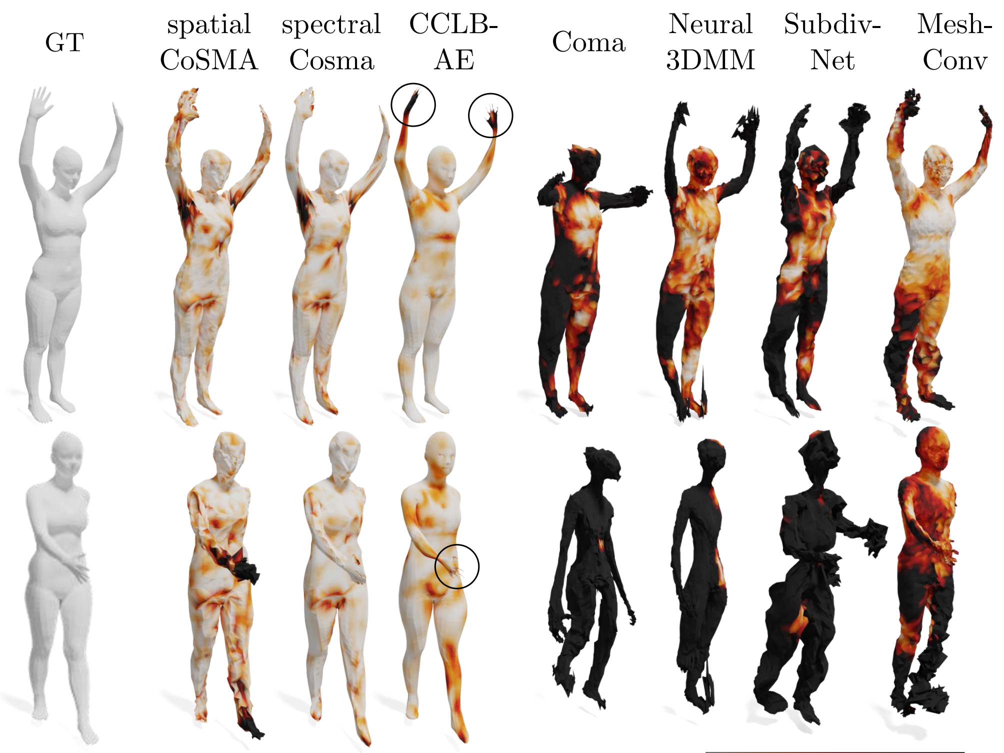
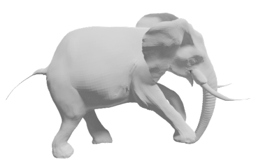


Figure 8.2: Reconstructed test meshes from the *FAUST* dataset of the “unknown poses” experiment setups. Vertex-wise reconstruction error is highlighted.



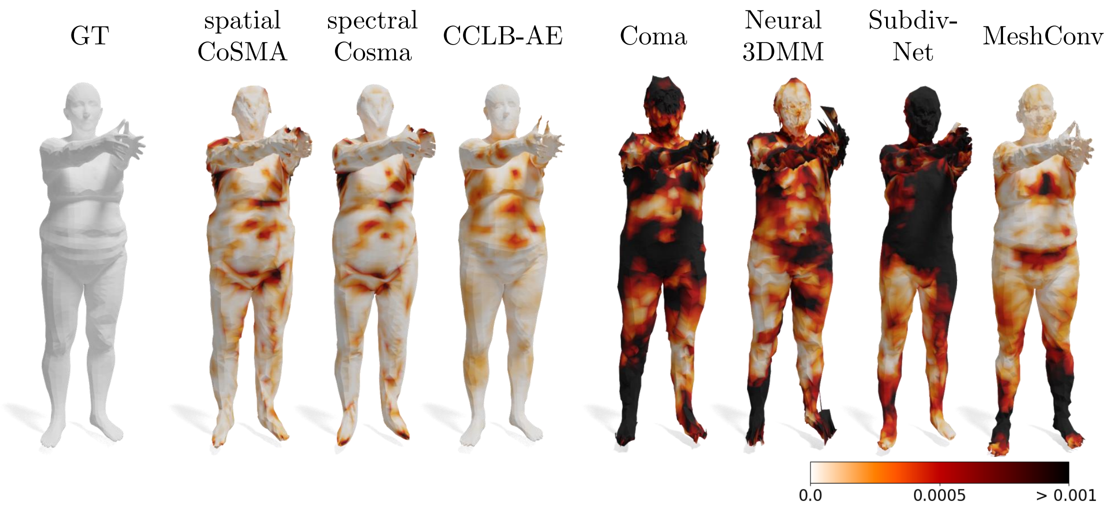


Figure 8.3: Reconstructed test meshes from the *FAUST* dataset of the "unknown individuals" experiment setups. Vertex-wise reconstruction error is highlighted.

The three introduced autoencoders have at least 80% lower reconstruction errors than all the baseline methods in the "unknown poses" setup and 70% lower errors in the "unknown individuals" setup. The spectral CoSMA reconstruction quality is superior to the spatial CoSMA for both setups.

The patch-based approach is convenient for reconstructing unknown poses, so the reconstruction quality of the CoSMAs is higher than for the CCLB-autoencoder. Figure 8.2 visualizes how the CCLB-autoencoder fails to reconstruct well-formed arms in unknown positions. No training shape includes arms or legs in the unknown step position, which makes its reconstruction more challenging, and baseline reconstructions fail entirely.

On the other hand, for known poses and unknown individuals, the CCLB-autoencoder reaches lower errors than the CoSMA networks. Figure 8.3 compares reconstructed test meshes from the "unknown individuals" setup to each other. Note that the CCLB-AE reconstructions are more detailed than the CoSMA reconstructions. Also, in the case of the CoSMAs, some patch boundaries are noticeable, and the reconstructions in feet and hands are less detailed. When comparing the spatial and spectral CoSMA reconstructions, it is noticeable that the spectral CoSMA reconstructed shapes have smoother surfaces.

To demonstrate the reconstruction quality of the introduced methods on datasets that contain multiple mesh connectivities, I add the *SCAPE* dataset to the "unknown poses" experiment. I use the same *FAUST* test shapes plus ten additional *SCAPE* shapes for testing. Note that for this experiment, the CCLB-autoencoder reconstruction quality is slightly worse when compared to the *FAUST*-only "unknown poses" setup. On the other hand, the results of the CoSMAs improve in comparison to the previous experiments since the networks seem to take advantage of the larger and more diverse training dataset. Comparing the results to the CCLB-autoencoder reconstructions, the

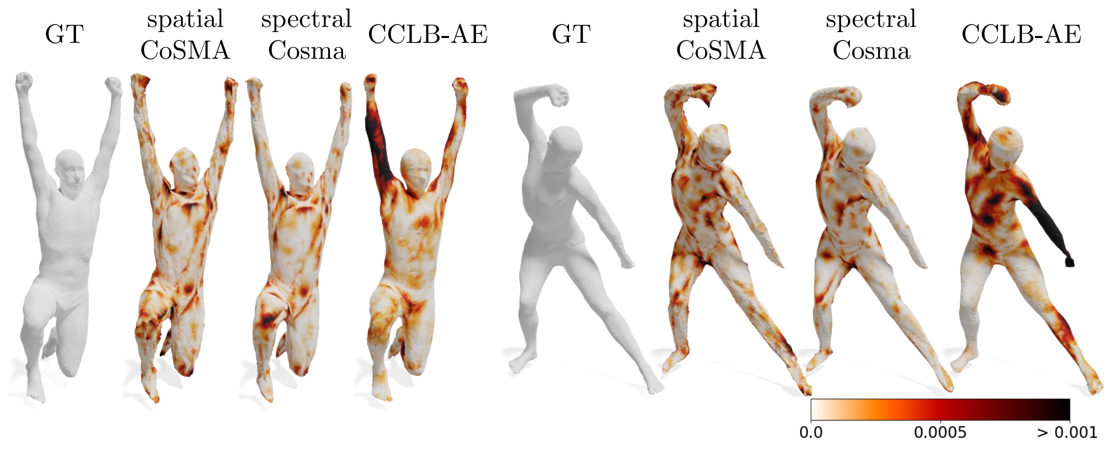


Figure 8.4: Reconstructed test meshes from the *SCAPE* dataset of the "unknown poses" experiment setup. Vertex-wise reconstruction error is highlighted.

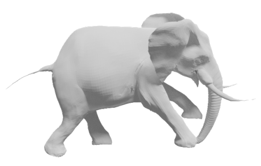
spectral CoSMA errors are more than 60% lower. Figure 8.4 visualizes reconstructed SCAPE test meshes.

### TRUCK Dataset

Due to the relatively large number of different mesh connectivities in the dataset, I restrict myself to testing the three presented autoencoding methods on the *TRUCK* dataset because they can be trained on all parts at once. Also, I use only 30% of the simulations for the training of the larger CCLB-autoencoder to reduce the runtime. Table 8.4 compares the reconstruction errors, and Figure 8.5 visualizes reconstructed test meshes.

The reconstruction errors are the lowest for the CCLB-autoencoder, and the reconstructed meshes are of very high quality. The spectral CoSMA reconstruction errors are five times higher. The actual reconstruction is of very high quality. The higher error is partly due to the remeshing of some indentations and bends of the components, as visible in Figure 8.5. In fact, the remeshing error from Table 7.1 is a third of the reconstruction error, whereas it is less than a tenth for the other datasets. The spatial CoSMA reconstructs the general shape and deformation of the components but fails to reconstruct details, which is why the error is almost 70 times higher than the error from the CCLB-autoencoder.

Table 8.4 allows for a comparison of the CoSMA reconstruction results for refinement levels $rl = 3$ and $rl = 4$. The spatial CoSMA reconstruction quality decreases when the refinement level increases. This is due to the fixed kernel size $KS = 2$. Since the mesh is finer, the neighborhoods considered by a spatial filter using kernel size $KS = 2$ cover smaller surface areas. The spectral CoSMA using $K = 6$, on the other hand, considers almost the entire patch in spectral representation. Therefore, increasing the refinement level does not impair the reconstruction quality. In fact, the errors decrease for finer



| Method | | Error |
|---|---|---|
| Spatial CoSMA | $rl = 3$ | $1.15 \pm 0.13$ |
| Spectral CoSMA | $rl = 3$ | $0.27 \pm 0.01$ |
| Spatial CoSMA | $rl = 4$ | $1.60 \pm 0.13$ |
| Spectral CoSMA | $rl = 4$ | $0.16 \pm 0.01$ |
| CCLB-AE | | **$0.03 \pm 0.001$** |

Table 8.4: Reconstruction errors on the *TRUCK* dataset. The component-wise errors for each training run are averaged.

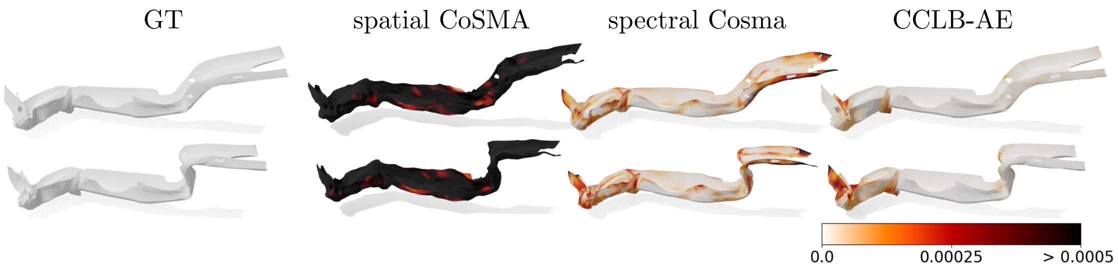


Figure 8.5: Reconstruction of a deformed component from two different simulations from the *TRUCK* dataset. Vertex-wise reconstruction error is highlighted.

meshes.

### SYN Dataset

Since the synthetic dataset *SYN* contains deforming triangular meshes that are regularly meshed, we can compare the reconstruction errors of the three presented models and the CoMA baseline without applying any remeshing. My CoSMA models and the CoMA network calculate a hidden representation of size $hr = 10$, the CCLB-AE $hr = F \cdot k_2 = 3 \cdot 3 = 9$.

Table 8.5 compares the reconstruction errors on *SYN*, where three corners of a meshed triangle bend. In general, the errors of the CCLB-AE are the lowest, followed by CoMA and the spectral CoSMA. Spatial CoSMA reconstruction errors are the highest.

For the first time, CoMA yields better results than the spectral CoSMA approach. This is due to the smaller sample size of only 153 vertices. Recall that the CoMA approach was unstable in analyzing the global deformation patterns, which we do not observe for the small synthetic mesh samples. Also, the patch approach of the CoSMA models is not used in the case of this synthetic dataset, where the shape is represented by one patch. This leads to worse results than the baseline.

Even though the CCLB-AE has the lowest reconstruction error, it has the highest runtime since its DiffusionNet Blocks only allow a batch size of one and have more learnable parameters than the light CoSMA models. Also, since the DiffusionNet Blocks are applied vertex-wise, their number of trainable parameters does not decrease for

| Method | Error | *hr* | Learnable parameters | Runtime per epoch |
|---|---|---|---|---|
| CoMA | $4.1 \pm 1.1$ | 10 | 25,901 | 0.3 s |
| Spatial CoSMA | $24.9 \pm 2.5$ | 10 | 30,090 | 0.5 s |
| Spectral CoSMA | $7.2 \pm 1.0$ | 10 | **23,053** | 0.6 s* |
| CCLB-AE | $\mathbf{0.3 \pm 0.1}$ | $3 \cdot 3$ | 923,782 | 21.3 s |

Table 8.5: Reconstruction errors ($\times 100$), learnable parameters, and runtimes on the synthetic dataset *SYN*.
*: using data augmentation, which increases the runtime $\times$ 3

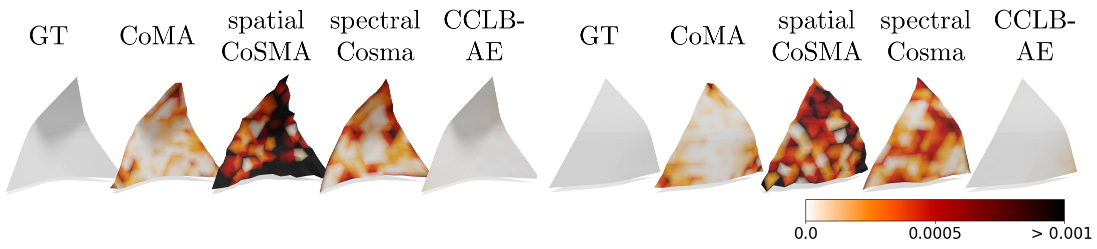


Figure 8.6: Reconstructed test meshes from the *SYN* dataset. Vertex-wise reconstruction error is highlighted.

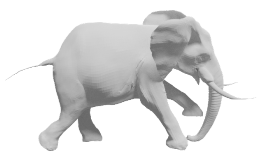
meshes with fewer vertices.

### 8.2.2 Out-of-Distribution Generalization using CoSMA Models

The spectral CoSMA and spatial CoSMA are the only networks that can reconstruct an unseen shape of different connectivity because the regularly meshed patches are handled separately. This admits testing the CoSMAs' out-of-distribution generalization abilities, where models are tested on data whose distribution differs from the training data [YXC+21, ZLQ+23, WLL+23]. The deforming patches are represented on the learned embedding manifold. Therefore, we expect reasonable reconstruction results for meshes whose local deformation patterns stem from similar sources as for the training meshes.

#### Elephant

In the case of the *GALLOP* dataset, we evaluate the reconstruction quality for the elephant's mesh after not presenting it to my network during training. The spectral CoSMA reconstruction error on the elephant is only 20% higher than when training on all three animals together, see Table 8.6. On the other hand, the spatial CoSMA reconstruction error on the unseen elephant is more than 2.5 times as high as when presenting it during training. While the spectral CoSMA reconstruction deteriorates slightly distributed over the entire shape, the spatial CoSMA reconstruction has outliers, and the surface is creased, see Figure 8.7. In this setup, the spatial CoSMA reconstruction quality of the unseen elephant is inferior to many baseline networks, which was not the case when in-



| Method | Error | | |
|---|---|---|---|
| | Horse | Camel | Elephant (only testing) |
| Spatial CoSMA | 1.5 ± 0.130 | 4.1 ± 0.37 | 46.0 ± 3.06 |
| Spectral CoSMA | **1.2 ± 0.005** | **3.3 ± 0.03** | **24.1 ± 0.25** |

Table 8.6: Reconstruction errors on the *GALLOP* dataset when excluding the elephant during training.

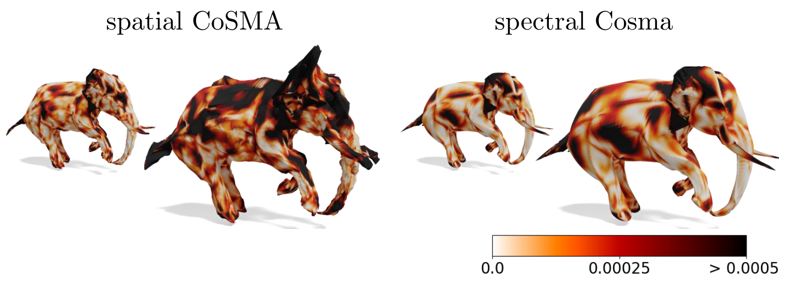


Figure 8.7: Reconstruction of the elephant from the *GALLOP* dataset, when excluding it during training. In small, I visualize the reconstruction result when including the elephant in the training set. Vertex-wise reconstruction error is highlighted.

cluding it in the training data. This highlights the improved transfer learning capability of the spectral approach.

### Out-of-Distribution Generalization on Different Datasets

Additionally, I test trained CoSMAs on a dataset containing structurally different shapes that have not been presented to these networks during training. Table 8.7 provides these results.

I first test the out-of-distribution generalization of the CoSMA networks on the *GALLOP* and *FAUST* since the patch-wise deformations are of natural origin for both datasets. To this end, I train the model on one and attempt reconstruction on another dataset. The reconstruction errors are compared to the ones after training on the actual dataset.

The spectral CoSMA reconstruction errors are only slightly higher (increase by 8% to 28%) when applying the *FAUST*-trained network to the *GALLOP* testing samples. Conversely, the reconstruction results on the *FAUST* dataset are even as good as the results from the “unknown poses” experiment. In contrast to the spatial approach, where the errors increase drastically, the spectral convolutions seem to capture and generalize the patch deformations and not overfit to training data. The spectral approach takes advantage of the large size and variability of the patches in the datasets. Since the training patches cover many possible regional deformations, a method that generalizes well allows for a successful application to unseen datasets.

| Test Dataset | Training Dataset | spectral CoSMA | spatial CoSMA |
|---|---|---|---|
| *FAUST* | *GALLOP* | **1.1 ± 0.01** | 72.9 ± 20.6 |
| Horse | | **1.3 ± 0.01** | 3.4 ± 0.2 |
| Camel | *FAUST* | **3.7 ± 0.04** | 23.4 ± 23.7 |
| Elephant | | **25.5 ± 0.50** | 48.9 ± 2.6 |
| *TRUCK* | *GALLOP* | 1.6 ± 0.03 | 21.9 ± 4.2 |
| | *FAUST* | **1.3 ± 0.02** | 7.2 ± 4.2 |
| | *TRUCK* | 4.5 ± 0.08 | 20.2 ± 1.3 |
| *YARIS* | *GALLOP* | 2.8 ± 0.07 | 36.3 ± 3.3 |
| | *FAUST* | **2.1 ± 0.01** | 12.1 ± 11.0 |

Table 8.7: Reconstruction errors for CoSMA out-of-distribution generalization experiments. The unknown poses setup was used if *FAUST* has been considered as the training dataset. Note that the errors for *YARIS* are not comparable to *TRUCK* because the number of vertices in the meshes differs.

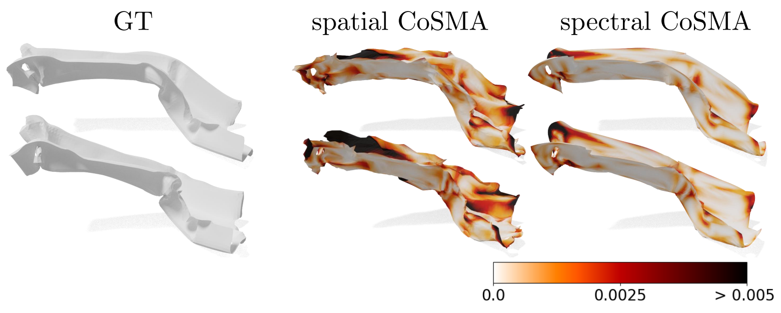
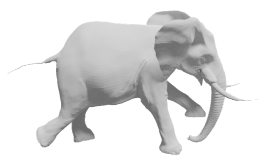


Figure 8.8: Reconstruction of a deformed component from two different simulations from the *YARIS* dataset, when applying a network trained on *FAUST*. Vertex-wise reconstruction error is highlighted.

Secondly, I aim to reconstruct shapes from the two datasets *YARIS* and *TRUCK* containing deforming car components. The selected components are beams with distinct mesh connectivity, and the passenger's safety mainly depends on their deformation behavior during a frontal crash. The reconstruction errors on the *TRUCK* dataset are eigth or ten times higher when using a pre-trained spectral CoSMA in comparison to a trained CoSMA on *TRUCK*.

When testing the reconstruction on the *YARIS* dataset, the trained spectral CoSMAs always generalize better than the spatial ones. Also, pre-trained CoSMAs from the *FAUST* dataset best reconstruct the *YARIS* components. Figure 8.8 visualizes reconstructed meshes. This is an unexpected observation since the local deformation on the *FAUST* surfaces is of natural origin. At the same time, the *TRUCK* deformation also stems from buckling components out of similar material. I assume that the patch deformation is more diverse in the *FAUST* than *TRUCK* dataset, which would explain



| Dataset | global pooling | unsupervised CCLB-AE |
|---|---|---|
| *GALLOP* | | |
| Horse | 4.1 ± 1.2 | **1.2** ± **0.1** |
| Camel | 81.8 ± 27.9 | **3.3** ± **0.3** |
| Elephant | 50.2 ± 9.7 | **11.5** ± **0.4** |
| *FAUST* | | |
| Unknown poses | 394.6 ± 162.1 | **4.7** ± **0.2** |
| Unknown indiv. | 32.8 ± 6.1 | **2.3** ± **0.1** |
| *TRUCK* | 0.16 ± 0.01 | **0.09** ± **0.02** |

Table 8.8: Reconstruction errors between the reconstructed and original meshes of the *GALLOP*, *FAUST*, and *TRUCK* datasets for the unsupervised CCLB-autoencoder experiments.

the better generalization results for the *YARIS* using a *FAUST*-trained model.

### 8.2.3 Unsupervised Shape Representation Learning using CCLB-AE

The baseline mesh autoencoders require point-to-point (p2p) supervision because they only handle meshes with 1-to-1 correspondence, and the CoSMA models require correspondence and fixed mesh connectivity at the patch level. Also, the above CCLB-AE results take advantage of constant mesh connectivity in the datasets and use supervised p2p maps for the CCLB calculation and the p2p training loss.

This section evaluates the CCLB-autoencoder in an unsupervised setup, where we apply unsupervised methods to learn p2p maps between the shapes in the dataset. We add the reconstruction loss to the p2p training loss using the unsupervised p2p maps between input mesh and template mesh (see section 5.3.4). This compensates for faulty mappings by the unsupervised p2p map.

For comparison, we construct a baseline method that uses unsupervised p2p maps and global average pooling instead of the introduced spectral mesh pooling. This corresponds to the case when the dimensionality of the CCLB is 1 ($k_1 = k_2 = 1$), see section 5.3.1 for the proof. Reconstruction errors are listed in Table 8.8.

Due to the highly non-isometric nature of the three animal categories in the *GALLOP* dataset, most unsupervised methods for shape matching fail. Thus, the unsupervised CCLB-AEs is trained on each animal category individually. The CCLB-autoencoder using spectral mesh pooling achieves results comparable to the baselines and outperforms the unsupervised global pooling approach. This demonstrates that learning high-quality mesh autoencoders is possible even without ground truth p2p maps between the shapes.

For the *FAUST* dataset, the unsupervised CCLB-autoencoder results are better than the supervised baseline results that do not require any remeshing, see Figure 8.9 for visualizations of reconstructed test meshes. This demonstrates the usefulness of the proposed approach and the regularization introduced by the losses to mitigate errors in the

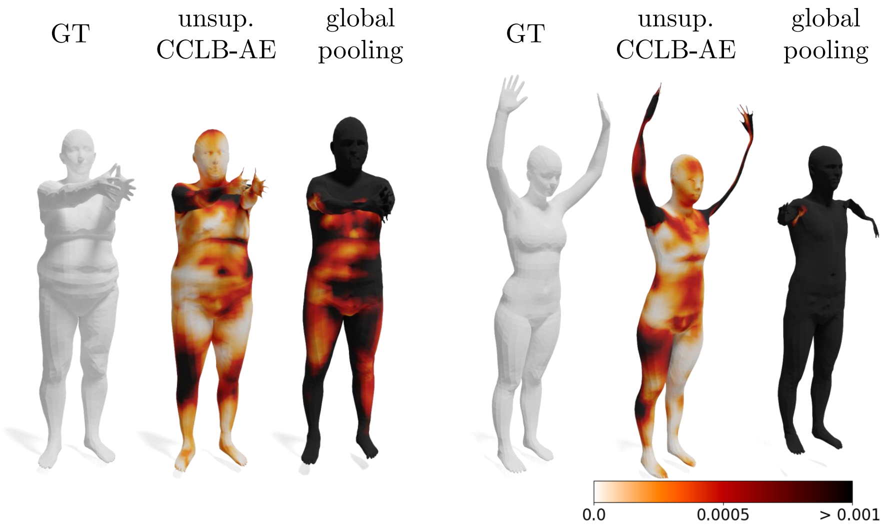


Figure 8.9: Reconstructed test meshes from the unsupervised CCLB-autoencoder experiments on the *FAUST* dataset of the “unknown individuals” (left) and “unknown poses” (right) experiment setups. Vertex-wise reconstruction error is highlighted.

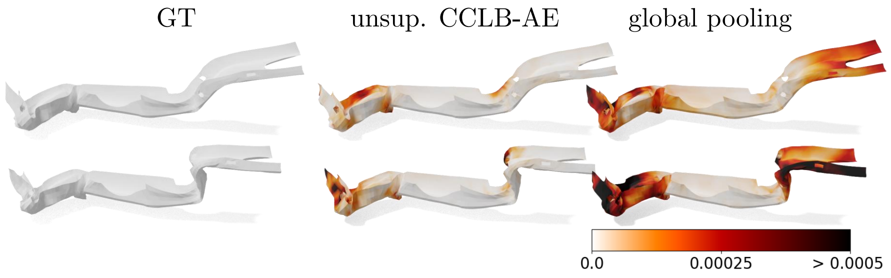
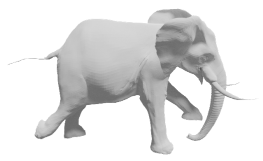


Figure 8.10: Reconstructed test meshes from the unsupervised CCLB-autoencoder experiments on the *TRUCK* dataset. Vertex-wise reconstruction error is highlighted.

maps. Additionally, the novel spectral mesh pooling strongly improves the reconstruction quality for the unsupervised experiments compared to using global pooling in the encoder.

On the *TRUCK* meshes, the unsupervised p2p maps are of high quality such that we do not apply the reconstruction loss and minimize only the p2p training loss using the unsupervised maps. When comparing to the supervised reconstruction errors in Table 8.4, one notices that the unsupervised reconstruction quality from the CCLB-autoencoder for this dataset is superior to both CoSMA reconstructions. Figure 8.10 visualizes the same reconstructed test meshes as for the supervised experiments.



| Setting | *FAUST* "unknown poses" | |
|---|---|---|
| | spatial CoSMA | spectral CoSMA |
| w/o surface-aware loss | 4.06 | 1.08 |
| $K = 4$ | - | 1.04 |
| $2^3$ and $2^4$ channels | 2.72 | 1.05 |
| $hr_p = 8$ | 7.29 | 1.38 |
| **full model** ($hr_p = 10; 2^4$ and $2^5$ channels) | **2.55** | **1.02** |

Figure 8.11: Ablation study on the component of the spatial and spectral CoSMA architecture.

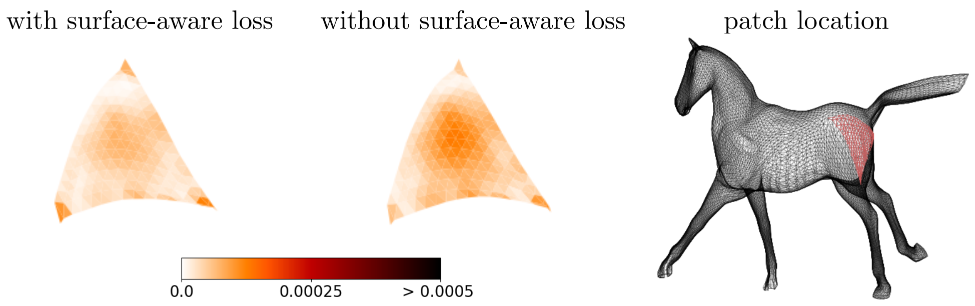


Figure 8.12: Comparison of reconstructed patches of the spectral CoSMA networks without and with the surface-aware loss calculation during training. I depict the face-wise reconstruction errors for the highlighted patch, which are averaged over time.

### 8.2.4 Ablation Studies on Proposed Mesh Autoencoders

I conduct ablation studies concerning the components of the CoSMA and the CCLB-autoencoder pipelines.

#### CoSMAs

We aim to evaluate the impact of the surface-aware loss calculation, the size $hr_p$ of the embedding space, and selected hyperparameters of the surface convolutional layers. The four experiments are conducted in the "unknown poses" setting of the *FAUST* dataset.

The CoSMAs are trained using a surface-aware loss that prevents over-weighting patch boundaries, whose vertices can be included in multiple patches, see section 5.2.3. Figure 8.12 shows how patch-wise training without using the surface-aware loss leads to higher errors in the interior of the patches, especially if the surface is curved. Table 8.11 lists the reconstruction errors when using a patch-wise train MSE compared to the surface-aware loss calculation during training. It also evaluates the effect of a smaller embedding dimension $hr_p$, fewer channels in the convolutional layers, and a lower number

| Setting | *FAUST* dataset |
|---|---|
| w/o spectral pooling | 19.6 |
| w/o reconstruction loss | 4.7 |
| with unsupervised maps | 2.3 |
| with supervised maps | **0.7** |

Table 8.9: Ablation study on the components of the CCLB-autoencoder pipeline.

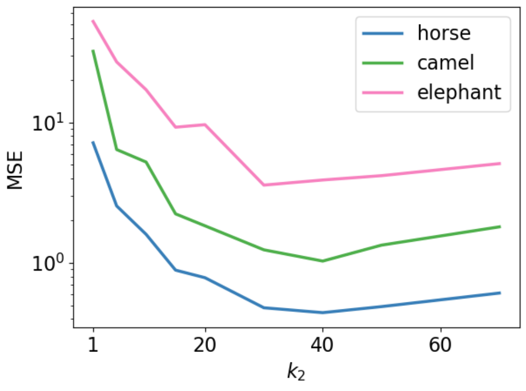


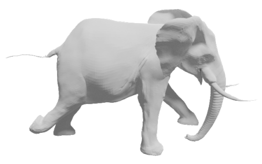
Figure 8.13: Impact of the dimensionality $k_2$ of the CCLB after 100 epochs. Test combinations of CCLB dimension $k_2$ and vertex feature dimension $F$, such that the hidden representation is of size $hr = F \cdot k_2 = 1040$. If $k_2 = 1$ this corresponds to global average pooling.

$K$ of Chebyshev polynomials in the spectral convolutional layers.

### CCLB-Autoencoder

I conduct three experiments in the "unknown individuals" setting of the *FAUST* dataset to ablate the role of the embedding space defined by the CCLB and the dependence on high-quality shape correspondences. The first one uses the entire pipeline with supervised or true p2p maps between the shapes, the second one with unsupervised maps, the third uses unsupervised maps but omits the reconstruction loss, and the last one uses global average pooling instead of spectral mesh pooling in combination with true p2p maps. The averaged features are then duplicated on the template shape, as previous works have done [LBBM18]. The results in Table 8.9 show that all components are necessary for optimal results. Note that spectral mesh pooling significantly contributes to the combined embedding space; using just global pooling leads to an inferior model performance.

Secondly, I evaluate the impact of the dimensionality of the CCLB on the reconstruction error on the *GALLOP* dataset. The size of the embedding space is fixed and equal to the embedding dimension from section 8.3 (1040), which is determined by $k_2$, the di-



| Model | Runtime per epoch (sec) | MSE 50 vs 150 epochs |
|---|---|---|
| spatial CoSMA | 19.2 | 175% |
| spectral CoSMA | 14.1 | **6%** |
| CCLB-AE | **9.7** | 415% |

Table 8.10: Runtimes and convergence speed of proposed models for the *GALLOP*. The last column depicts how much higher the training error is after 50 compared to 150 epochs. All values are averages over three training runs.

mension of the CCLB, multiplied by the feature dimensions $F$ of the encoder. I increase $k_2$ from $k_2 = 1$ to $k_2 = 70$, while adapting the feature dimension $F$ accordingly. Figure 8.13 shows that the CCLB dimension $k_2$ and the feature dimension $F$ must be balanced. Almost no spectral pooling in combination with a high feature dimension (low $k_2$, high $F$) as well as a shallow feature dimension in combination with a high CCLB dimension (low $F$, high $k_2$) lead to a degradation of the performance.

### 8.2.5 Runtime comparison

Table 8.10 compares the runtimes per epoch when training the proposed methods for the *GALLOP* dataset on an Nvidia Tesla A100 GPU. It also compares the training errors after 50 and 150 epochs, when the optimization of all networks has converged, to evaluate the convergence speeds. The CCLB-autoencoder has the fastest runtime per epoch. The spectral CoSMA runtime per epoch is the highest, being twice as high as for the CCLB-autoencoder. Nevertheless, the CoSMAs converge faster than the CCLB-autoencoder. This is due to the smaller network sizes and data augmentation, which rotates the patches and increases the size of the dataset by three. Therefore, although all networks have similar runtimes per epoch, the spectral CoSMA converges the fastest. After 50 epochs, the training error is only 6% higher than after 150.

## 8.3 Low-dimensional Representation

This section evaluates the learned low-dimensional representations for the studied datasets. We aim to make the dataset structures visible by visualizing the learned features. At first, I evaluate the learned low-dimensional representations of the CoSMA networks. Leveraging the patch-based approach, I can localize visible patterns in the low-dimensional feature spaces. Then, the hidden features of the CCLB-autoencoder are evaluated, whose common low-dimensional space allows a comparison of multiple shapes in one embedding space.

### 8.3.1 CoSMA

The CoSMAs calculate $hr_p$-dimensional representations for each patch. To calculate shape-wise features, I concatenate the patch-wise hidden representations, obtaining one

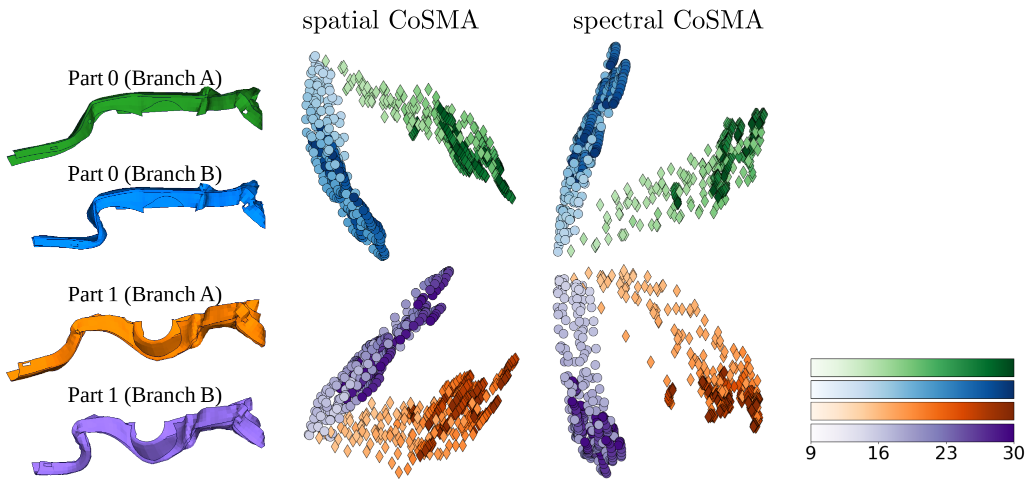


Figure 8.14: Embeddings of two TRUCK components from Figure 7.6 and visualization of the deformation pattern in two branches. The color refers to the part and deformation branch and the color intensity to the simulation time.

embedding for each shape of size $hr = hr_p \cdot \#(\text{patches})$. In all experiments $hr_p = 10$. For the visualization, I calculate 2D embeddings of the concatenated hidden patch-wise representations using the linear dimension reduction method Principal Component Analysis (PCA) [Pea01]. This allows an analysis of several surface meshes represented by the same semi-regular mesh in a low-dimensional space.

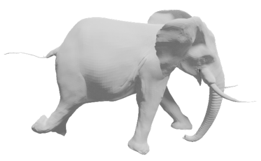
Also, the patch-wise hidden representations of size $hr_p$ can be projected to the two-dimensional space using PCA. I compare these patch-wise results to the 2D embedding of the entire shape. This comparison provides an understanding and interpretation of which surface areas lead to the patterns in the embedding space.

### TRUCK and YARIS

In the 2D visualizations of the learned features from the spectral and spatial CoSMA for the *TRUCK* components, we detect two clusters corresponding to two different deformation patterns in Figure 8.14.

The patch-based approach allows an additional comparison of the 2D patch-wise embeddings and the 2D embedding for the entire shape. I aim to identify the patches that contribute the most to the pattern visible for the entire shape. For each patch, I define a score, which equals the accuracy of an SVM (between 0.5 and 1) that classifies the observed two deformation patterns of the entire component in the patch embedding, see Figure 8.15, where a high similarity is colored in yellow.

For the spectral CoSMA, the highlighted patches correlate to the left part of the beam (part 0), where the deformation is visibly different for two different *TRUCK* simulations in Figure 8.15. On the other hand, the spatial approach highlights patches from the middle of the component, although the deformation pattern manifests towards both ends of the beam. These less significant results for the spatial CoSMA might be due to



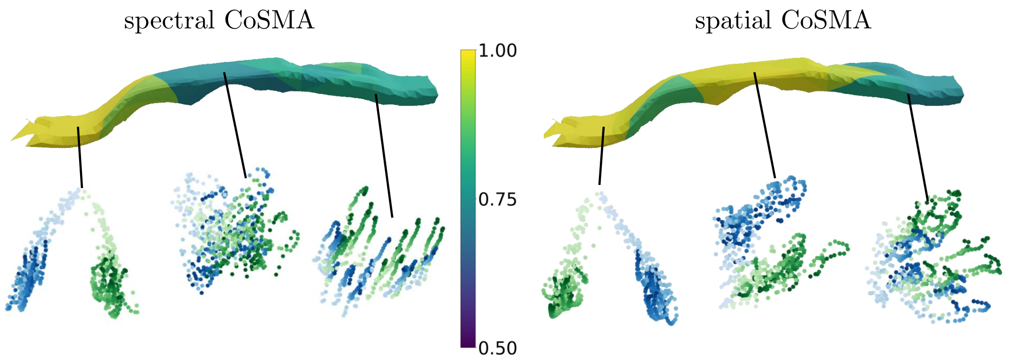


Figure 8.15: Patch-wise score for the TRUCK front beam (Part 0). Only the patches with the high scores manifest the deformation in two patterns. This is visible in the example of patch-wise embeddings with high and low scores. The embedding colors encode timestep and branch.

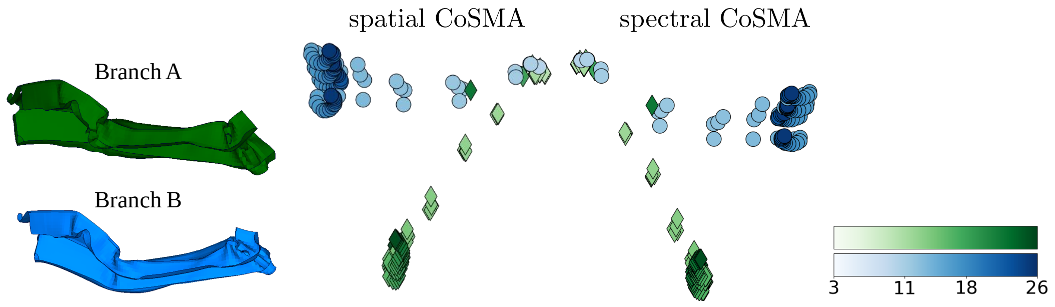


Figure 8.16: Embedding of a YARIS component and visualization of the deformation pattern. The color refers to the deformation branch and the color intensity to the simulation time.

the instability of its results, which is also visible in higher reconstruction errors.

For the *YARIS*, which the CoSMA networks have never seen during training, Figure 8.16 visualizes the low-dimensional representations for a component in 2D using PCA. For calculating the representations, I chose the *FAUST* trained model, which has the lowest reconstruction errors on the *YARIS* data. A deformation pattern is visible for the front beam that splits up the simulation set into two clusters.

## GALLOP

The time-dependent CoSMA embeddings from the *GALLOP* dataset exhibit a periodic galloping sequence, visualized in Figure 8.17, that are of similar shape for both the spatial and the spectral CoSMA. The features of the elephant are calculated by the CoSMA models trained only on horse and camel, while the elephant is excluded during training. Both CoSMAs calculate significant features of the unseen elephant that capture the periodic deformation pattern of the meshes. The flipbook starting at page 35 animates

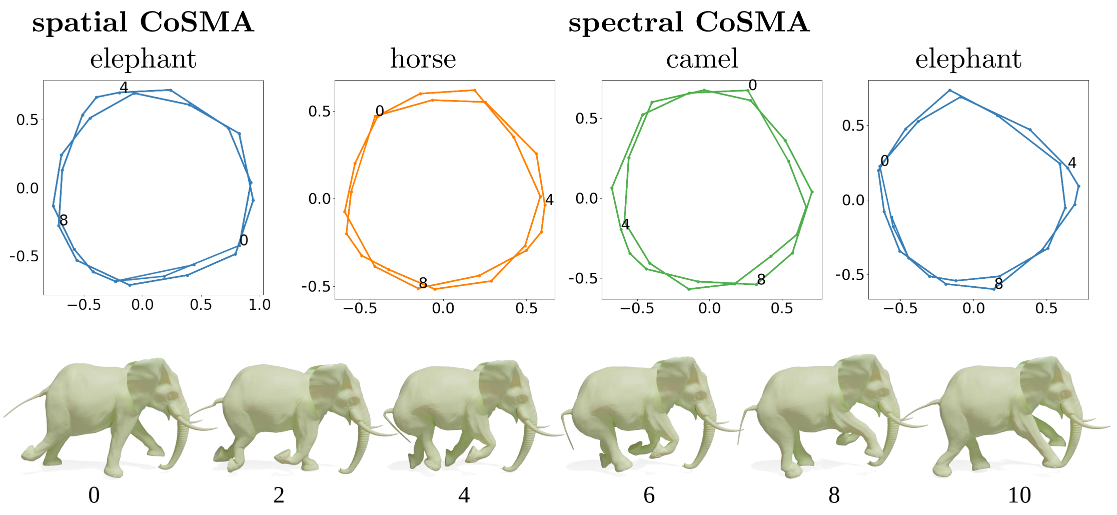
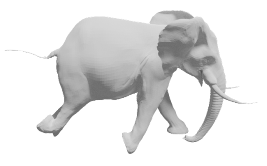


Figure 8.17: Top: 2D embeddings of the animals' low-dimensional representation from the CoSMAs over time. Timesteps are depicted in the plots.
Bottom: The elephant's pose is visualized at six equally distributed timesteps from one cyclic deformation behavior.

the galloping sequence for the elephant and makes its cyclic dynamical pattern visible.

The dataset is constructed such that the movement of the three animals aligns. Nevertheless, the positions of the timesteps in the embedding change from animal to animal because the animals do not share the same mesh connectivity and, therefore, do not share the same embedding space; only their patches do. Note how the order of the first two PCA components change and also their signs. For example, when comparing the horse-embedding to the camel-embedding, the sign of the x-axis is switched.

Again, I compare the patch-wise embedding to the embedding of the entire shape over time. Because of the poorer results on the spatial CoSMA, the comparison focuses on the spectral CoSMA for the *GALLOP* dataset. By measuring how similar the patch-wise embedding is to the component embedding, I can determine how important the deformation of the patch is for the general deformation behavior of the whole shape. The patch-wise distance (low distance in yellow) is visualized in Figure 8.18 for the spectral CoSMA learned features. To calculate this distance, I interpolate and densely subsample the lines connecting the embedding points of consecutive timesteps. Between the sampled points $p_i^s$ describing the deformation of the entire shape over time and the sampled points $p_j^p$ from the patch embedding, I calculate the chamfer distance, since the embedding shape is cyclic. Then, the distance is the lowest for circle-like patch-wise embeddings. This is the case for the body and legs, which define the elephant's gallop, whereas the movement of the trunk and head do not follow the periodic pattern.



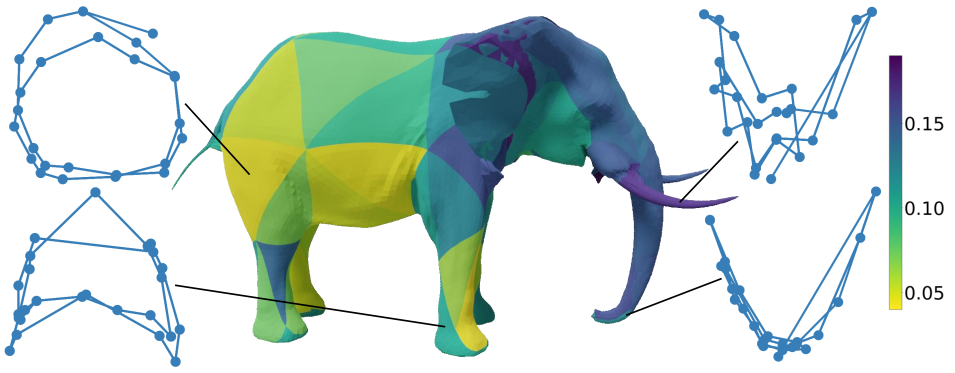


Figure 8.18: Highlighting the distance of the patch-wise embeddings to the embedding of the entire shape analyzed by the spectral CoSMA.

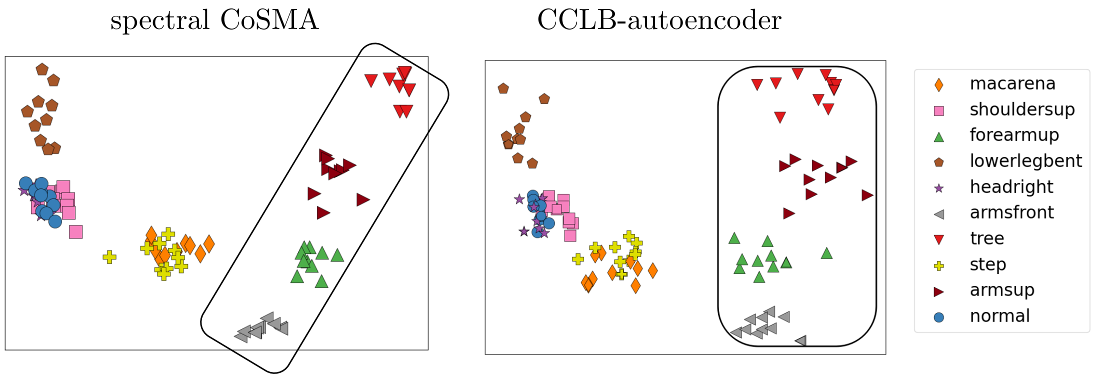


Figure 8.19: Spectral CoSMA and CCLB-autoencoder 2D-Embedding of the *FAUST* shapes in common basis. Poses marked with a triangle raise the arms; their embedded points are clustered towards the right. Figure 7.2 visualizes the ten different poses.

### FAUST and SYN

In the case of the *FAUST* dataset, several clusters form in the 2D visualization of the features learned by the spectral CoSMA in Figure 8.19 corresponding to different positions. The spectral CoSMA has been trained in the "unknown individual" setup. Note that along the horizontal axis, the position of the arms can be split into raised or not raised. Also, the positions with the least deformation from the upright standing position ("normal", "shouldersup", and "headright") are clustered together.

The deformation of the triangular meshes in the *SYN* dataset is parametrized by three variables $\alpha, \beta, \gamma \in [-1, 1]$ that describe the deformation of the corners. The three variables are approximately sampled from a rectangular area on a plane in 3D. The spectral CoSMA recovers this structure and calculates features for the samples that, when embedded in 2D, form a projection of the plane to the two-dimensional space, see Figure 8.20.

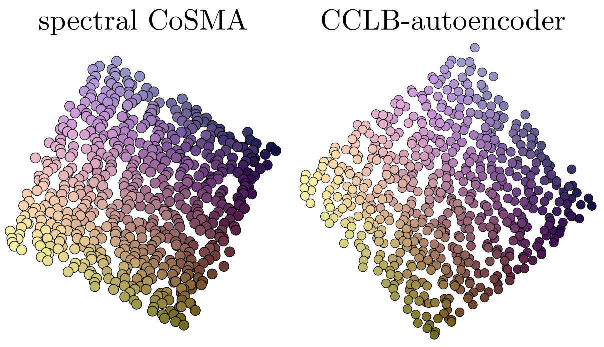


Figure 8.20: Spectral CoSMA and CCLB-autoencoder 2D-embeddings of the deformed triangles from the *SYN* dataset. The colors are sampled from a 2D colormap depending on the position of the sampled deformation parameters from the plane in 3D.

### 8.3.2 CCLB-Autoencoder

For every mesh from the collections, the CCLB-autoencoder calculates a hidden representation of size $hr = k_2 \times F$ that describes the entire shape. The shape features from the same collection can be visualized in 2D or 3D using a principal component analysis [Pea01].

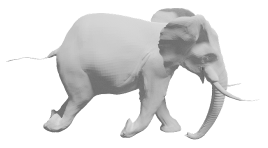
The CCLB-autoencoder allows us to embed the different shapes separately from each other and detect their deformation patterns, for example, in the case of the *FAUST* dataset, where all the meshes share the same connectivity. Compared to the CoSMAs, we can additionally visualize the features from various shapes of different connectivity in a common basis.

#### TRUCK

For the first time, different *TRUCK* components can be visualized together using the representation in the CCLB. Figure 8.21 visualizes the two left front beams of the car in three dimensions. The two deformation branches, A and B, in two different components are split along the vertical axis of the 3-dimensional embedding space. This visualizes nicely that the deformation of the two components manifests in similar deformation patterns. The features of both components align over time along a horizontal axis. Since the method is not invariant to different geometries and mesh connectivities, the two components are separated along the third axis.

#### GALLOP

The CCLB-autoencoder allows a joint visualization of camel, horse, and elephant galloping sequences from the *GALLOP* shape collection. These align over time up to translation but are still separated from each other, which captures the different shape categories, see Figure 8.21.



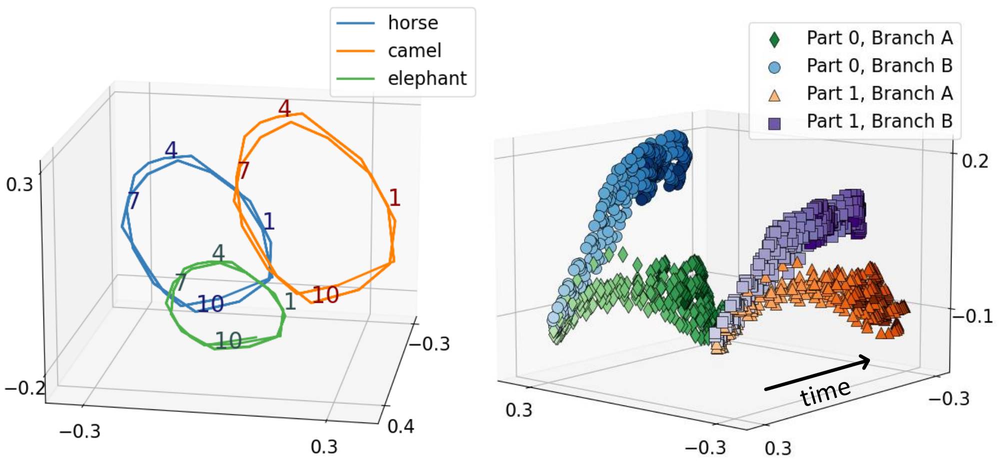


Figure 8.21: Left: Embedding of galloping sequences from the *GALLOP* dataset in common basis. Timesteps are provided in the plot.
Right: Embedding of two *TRUCK* components (left front beams), which deform in two clusters over time. The clusters of the two beams correspond to the deformation patterns highlighted in the same colors in Figure 8.14.

### FAUST and SYN

In the case of the *FAUST* dataset, several clusters form in the 2D visualization in Figure 8.19 corresponding to different positions. Note that the visualization looks similar to the embedded features calculated by the spectral CoSMA. First, in both embeddings, the position of the arms can be split into raised or not raised along the horizontal axis. Also, the same positions are clustered as in the spectral CoSMA embedding.

Like the spectral CoSMA, the CCLB-autoencoder recovers the rectangle from which the triangle deformation in the *SYN* dataset is sampled, see Figure 8.20.

## 8.4 Generative Results

I generate new shapes by sampling from the latent feature spaces of the spectral CoSMA and the CCLB-autoencoder. The quality of the generative results lets us evaluate if the shape features lie on a smooth manifold and if the networks are not overfitting to the training samples. I conduct three different generative experiments on the *FAUST* shape collection using the learned models from the “unknown individuals” setup: Interpolation of two test shapes, generation of combined positions, and feature transfer between two different bodies.

For the interpolation, I select an individual from the test set in two distinct poses. I interpolate along a straight line in the $hr$-dimensional space, pick four equally distributed points from that line, and input them to the decoders. Since the spectral CoSMA handles centered patches, the corresponding patch centers must also be interpolated. The generated shapes from both models in Figure 8.22 resemble averaged positions between

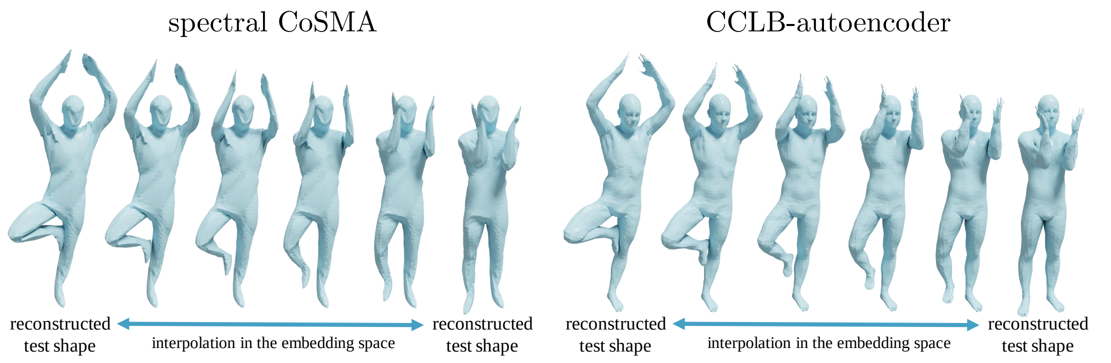


Figure 8.22: Interpolating between different *FAUST* test shapes, using the latent space representation from the spectral CoSMA and the CCLB-autoencoder.

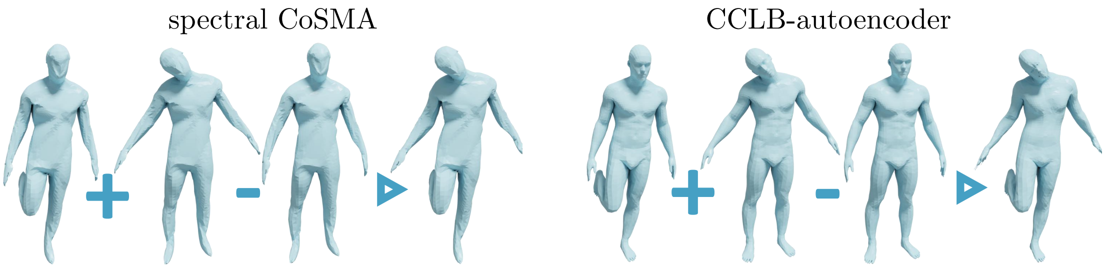


Figure 8.23: Combining two positions of *FAUST* test shapes

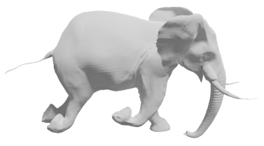
the two test samples and are of similar quality to the reconstructed meshes. The CCLB-autoencoder generates equally detailed shapes for the interpolated shape features, and arms and legs are well-formed, while the CoSMA generations slightly collapse. I apply the interpolation to the elephant galloping sequence to create the flipbook that animates it. It starts on page 35. All elephants are reconstructed by the CCLB-autoencoder, and subsequent timesteps have been interpolated in the embedding space for a smoother animation.

To blend two different shapes, I combine the low-dimensional shape features of two deformed shapes $en(S_1)$ and $en(S_2)$. To ensure that the calculated features lie on the embedding manifold and that I only transfer the deformation information, I subtract the features of the undeformed meshes in the upright position $en(S_1^0)$ and $en(S_2^0)$. Then I add $en(S_1) - en(S_1^0) + en(S_2) - en(S_2^0)$ to the shape features of the undeformed target shape $en(S_t^0)$.

When blending two distinct poses, I select one individual from the test set, add the shape features of two different poses, and input the resulting feature to the decoder

$$de\big(en(S_1) - en(S_1^0) + en(S_2) - en(S_2^0) + en(S_t^0)\big). \tag{8.2}$$

Since poses of the same individual are combined, this simplifies to $en(S_1) + en(S_2) - en(S_1^0)$. Figure 8.23 visualizes the generative results, combining a bent leg with a tilted



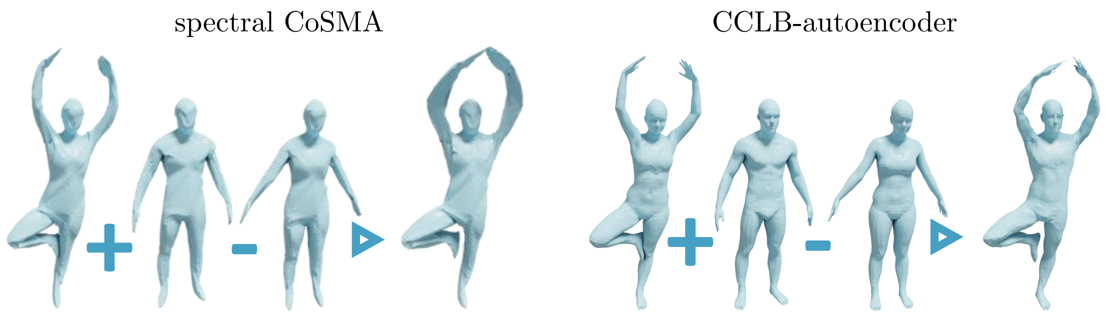


Figure 8.24: Transferring the pose from a female to a male individual from the *FAUST* dataset.

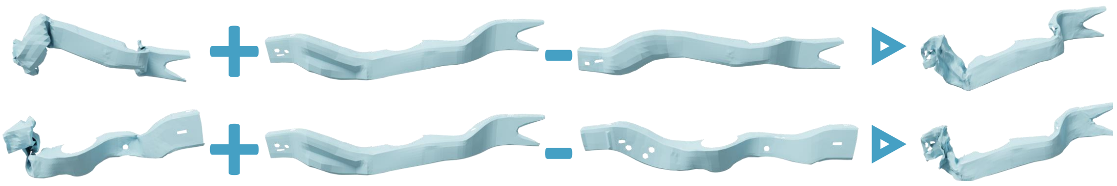


Figure 8.25: Feature transfer from one component to another from the *TRUCK* dataset using the shape features learned by the CCLB-autoencoder.

head. This position combination experiment creates shapes not found in the shape collection and shows that my models do not overfit the training poses.

To transfer features, I select a female individual and transfer her pose to a male individual, again subtracting the upright female individual shape features from the deformed ones to ensure sampling from the embedding manifold. Figure 8.24 visualizes the generated meshes. While the generated shape by the CCLB-autoencoder is smooth and the limbs are positioned correctly and naturally, the spectral CoSMA generation collapses in the arms. This might be due to the algebraic manipulation of the patch centers since this experiment involves two different individuals.

The combination of positions and feature transfer shows that the embedding spaces are smooth and allow algebraic manipulation (addition and subtraction) of shape embeddings. Nevertheless, the spectral CoSMA generations, when involving two different individuals, tend to collapse in the limbs. The generated shapes by the CCLB-autoencoder, on the other hand, are detailed, with little loss of details, well-formed, and have correctly and naturally positioned limbs.

## Feature Transfer to Different Shape

The CCLB-autoencoder represents shapes of different mesh connectivity in the same basis. Therefore, for the first time, it allows the transfer of features from shape $S_1$ to another shape $S_2$ that does not share the same mesh connectivity. I test this for the *TRUCK* dataset and transfer the deformation of one component to another one. The results are visualized in Figure 8.25. The generated deformed component is plausible when the two different components are of similar shape (upper row). It looks less realistic when the two components have a different structure. While being useful for animations in games or virtual reality, the generated deformation might not be physically correct because conservation laws apply for the deformation of components, for example, that the mass must stay constant. Nevertheless, the network was not conditioned to fulfill the conservation laws.

## 8.5 SubdivNet Autoencoder Architecture

I compare my surface mesh autoencoders to an additional baseline model that learns hierarchical features for semi-regular surface meshes using convolutional layers, see section 8.1. To this end, the CoSMA architecture is translated to the SubdivNet baseline by replacing the convolutional layers with the subdivision-based mesh convolutions and the corresponding pooling and unpooling operations introduced in [HLG+22], see Table 8.11. All SubdivNet convolutions use stride and dilation equal to one and kernel size equal to three and are followed by ReLU activations. As the SubdivNet convolutions operate on face features rather than vertex features, we use the coordinates of the three adjacent vertices per face as input features. The bullets • reference the corresponding batch size. The second dimension is the number of features, and the last dimension is the number of faces of the current mesh.

**Subdivnet Autoencoder**

| Encoder Layer | Output Shape | Param. | Decoder Layer | Output Shape | Param. |
|---|---|---|---|---|---|
| Input | $(\bullet, 9, 25600)$ | 0 | Fully Conn. | $(\bullet, 6, 1600)$ | 9,840,000 |
| SubdivConv | $(\bullet, 16, 25600)$ | 592 | SubdivUnpool | $(\bullet, 6, 6400)$ | 0 |
| SubdivPool | $(\bullet, 16, 6400)$ | 0 | SubdivConv | $(\bullet, 32, 6400)$ | 800 |
| SubdivConv | $(\bullet, 6, 6400)$ | 390 | SubdivUnpool | $(\bullet, 32, 25600)$ | 0 |
| SubdivPool | $(\bullet, 6, 1600)$ | 0 | SubdivConv | $(\bullet, 16, 25600)$ | 2,064 |
| Fully Conn. | $(\bullet, 1024)$ | 9,831,424 | SubdivConv | $(\bullet, 9, 25600)$ | 585 |

Table 8.11: Structure of the SubdivNet [HLG+22] autoencoder using Mesh Convolution, Mesh Pooling, and Mesh Unpooling for meshes with subdivision connectivity.

## 8.6 Summary

This chapter compares the proposed surface mesh autoencoder architectures to each other based on the quality of reconstructed unseen meshes, the learned low-dimensional features, and the generation by decoding manipulated low-dimensional shape features.

On one hand, I propose the spatial and spectral CoSMA that handle padded patches of the surface meshes. To this end, all the surface meshes have been remeshed to semi-regular mesh connectivity. Since the patches have the same regular mesh connectivity, I train one CoSMA network that handles all the patches and learns hierarchical features

using pooling. On the other hand, the CCLB-autoencoder handles entire meshes and projects the mesh features to a common spectral basis using the proposed spectral mesh pooling. This requires a functional map network that connects the shapes to each other by functional maps or p2p correspondences. The CCLB-autoencoder handling entire meshes has a slower convergence and a 30 times higher number of trainable parameters than the CoSMA models, which apply the lighter patch-based approach.

In most cases, the reconstruction quality of the CCLB-autoencoder is higher than the CoSMA reconstruction quality. The spectral CoSMA always has lower reconstruction errors than the spatial CoSMA, except for one exception in Table 8.2.

The cases when the spectral and spatial CoSMA reconstructions are better than the CCLB-autoencoder are of special interest. In the "unknown poses“ experiment setup on the *FAUST* dataset, possibly in combination with the *SCAPE* dataset, test poses are not included in the training set. Here, the CCLB-autoencoder generalizes less successfully, while the reconstruction quality of the spectral CoSMA is similar to including the poses in the training set. The out-of-distribution experiments highlight this generalization capability of the spectral CoSMA models. Here, I successfully apply a learned spectral CoSMA to a different dataset. The patch deformation seems well-parametrized in the CoSMA embedding space since the network reconstructs unknown shapes. This is only possible because of the patch-based approach. Since the CCLB-autoencoder requires correspondence maps between the shapes, it cannot be applied to shapes without any natural correspondence.

When different shapes are of similar pose to the training shapes, for example, in the *GALLOP* dataset, the CCLB-autoencoder reconstructs small details as the ears, even though the p2p maps are erroneous, since the three animals have no perfect correspondence. The spectral basis seems to capture the general pose of the animals, see Figure 2.6 in the related work on surfaces. Since the dimensionality of the CCLB is low, the errors in the p2p maps are not considered. The details are taken from the template mesh used for reconstruction and the vertex-wise features learned by DiffusionNet. Even if the p2p maps are learned unsupervised, leading to more mapping errors, the reconstructions are of reasonable quality because of the effective regularization of the loss.

Both networks learn significant low-dimensional features for the training and test shapes that allow the detection of deformation patterns. Interestingly, the 2D plots of the learned features look very similar, for example, in Figure 8.19 for *FAUST*, albeit applying different approaches. The CoSMA networks calculate patch-wise features, whose comparison to the shape-wise features allows the localization of deformation patterns on the surfaces. For the first time, the CCLB-autoencoder calculates surface mesh features in one embedding space for different mesh connectivities without applying global pooling. This allows the comparison of shape features in one space, for example, different components from the *TRUCK* datasets.

The stable spectral mesh pooling and significant learned features lead to smooth and well-formed generated shapes using the CCLB-autoencoder. Although the generated shapes using the learned spectral CoSMA have noticeable patch boundaries, the patch deformations are well interpolated. Both approaches can generate shapes that are not in the training set.

# 9 Additional Experiments

We conduct additional experiments to evaluate, on the one hand, the patch-based approach to learning hierarchical features and, on the other hand, to test time series prediction in the joint embedding space of the CCLB-autoencoder. For the additional experiment in section 9.1, we build a patch-based mesh segmentation model based on the spectral CoSMA. These segmentation results are compared to baseline approaches on the human body part segmentation experiment.

Section 9.2 evaluates whether the common embedding space of the CCLB-autoencoder allows the joint analysis of several shapes of different connectivity. Since generative experiments have shown that the joint embedding space is smooth and allows interpolation, we test the prediction of the time series defined by the shape features in the joint embedding space of the *TRUCK* dataset. The CCLB-decoder reconstructs the predicted mesh features.

## 9.1 Mesh Segmentation

The CoSMA autoencoding architecture learns features at various levels of detail, which means features describing fine and coarser shape structures because of the mesh downsampling in the encoder. We want to test the network structure and the patch-wise approach handling semi-regular mesh representations in a different experimental setup, which requires coarser and finer features and a pipeline that handles meshes of different sizes and connectivity. This is why we chose to test the CoSMA pipeline on shape segmentation. Additionally, the spectral CoSMA autoencoder generalized well for unseen meshes in several out-of-distribution experiments, see section 8.2.2. Therefore, we chose the human body part segmentation experiment, where a label is assigned to every face of the surface mesh that corresponds to one of eight human body parts. The experiment is conducted on five human datasets, see Figure 7.4 in section 7.1.2, where the *SHREC* dataset is only used for testing. Therefore, the learned method must generalize well to unseen meshes of unknown mesh connectivity. A network is trained by supervised learning to predict the face-wise probability of belonging to a particular body part. This prediction generally needs global information about the location of the vertex or face in the entire mesh and about local mesh features.

All recent architectures aim to solve the mesh segmentation task by applying mesh convolutional layers. The networks handle entire meshes and apply the filters on a local mesh structure. [SS21, LT20, SACO22] calculate vertices-wise features, [HLG$^+$22] faces-wise, [HHF$^+$19] edge-wise features, and [MLR$^+$20] a combination of all of them. [HSBH$^+$19, MGA$^+$17] handle a 2D parametrization of the mesh. The networks concatenate multiple convolutional layers to incorporate non-local information in the learned fea-

tures. Additionally, some networks provide global information via pre-computed shape features, for example, the shape descriptors heat kernel signature (hks) or wave kernel signature (wks) [SACO22, HSBH+19, MGA+17], see section 2.5. Most networks do not apply pooling to the local features or downsample the meshes. They operate only on the local structures, which makes it possible to handle meshes of diverse connectivities. MeshCNN [HHF+19] and PD-MeshNet [MLR+20] implement a mesh-dependent down and up-sampling, and SubdivNet [HLG+22] handles semi-regular meshes and applies a pooling operation similar to mine.

### 9.1.1 Method

The CoSMA architectures combine concatenated convolutional and pooling layers to calculate features capturing a different level of detail. Since the spectral CoSMA generalizes well, we only apply spectral convolutions in the mesh segmentation experiments. To make use of the features learned at different resolutions, skip connections are added to the CoSMA network, similar to [MLR+20, HHF+19]. [RFB15] proposed implementing skip connections for image segmentation in their U-Net architecture. The skip connections resemble shortcuts between layers in the encoder to the corresponding decoder layers, skipping the hidden layer of lowest dimensionality and providing more data for the label prediction than only the low-dimensional latent representation. Additionally, we increase the size of the two convolutional layers to 64 and 128 output channels and calculate a latent representation of $hr = 256$. The resulting CoSMA network for segmentation has 1,324,680 learnable parameters.

Similarly to baseline approaches [SACO22, HSBH+19], we input the 3D coordinates describing the vertex positions and provide additional precalculated vertex-wise features that capture global information. To the vertex-wise 3D coordinates, we concatenate the vertex-wise hks in $\mathbb{R}^{n\times t}$ [SOG09] and the wks in $\mathbb{R}^{n\times t}$ [ASC11] descriptors for $t = 20$ timesteps, see section 2.5.

Also, every mesh from the five datasets is remeshed separately to a semi-regular mesh representation. This way, the space of deformed patches is more diverse, which prevents the network from overfitting to a fixed semi-regular mesh representation. At the same time, we test the stability of the CoSMA segmentation network to diverse remeshing results.

Figure 9.1 visualizes the human body mesh segmentation experiment pipeline.

### 9.1.2 Experiment Setup

The hks and wks descriptors are calculated on the irregular meshes. The signatures are not scale invariant, so they are calculated on normalized meshes with unit area, similar to Figure 2.7. We use the parametrization calculated during the semi-regular remeshing to project the vertex-wise descriptors to the semi-regular mesh. The signatures are concatenated to the vertex-wise 3D coordinates before input into the network.

The four training human body datasets containing 381 shapes are split into train and validation set (75% to 25% per dataset). The *SHREC* dataset containing 18 meshes is

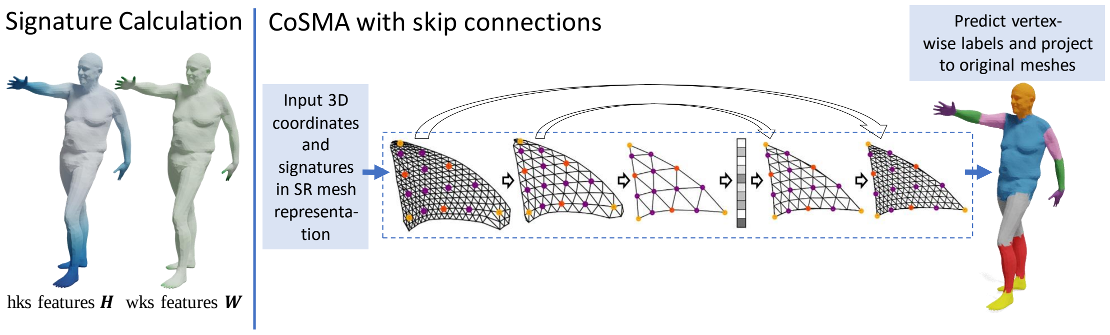

Signature Calculation
hks features H
wks features W
CoSMA with skip connections
Input 3D coordinates and signatures in SR mesh representation
Predict vertex-wise labels and project to original meshes

Figure 9.1: Pipeline for human body mesh segmentation using a CoSMA inspired patch-wise segmentation network.

only used for testing. All semi-regular meshes are aligned along the vertical axis and normalized into the range $[-1, 1]$.

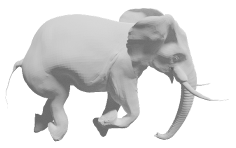
We train the spectral CoSMA segmentation network with the adaptive learning rate optimization algorithm [LH19] using a learning rate of 0.0001 and patch-wise cross-entropy as the loss function, see section 3.1.2, with class weights to handle the unbalanced labels. Following SubdivNet [HLG+22], the patches are augmented by randomly rotating the patch 3D coordinates around the vertical axis by $\tau \in \{0°, 90°, 180°, 270°\}$ and scaling each axis of a patch by a factor sampled from $\mathcal{N}(1, 0.1)$.

After training, we evaluate the performance of the models on the original irregularly meshed meshes. To this end, we first reassemble a semi-regular mesh from the labels of the unpadded patches. In case of overlap at the patch boundaries, we sum the outputs from neighboring patches for the same vertex and rescale to one. The predicted vertex label on the semi-regular mesh is the class with the highest weight. Then, we determine the face labels on the original irregular meshes from the vertex-wise predicted labels on the semi-regular meshes. For this, the label of the node closest to the face centroid is chosen.

Two metrics measure the fidelity of the predicted face labels $\hat{L}_i$ for mesh $\mathcal{M}_i$. The ground truth labels are referred to as $L_i$. The prediction accuracy $acc(L_i, \hat{L}_i)$ for each mesh in the datasets is the number of correctly classified faces divided by the number of faces

$$acc(L_i, \hat{L}_i) = \frac{|\{(L_i)_j = (\hat{L}_i)_j \mid j = 1, \dots, |\mathcal{F}|\}|}{|L_i|}. \tag{9.1}$$

Since the classes are not represented equally, for example, there are more faces labeled "torso" than "hand", we also calculate the mean Intersection-over-Union (mIoU) $mIoU(L_i, \hat{L}_i)$, which is a widely used metric for segmentation or object detection tasks. Per class, it divides the correctly labeled faces by the number of true positive, false



| Method | Test accuracy | mIoU |
|---|---|---|
| Pointnet++ [QYSG17] | 82.3 % | 74.7 % |
| PD-MeshNet [MLR+20] | 86.9 % | 86.9 % |
| MeshCNN [HHF+19] | 87.7 % | 92.3 % |
| SNGC [HSBH+19] | 91.3 % | 91.0 % |
| DiffusionNet [SACO22] | 91.5 % | |
| SubdivNet [HLG+22] | **93.0 %** | 93.0 % |
| MeshFormer [Won23] | | **94.2 %** |
| CoSMA | $87.0 \pm 0.7\%$ | $78.4 \pm 0.9\%$ |

Table 9.1: Mesh segmentation accuracy on the human body dataset by several surface mesh convolutional networks. For the CoSMA network, the average and standard deviation over three training runs are provided. Baseline accuracies from [HLG+22] and mIoU from [Won23]. DiffusionNet accuracy from [SACO22].

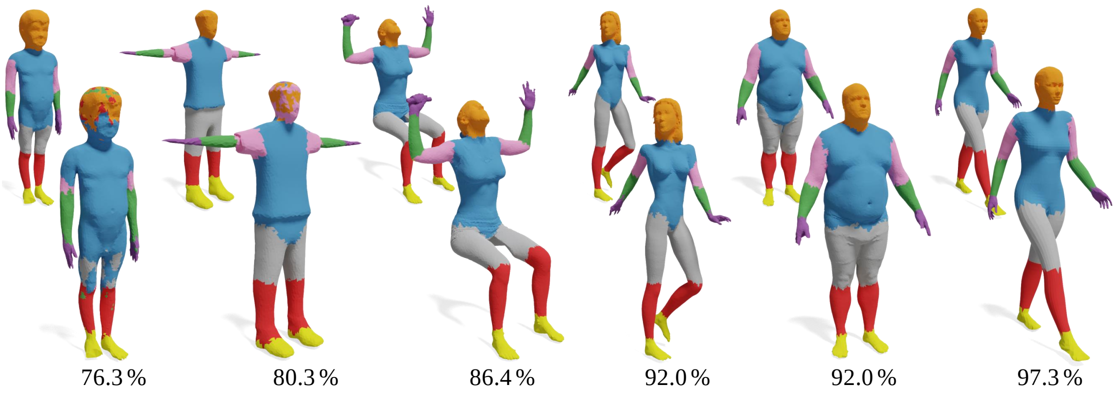


Figure 9.2: Test meshes from the *SHREC* dataset with highlighted predicted and true (small shapes) face labels and mesh-wise segmentation accuracy.

positive, and false negative labeled faces:

$$\begin{aligned} mIoU(L_i, \hat{L}_i) =& \frac{1}{\#(\text{classes})} \sum_{\text{class } c} IoU_c(L_i, \hat{L}_i) \\ =& \frac{1}{\#(\text{classes})} \sum_{\text{class } c} \frac{|\{L_i = \hat{L}_i = c\}|}{|\{L_i = c\}| + |\{\hat{L}_i = c\}| - |\{L_i = \hat{L}_i = c\}|}. \end{aligned} \tag{9.2}$$

## 9.1.3 Experiment Evaluation

The test accuracies and mIoU of the CoSMA segmentation network and baseline models are listed in Table 9.1 and predicted labels on test meshes in Figure 9.2. The CoSMA test accuracy is comparable to results from 2020 [MLR+20]. The mIoU of baseline

| Dataset | Validation accuracy | mIoU | Top-2 accuracy |
|---|---|---|---|
| *FAUST* | $96.4 \pm 0.2$ % | $91.6 \pm 0.5$ % | $99.9 \pm 0.04$ % |
| *MIT* | $94.1 \pm 0.1$ % | $89.5 \pm 0.3$ % | $99.9 \pm 0.04$ % |
| *ADOBE* | $94.0 \pm 1.0$ % | $87.5 \pm 1.3$ % | $99.6 \pm 0.2$ % |
| *SCAPE* | $95.6 \pm 0.1$ % | $91.6 \pm 0.1$ % | $99.9 \pm 0.1$ % |
| *SHREC* | $87.0 \pm 0.7$ % | $78.4 \pm 0.9$ % | $97.9 \pm 0.5$ % |

Table 9.2: CoSMA mesh segmentation accuracy, mIoU, and top-2 accuracy on test and validation sets from the human body dataset. Average values and standard deviation over three training runs.

| | | predicted labels | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | head | hand | forearm | upper arm | torso | upper leg | lower leg | feet |
| true labels | head | 88% | | | 3% | 9% | | | |
| | hand | | 92% | 6% | | 1% | | | |
| | forearm | | 5% | 90% | 4% | | | | |
| | upper arm | 2% | | 7% | 75% | 16% | | | |
| | torso | 1% | | | 1% | 90% | 7% | | |
| | upper leg | 1% | 1% | | | 14% | 79% | 5% | |
| | lower leg | 1% | 1% | | | | 4% | 90% | 3% |
| | feet | | | | | | | 10% | 89% |

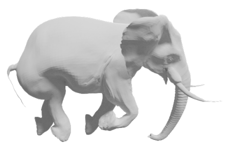
Table 9.3: CoSMA mesh segmentation confusion matrix for the test set predictions normalized by true labels.

networks handling meshes cannot be improved. The CoSMA mIoU is only higher than the Pointnet++ results handling point clouds.

Nevertheless, the validation accuracies are comparable to the baseline methods, as Table 9.2 indicates. This means that the CoSMA-inspired network does not generalize well on the test dataset, which contains meshes from a different dataset (*SHREC*).

The most misclassifications are due to the difficulties of the CoSMA segmentation network to distinguish body parts that are next to each other, like forearm and hand, as well as forearm and upper arm, as the confusion matrix in Table 9.3 indicates. This is also visible in Figure 9.2 at the knees and hips. The high top-2 accuracies in Table 9.2, which counts a prediction as correct if the true label is one of the two labels with the highest predicted probabilities, reflects this observation.

The ablation study of the CoSMA segmentation network in Table 9.4 indicates that the performance of the network depends highly on the surface descriptors. On the other hand, reducing the model size or omitting 3D coordinates as input features only slightly deteriorates the results. This is because the signatures provide information on the vertex positions in the context of the entire surface mesh that the 3D coordinates on the patch and the patch padding cannot provide. This information is crucial for a segmentation network handling only patches of the entire mesh and allows high-quality segmentation results on the validation meshes.



| Input features | Test accuracy |
|---|---|
| Only 3D coordinates | 78.6 % |
| Only signatures | 86.3 % |
| Small model | 84.4 % |
| 3D coordinates and signatures, full model | **87.0 %** |

Table 9.4: Ablation Study: CoSMA mesh segmentation test accuracies when inputting only 3D coordinates or hks and wks shape descriptors or reducing the model size.

Nevertheless, the hks and wks depend on the mesh connectivity and change under non-isometric deformation (deformations that change the geodesic distances on the surface), see Figure 2.7 in section 2.5. Since several meshes from the test dataset are structurally different, for example, the two shapes on the left of Figure 9.2, the signatures might not be directly comparable to the ones of the train shapes. Therefore, they do not provide comparable global information about the test meshes to segment successfully with a patch-wise approach, and the network does not generalize well.

## 9.2 Time Series Prediction

The deforming car components from the *TRUCK* dataset manifest similar deformation behavior over time. This is visible in the joint 3D embedding of the learned features by the CCLB-autoencoder, see Figure 8.21, where one of the 3D embedding dimensions is highly correlated with the simulation time.

Also, for the first time, multiple components are represented in one embedding space and the same basis, enabling a joint analysis of multiple components. Besides this, the representation in the low-dimensional embedding space reduces spatial redundancies in the data. We do not need to handle surface meshes that are represented by a high number of 3D vertex coordinates combined with a set of faces or edges. This information is encoded in the low-dimensional representation.

The *TRUCK* dataset stems from a design of experiment study, where multiple simulations are calculated for different model parameters. The goal of the experiment is to understand the effect of the model parameters on the simulation output and to select the combination of parameters such that the model and simulation behavior fulfill specific design goals. This is a time-consuming process, but the in-situ analysis of simulations, which means analyzing the simulation after a few initial timesteps $t_1, \dots, t_s$, can speed it up. If undesirable behavior is detected or predicted after the first $s$ of a total of $e$ timesteps, a new simulation can already be started with adapted model parameters or the ongoing simulation can be interrupted. For this decision, a good prediction of the simulation results and the deformation of the components in the following timesteps can be calculated. Since surface meshes over time contain many redundancies, their deformation is often predicted using low-dimensional data representations

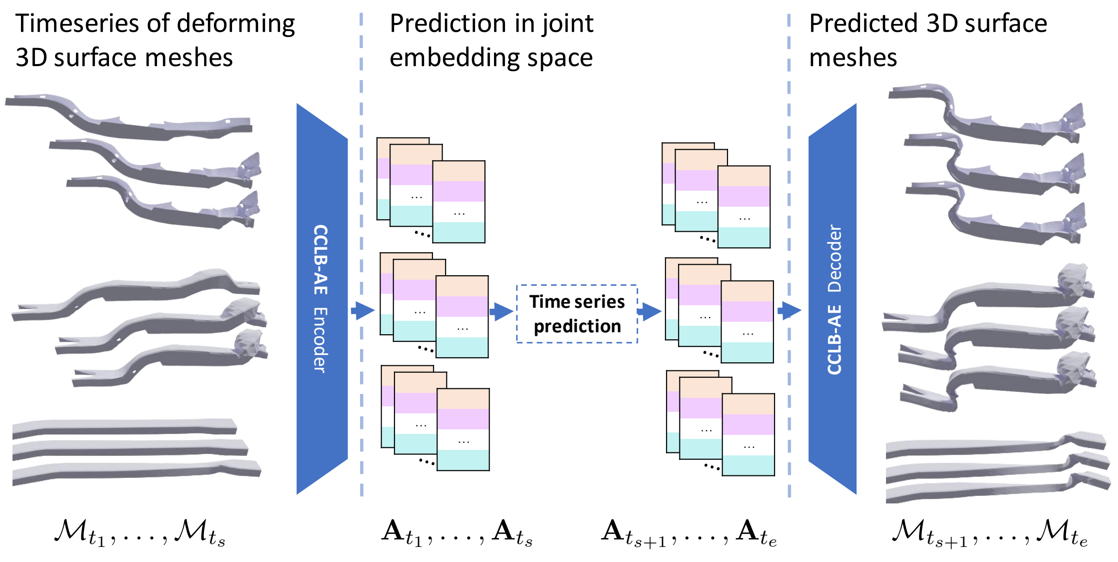

Timeseries of deforming 3D surface meshes
Prediction in joint embedding space
Predicted 3D surface meshes
CCLB-AE Encoder
Time series prediction
CCLB-AE Decoder
$\mathcal{M}_{t_1}, \dots, \mathcal{M}_{t_s}$
$\mathbf{A}_{t_1}, \dots, \mathbf{A}_{t_s}$
$\mathbf{A}_{t_{s+1}}, \dots, \mathbf{A}_{t_e}$
$\mathcal{M}_{t_{s+1}}, \dots, \mathcal{M}_{t_e}$

Figure 9.3: Prediction pipeline: The first $s$ timesteps of the sequences of deforming components are encoded in the joint CCLB-autoencoder embedding space. In the low-dimensional space, a learned predictor predicts the timesteps $t_{s+1}, \dots, t_e$. The predicted shape features are reconstructed to the mesh representation using the decoder.

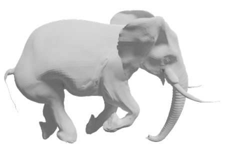
[DSC18, HITG20, KGE+21].

The CCLB-embedding space for different components not only calculates significant low-dimensional shape features but also allows for a joint analysis of multiple components that manifest similar deformation behavior yet are represented by different surface meshes. Also, the generative experiments with the CCLB-autoencoder have shown that the learned embedding manifold of the CCLB-autoencoder is smooth and that the decoder $de_{CCLB}$ robustly reconstructs interpolated or algebraicly manipulated features. Therefore, we decide to predict the remaining timesteps $t_{s+1}, t_{s+2}, \dots, t_e$ of the simulation in the low-dimensional feature space, where the data has fewer redundancies and where a feature vector compactly represents every mesh. Then, we can predict the surface mesh deformation by applying our decoder $de$ to the predicted shape features. Figure 9.3 visualizes this pipeline.

### Notation

For this experiment, we consider the *TRUCK* dataset as a set of sequences and not only a collection of shapes; hence, we change the notation to describe the timestep. $\mathcal{M}_{i,t}(V_{i,t}, \mathcal{E}_i)$ denotes the mesh from the $i$-th simlution at timestep $t$. The mesh connectivity stays the same over time. Applying the trained CCLB-autoencoder, the mesh is compactly represented by a flattened $hr$-dimensional feature vector ($hr = k_2 \cdot F$)

$$\mathbf{A}_{i,t}^{CCLB} = en_{CCLB}(V_{i,t}) \qquad \in \mathbb{R}^{hr}. \tag{9.3}$$



Since we only use representations in the CCLB, we omit the subscript $\bullet^{CCLB}$.

### 9.2.1 Methods

We test different approaches to predict the shape features $\mathbf{A}_{i,t_{s+1}}, \dots, \mathbf{A}_{i,t_e}$ at the remaining timesteps, given the shape features of the first $s$ timesteps of the sequences $\mathbf{A}_{i,t_1}, \dots, \mathbf{A}_{i,t_s}$. Section 3.4 gives a broad overview of machine learning approaches to learning the dynamics of simulations. Nevertheless, since the CCLB-autoencoder provides significant low-dimensional features and due to the rather small size of the *TRUCK* dataset containing six components and 32 simulations, this additional experiment mainly focuses on deterministic instead of machine learning methods for predicting the simulation.

#### $k$-Nearest Neighbor

As a simple baseline, we apply a $k$-nearest neighbor (kNN) search in the set of all training time series. For predicting the timesteps $t_{s+1}, \dots, t_e$ of the test simulation, we average the low-dimensional representations of the $k$ nearest neighbors in the set of all training simulations up to timestep $t_s$.

#### Dynamic mode decomposition

The dynamic mode decomposition (DMD) is a dimension reduction algorithm for time series data, originally presented to extract and analyze dynamical features from flow data [Sch10, Sch22]. The method (in a discrete-time setting) assumes a time series of observables describing the state of a dynamical system. In our case, we first consider $\mathbf{A}_{i,t_j}$ and $\mathbf{A}_{i,t_{j+1}}$ for all simulations $i$ in the train set and $j = 1, \dots, e-1$ as the observables from the underlying dynamical system. We define two matrices with the shape features shifted in times and for all train simulations in their columns

$$X \hat{=} [\mathbf{A}_{i,t_j} \dots]_{\substack{j=1,\dots,e-1 \\ i=1,2,\dots}} \qquad Y \hat{=} [\mathbf{A}_{i,t_j} \dots]_{\substack{j=2,\dots,e \\ i=1,2,\dots}} \tag{9.4}$$

and want to approximate the matrix $U$, describing the temporal correlations for the training simulations, by solving the potentially over- or underconstrained problem

$$Y = UX. \tag{9.5}$$

The DMD (for example, algorithm 2 for the exact DMD from [HWM+14]) estimates $U \hat{=} Y X^\dagger$ and, at the same time, calculates the eigenmodes of the dynamical system. Since the optimization may be inefficient for high data dimensions or numbers of samples, the exact DMD algorithm circumvents an explicit representation or direct manipulation of $U$. It takes advantage of a reduced decomposition using only the $r_{DMD}$ most important eigenmodes. The eigenmodes, together with the corresponding eigenvalues, can be used to analyze the dynamics of the system [Mez05, RMB+09, Sch10, KBBP16, Sch22]. We, on the other hand, are interested in the matrix $U$ or its description by the most important

GT , knn, dmd, dmd + delay, nn + delay
s=12
Part 000, (oben 018, unten 001)

—

GT , knn, dmd, dmd + delay, nn + delay
s=10
Part 000, (oben 018, unten 001)

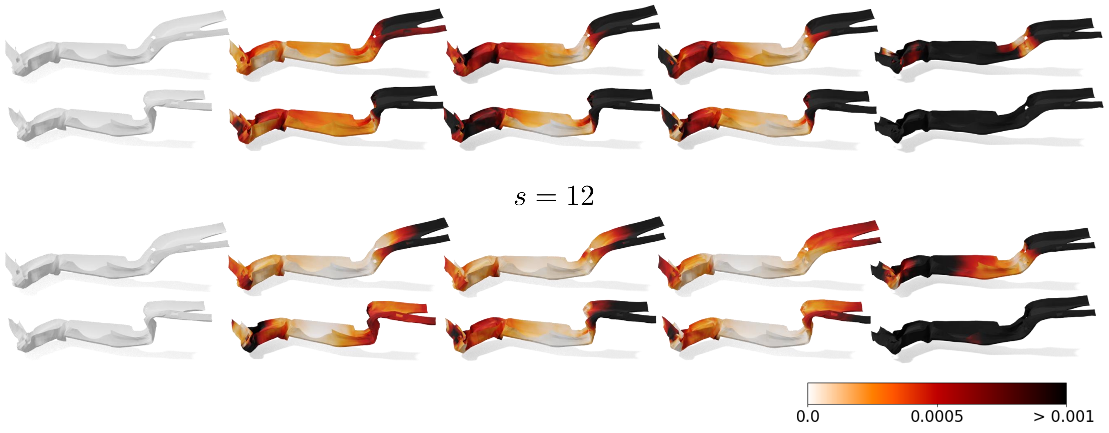


Figure 9.4: Reconstruction of predicted shape features using the CCLB-autoencoder. Vertex-wise prediction error to true mesh is highlighted.

eigenfunctions since it allows the prediction of the remaining timesteps $t_{s+1}, \dots, t_e$, given $\mathbf{A}_{i,t_s}$

$$\mathbf{A}_{pred,t_{s+k}} = U^k \mathbf{A}_{t_s}. \tag{9.6}$$

## DMD & Delay Embedding

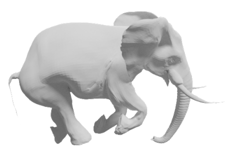
Figure 8.21 shows that the observables $\mathbf{A}_{i,t_j}$ embedded in 3D do not evolve linearly, which suggests that equation (9.5) is illposed. Nevertheless, Koopman theory [Koo31, KN32] states that nonlinear dynamical systems can be reproduced by a linear evolution in a space of observables on the state variables

$$g(\mathbf{A}_{t+1}) = U g(\mathbf{A}_t), \tag{9.7}$$

where $g$ is the mapping from the state space to the space of observables. Given the mapping to a space of observables, DMD algorithms often approximate the Koopman Operator that describes this linear evolution [RMB+09, Sch10, HWM+14, KBBP16].

Nevertheless, the approach requires finding a mapping $g$ from the observed measurement series into a space where the dynamics are linear. Various methods have been taken into account to find the mapping, including kernel methods [Gia19], deep learning of coordinate transformations (see section 3.4), and time-delay embeddings [BBP+17, AM17, KKBK20, YZZ+21, DKBK22].

The time-delay embedding is a coordinate transformation $g : \mathbb{R}^{hr} \to \mathbb{R}^{hr \cdot (d)}$ that concatenates $d$ copies of an observation that are shifted in time at $t - j \cdot \tau$ for $j =$



$0, \ldots, d-1$

$$g(\mathbf{A}_t) = \begin{bmatrix} \mathbf{A}_t \\ \mathbf{A}_{t-1\cdot\tau} \\ \vdots \\ \mathbf{A}_{t-(d-1)\cdot\tau} \end{bmatrix}. \tag{9.8}$$

Hence, the time-delay embedding as the space of Koopman observables allows considering relations between more than two consecutive timesteps. This technique is motivated by the influential Takens embedding theorem, which demonstrates that under certain conditions, a chaotic attractor can be reconstructed up to a diffeomorphism from a time series of observations of the state of the dynamical systems [Tak81]. We refer to this approach of using Koopman observables from the time-delay embedding and approximating the Koopman operator using the DMD as DMD & delay embedding.

#### NN & Delay Embedding

To evaluate the power of the DMD to approximate the Koopman operator when using the time-delay embedding as the space of Koopman observables, we compare it to learning a nonlinear function $u_{NN} : \mathbb{R}^{hr*(d)} \to \mathbb{R}^{hr*(d)}$ that predict the following observable

$$g(\mathbf{A}_{t+1}) = u_{NN}(g(\mathbf{A}_t)), \tag{9.9}$$

by a two-layer fully connected NN with one hidden layer of size $r_{NN}$.

### 9.2.2 Experiment Setup

The six different components lie in the same embedding space. Therefore, the approaches handle the six different components together. To reduce the differences in the embedding that are due to the different geometries of the components, as visible in the 2D embedding of several *TRUCK* components in Figure 8.21, we translate every time series by $-\mathbf{A}_{i,t_1}$ to start at zero. We run experiments for $s = 10$ and $s = 12$ and predict the last $e - s$ timesteps of the simulations. For the *TRUCK* simulations, we consider $e = 29$ timesteps in this experiment. Therefore, similarly to [HITG20], we input approximately one-third of the time series to the predictor. The same 30% of the simulations are provided for training and optimizing the predictors as for training the CCLB-autoencoder.

For the $k$-nearest neighbor (kNN) baseline, it always holds $k = 3$. The chosen parameters for the DMD embedding dimension $r_{DMD}$, the parameters $\tau$ and $d$ of the delay embedding, and the size of the hidden layer $r_{NN}$ in the NN are provided together with the results.

For the loss calculation, we apply the same error as for the evaluation of the reconstruction experiments, see (8.1) in section 8.2. We sum up the squared Euclidean distance between the true vertex coordinates $V$ and the reconstructed predicted shape features $V_{pred} = de_{CCLB}(\mathbf{A}_{pred})$

$$\mathrm{RecErr}(\mathcal{M}(V, \mathcal{E}), \mathcal{M}_{pred}(V_{pred}, \mathcal{E})). \tag{9.10}$$

| Method | Parameters | $s$ | Error between predicted mesh coordinates $V_{pred}$ to true $V$ | reconstr. $V_{rec}$ |
|---|---|---|---|---|
| CCLB-AE | | | 0.03 | 0 |
| kNN | $k = 3$ | 10 | 0.77 | 0.72 |
| DMD | $r_{DMD} = 34$ | 10 | 0.82 | 0.77 |
| DMD & delay embedding | $r_{DMD} = 125, \tau = 2, d = 5$ | 10 | **0.58** | 0.53 |
| NN & delay embedding | $r_{NN} = 100, \tau = 2, d = 5$ | 10 | 6.88 ± 0.3 | 6.80 ± 0.3 |
| kNN | $k = 3$ | 12 | 0.66 | 0.61 |
| DMD | $r_{DMD} = 85$ | 12 | 0.63 | 0.58 |
| DMD & delay embedding | $r_{DMD} = 70, \tau = 3, d = 4$ | 12 | **0.33** | 0.29 |
| NN & delay embedding | $r_{NN} = 130, \tau = 3, d = 4$ | 12 | 4.61 ± 0.3 | 4.54 ± 0.3 |
| | **Separate components** | | | |
| DMD & delay embedding | $r_{DMD} = 62, \tau = 2, d = 5$ | 10 | 0.54 | 0.48 |
| DMD & delay embedding | $r_{DMD} = 35, \tau = 3, d = 4$ | 12 | 0.33 | 0.28 |

Table 9.5: Errors for the predicted meshes from the *TRUCK* dataset, providing $s = 10$ or $s = 12$ input timesteps and shape representations in the common embedding space from the CCLB-autoencoder experiments. For the best method, we also provide results when considering the components separately. For the learned method, the mean and std of three training runs are provided.

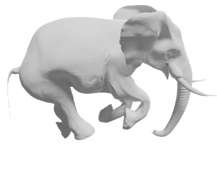
Since the reconstruction is imperfect, the upper error cannot be zero. Therefore, we also provide the error comparing the reconstructed predicted shape features $V_{pred}$ to the reconstructed true shape features $V_{rec} = de_{CCLB}(en_{CCLB}(V))$

$$\text{RecErr}(\mathcal{M}_{rec}(V_{rec}, \mathcal{E}), \mathcal{M}_{pred}(V_{pred}, \mathcal{E})). \qquad (9.11)$$

### 9.2.3 Experiment Evaluation

Figure 9.4 and Table 9.5 provide quantitative and qualitative prediction results for all tested predictors. The *TRUCK* dataset is densely sampled, which leads to relatively good results using the $k$-nearest neighbors search in the training dataset, which was also observed in [HITG20]. The DMD using the delay embedding always performs better than the kNN for both quantities of provided input timesteps. On the other hand, the same delay embedding in combination with an NN can neither capture the dynamics nor the two deformation patterns. In general, providing only $s = 10$ input timesteps leads to worse prediction results since more timesteps need to be predicted. Also, the deformation patterns can not be entirely distinguished after ten simulation timesteps.

Figure 9.5 shows how the errors develop over time for the six different components and numbers of input timesteps $s = 10, 12$. While being almost zero at the first predicted timestep, the error increases over time. This is expected since the predictions of the following timesteps use possibly incorrectly predicted features as inputs. Possibly due



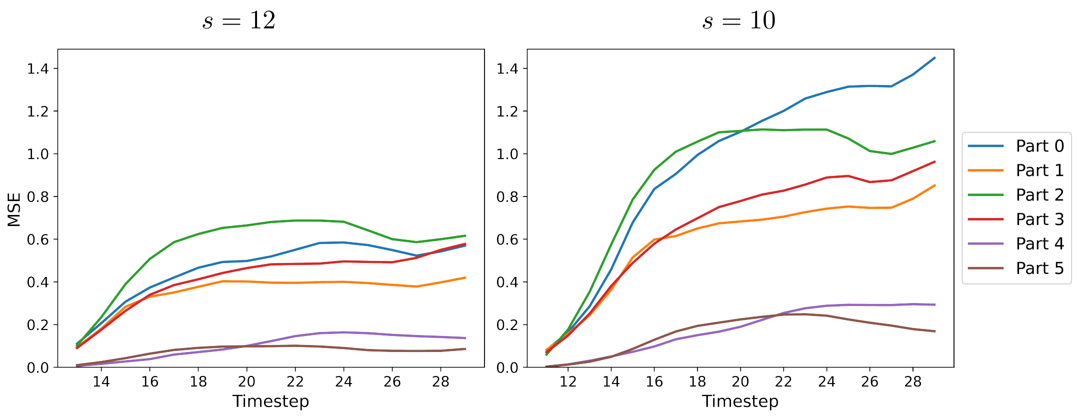


Figure 9.5: Reconstruction error to true meshes over time with DMD & delay embedding for a different number of input timesteps $s$.

to a bouncing back of the entire car towards the end of the frontal crash simulation, the errors decrease for the last timesteps.

To evaluate the usefulness of shape representations in a common space, Table 9.5 also provides the prediction results when considering one predictor for each of the six parts. For the single-part DMDs, we only use half of the DMD-embedding dimensions $r_{DMD}$. The average prediction errors are the same or slightly lower when estimating a DMD for every component separately. This shows, on the one hand, that the deformation dynamics of the six components are similar and, on the other hand, that the joint CCLB-embedding space captures these similar dynamics independent of the different mesh representations.

# 10 Conclusion and Outlook

This research work proposes and evaluates new methods for representation learning for various 3D shapes represented by different surface meshes. The two proposed surface mesh autoencoder architectures CoSMA and the CCLB-autoencoder emerged as reliable learners of shape features. The experiments with the two networks confirmed that they not only reconstruct meshes of high quality but, more importantly, calculate significant features that facilitate follow-up tasks analyzing deformations and generating shapes. This final chapter summarizes the main research findings considering all experiments and compares the two proposed architectures, given their specific advantages and limitations. Finally, I propose questions that remain to be answered and provide an outlook.

## 10.1 Summary and Contributions

### Learning 3D Shape Features

In this thesis, I propose two general network architectures, the CoSMA and the CCLB-autoencoder, to calculate low-dimensional representations of 3D shapes represented by surface meshes. Both methods have a common feature: they can handle shapes represented by different meshes, which is impossible with the baseline methods. For the analysis with baseline methods, I had to split up the datasets, separating different shapes from each other, making the task easier. Notwithstanding, the reconstructed meshes by my proposed methods are smoother and of higher quality, showing the effectiveness of both proposed methodologies.

The proposed network architectures follow different approaches for calculating low-dimensional representations for different meshes. The CoSMA pipelines split up the surface in regularly meshed patches handled individually by a network. The networks then calculate patch-wise low-dimensional representations that are set together for analyzing the entire shape. Since the CoSMAs handle patches, the pipeline using the same trained network can be applied to all surface meshes that can be split up into regularly meshed patches, making this approach very flexible. For instance, the out-of-distribution experiments in section 8.2.2 with the spectral CoSMA have proven that a network trained on meshes representing humans can be applied to a dataset of deforming car components and that I can detect deformation patterns in the calculated low-dimensional representations of the components. This also shows that the network does not overfit to one semi-regular representation and the resulting patch locations but that it learns features that are independent of the semi-regular mesh structure.

The CCLB-autoencoder can handle different mesh connectivities because it calculates shape-wise projection matrices to a common low-dimensional space that combines the

spectral bases of all shapes. I call this novel projection approach spectral mesh pooling. At the same time, the spectral mesh pooling projects the shape features to a shared space and disentangles vertex-wise features from the mesh by representing them in the spectral space. The basis defining the common low-dimensional space is optimized by leveraging correspondence maps between the shapes in the dataset. The CCLB-autoencoder has shown stable reconstruction quality despite using erroneous correspondence maps and is therefore successfully applied to datasets for which perfect maps cannot be computed, for example, between an elephant and a horse. In most cases, the CCLB-autoencoder reconstructs smoother and better-formed surface meshes than the CoSMA approaches. Only if the test meshes differ strongly from the shapes in the training dataset, the CoSMAs generalize better and reconstruct more accurate surface meshes.

Both networks learn significant low-dimensional features for all surface meshes that allow the detection of patterns and characteristics in the datasets. The low-dimensional shape features from the CoSMA networks depend on the division of the mesh into patches. Therefore, while the training is independent of the connectivity, the shape features are only comparable for meshes with the same connectivity. On the other hand, the spectral mesh pooling of the CCLB-autoencoder overcomes this limitation and allows for the first time a joint analysis of different shapes, for example, different car components that manifest similar deformation behavior over time.

### Application of Learned 3D Shape Features

The learned 3D shape features are applied in a variety of tasks. The CoSMAs patch-wise low-dimensional representations allow us to compare patterns detected on the patches to patterns in the representation of the entire shape. This comparison provides an understanding and interpretation of which surface areas lead to the patterns observed in the embedding space.

I test the smoothness of the learned embedding spaces by generating new shapes via sampling from the embedding space and decoding the created shape representations. The spectral CoSMA and the CCLB-autoencoder generate new shapes that cannot be found in the datasets. The CCLB-autoencoder can also transfer the deformation encoded in the low-dimensional representation from one shape to another because different shapes are represented in one joint embedding space.

Another advantage of this common embedding space is that it allows for conducting follow-up tasks independent of the mesh connectivity for several parts together. If the small size of the dataset limits its analysis, a representation of several shapes in one embedding space opens the door to a joint analysis by the same method. The experiments predicting time series in section 9.2 in the embedding space show how the predictions are of similar quality when considering more parts than when considering them separately. Machine Learning methods depending on large datasets will profit from joint embedding spaces for different shapes.

In comparison to the CCLB-autoencoder, which applies only one spectral mesh pooling, the CoSMAs learn hierarchical features by interleaved convolutional and pooling layers. These hierarchical features were tested in the human mesh segmentation experiment in section 9.1. Since the CoSMAs handle regional patches, providing vertex-wise shape descriptors capturing global information has proven crucial. Nevertheless, the performance of the segmentation model depends on the quality of the input shape descriptors.

#### Semi-Regular Remeshing Pipeline

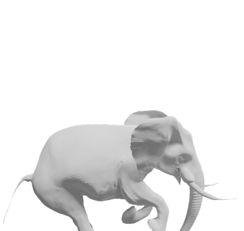
To divide the surface meshes into patches for analysis by the CoSMAs, the mesh representation must be remeshed to semi-regular connectivity. Since existing semi-regular remeshing algorithms place substantial requirements on the input surface meshes, for example, they have to be watertight and may not have holes or boundaries, I propose a novel semi-regular remeshing pipeline. It creates smooth and well-fitted semi-regular surface meshes and poses no requirements on the input meshes, as the remeshing results in section 7.2 show. Nevertheless, it does not give any convergence guarantees to a limit surface because this is irrelevant for the mesh autoencoders. The weights of the regularization terms in the loss function of the remeshing algorithm have to be selected carefully. The algorithm creates stable results once these hyperparameters are optimized for a shape. The human segmentation experiments using a CoSMA-inspired network in section 9.1 prove this, where more than 300 different human surface meshes are remeshed using the same hyperparameters for my algorithm.

## 10.2 Open Questions and Outlook

Both proposed surface mesh autoencoders require preprocessing steps to handle meshes of different connectivity. On the one hand, the CoSMA pipeline requires the surface meshes to be of semi-regular connectivity, possibly decimating detailed structures on the surface to divide the surface meshes into patches. The CCLB-autoencoder, on the other hand, requires correspondence maps between the input shapes to calculate the common spectral basis. While being stable with respect to slightly erroneous maps, it limits the datasets to contain shapes with some natural correspondence. Correspondence maps can be calculated without supervision, the provided loss function handles erroneous correspondence maps, and the reconstruction results are better than unsupervised baseline methods, see section 8.2.3. Nevertheless, it impairs the reconstruction quality, and the unsupervised correspondence learning algorithms do not yet work for all shapes. Using soft maps capturing insecurities in the learned correspondences between the shapes might improve the reconstruction quality.

While this thesis provides successful approaches to learning low-dimensional representations, it only answers how this is possible in combination with preprocessing steps. A method to divide surface meshes into patches without remeshing would make the CoSMA an even more flexible network architecture. The patch-based approach could be combined with a global feature learner to incorporate global information as done for the mesh segmentation experiment in section 9.1.

When generating meshes, for example, using the decoders of my methods, it remains an open question of how to choose a mesh for the reconstruction. Both approaches require



the mesh connectivity of a template mesh because methods generating surface meshes, including their connectivity, are still a recent and young research direction. To better understand the implications of my generative results, future studies could address how the learned low-dimensional representations improve shape generation in combination with generative neural networks.

The CoSMA networks learn hierarchical features by iterative coarsening the mesh, while the CCLB-autoencoder only applies one spectral mesh pooling operation. In general, the hierarchical calculation of features leads to more significant results. Nevertheless, it remains an open question on how to use the proposed spectral mesh pooling for iteratively coarsening the feature resolution.

Lastly, the findings of this thesis suggest testing the patch-based approach as a foundation model for the analysis of surface meshes. Large language models apply a similar approach, where a text is split into words represented in word embeddings. The CoSMA method has learned significant patch-wise features and generalizes to unseen shapes that can be reconstructed to a high quality. Therefore, these experiments open the door for a pre-learned foundation model learning shape features that can be adapted to a wide range of downstream tasks analyzing these 3D shapes.

# Bibliography

[ADMG18] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning Representations and Generative Models for 3D Point Clouds. In *Proceedings of the International Conference on Machine Learning*, volume 80, pages 40–49, 2018.

[Ado23] Adobe. Adobe Fuse 3D Characters. (`https://www.mixamo.com`). Accesed on: 2023-05-09, 2023.

[AM17] Hassan Arbabi and Igor Mezić. Ergodic Theory, Dynamic Mode Decomposition, and Computation of Spectral Properties of the Koopman Operator. *SIAM Journal on Applied Dynamical Systems*, 16(4):2096–2126, 2017.

[AO22] Souhaib Attaiki and Maks Ovsjanikov. NCP: Neural Correspondence Prior for Effective Unsupervised Shape Matching. In *Advances in Neural Information Processing Systems*, 2022.

[ASC11] Mathieu Aubry, Ulrich Schlickewei, and Daniel Cremers. The Wave Kernel Signature: A Quantum Mechanical Approach to Shape Analysis. In *Proceedings of the International Conference on Computer Vision Workshops*, pages 1626–1633. IEEE, 2011.

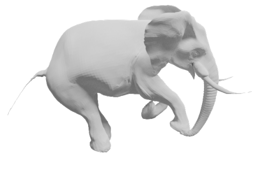
[ASK$^+$05] Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. SCAPE: Shape Completion and Animation of People. *ACM Transactions on Graphics*, 24(3):408–416, 2005.

[BBCV21] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. *arXiv preprint arXiv:2104.13478*, 2021.

[BBK09] Alexander M. Bronstein, Michael M. Bronstein, and Ron Kimmel. *Numerical Geometry of Non-Rigid Shapes*. Springer New York, 2009.

[BBL$^+$17] Michael M. Bronstein, Joan Bruna, Yann Lecun, Arthur Szlam, and Pierre Vandergheynst. Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[BBP$^+$17] Steven L. Brunton, Bingni W. Brunton, Joshua L. Proctor, Eurika Kaiser, and J. Nathan Kutz. Chaos as an Intermittently Forced Linear System. *Nature Communications*, 8(1):19, 2017.



[BBP$^+$19] Giorgos Bouritsas, Sergiy Bokhnyak, Stylianos Ploumpis, Stefanos Zafeiriou, and Michael Bronstein. Neural 3D Morphable Models: Spiral Convolutional Networks for 3D Shape Representation Learning and Generation. In *Proceedings of the International Conference on Computer Vision*, pages 7212–7221. IEEE, 2019.

[BF21] Jan Bohn and Michael Feischl. Recurrent Neural Networks as Optimal Mesh Refinement Strategies. *Computers & Mathematics with Applications*, 97:61–76, 2021.

[BGIT$^+$13] Bastian Bohn, Jochen Garcke, Rodrigo Iza-Teran, Alexander Paprotny, Benjamin Peherstorfer, Ulf Schepsmeier, and Clemens August Thole. Analysis of Car Crash Simulation Data with Nonlinear Machine Learning Methods. *Procedia Computer Science*, 18:621–630, 2013.

[BHMK$^+$18] Heli Ben-Hamu, Haggai Maron, Itay Kezurer, Gal Avineri, and Yaron Lipman. Multi-Chart Generative Surface Modeling. *ACM Transactions on Graphics*, 37(6):1–15, 2018.

[BK88] H. Bourlard and Y. Kamp. Auto-Association by Multilayer Perceptrons and Singular Value Decomposition. *Biological Cybernetics*, 59(4-5):291–294, 1988.

[BK07] David Bommes and Leif Kobbelt. Accurate Computation of Geodesic Distance Fields for Polygonal Curves on Triangle Meshes. In *Proceedings of Vision, Modeling, and Visualization Workshop*, pages 151–160, 2007.

[BKP$^+$10] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Levy. *Polygon Mesh Processing.* A K Peters/CRC Press, 2010.

[BMR$^+$16] Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Michael M. Bronstein, and Daniel Cremers. Anisotropic Diffusion Descriptors. *Computer Graphics Forum*, 35(2):431–441, 2016.

[Bor84] G. Borgefors. Distance transformations in arbitrary dimensions. *Computer Vision, Graphics, and Image Processing*, 27(3):321–345, 1984.

[BPAD23] Thomas Besnier, Emery Pierson, Sylvain Arguillère, and Mohamed Daoudi. Toward mesh-invariant 3D generative deep learning with geometric measures. *Computers & Graphics*, 2023.

[BRFF18] Pierre Baque, Edoardo Remelli, Francois Fleuret, and Pascal Fua. Geodesic Convolutional Shape Optimization. In *Proceedings of the International Conference on Machine Learning*, volume 80, pages 472–481, 2018.

[BRLB14] Federica Bogo, Javier Romero, Matthew Loper, and Michael J. Black. FAUST: Dataset and Evaluation for 3D Mesh Registration. In *Proceedings* *of the Conference on Computer Vision and Pattern Recognition*, pages 3794–3801. IEEE, 2014.

[BWS⁺18] Timur Bagautdinov, Chenglei Wu, Jason Saragih, Pascal Fua, and Yaser Sheikh. Modeling Facial Geometry Using Compositional VAEs. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 3877–3886. IEEE, 2018.

[BZSL14] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral Networks and Deep Locally Connected Networks on Graphs. In *Proceedings of the International Conference on Learning Representations*, pages 1–14, 2014.

[CBZ⁺19] Shiyang Cheng, Michael Bronstein, Yuxiang Zhou, Irene Kotsia, Maja Pantic, and Stefanos Zafeiriou. MeshGAN: Non-linear 3D Morphable Models of Faces. *arXiv preprint arXiv:1903.10384*, 2019.

[CL06] Ronald R. Coifman and Stéphane Lafon. Diffusion maps. *Applied and Computational Harmonic Analysis*, 21(1):5–30, 2006.

[CUH16] Djork Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). In *Proceedings of the International Conference on Learning Representations*, pages 1–14, 2016.

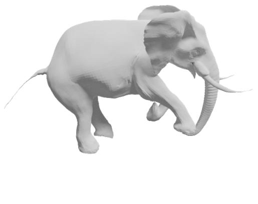
[CWKW19] Taco S. Cohen, Maurice Weiler, Berkay Kicanaoglu, and Max Welling. Gauge Equivariant Convolutional Networks and the Icosahedral CNN. In *Proceedings of the International Conference on Machine Learning*, volume 97, pages 1321–1330. PMLR, 2019.

[CWW13] Keenan Crane, Clarisse Weischedel, and Max Wardetzky. Geodesics in Heat: A New Approach to Computing Distance Based on Heat Flow. *ACM Transactions on Graphics*, 32(5):1–11, 2013.

[CYO22] Junhyeong Cho, Kim Youwang, and Tae-Hyun Oh. Cross-Attention of Disentangled Modalities for 3D Human Mesh Recovery with Transformers. In *Proceedings of the European Conference on Computer Vision*, pages 342–359, 2022.

[DBV16] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Advances in Neural Information Processing Systems*, volume 29, pages 3844–3852, 2016.

[DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the Conference of the North American*



*Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1*, pages 4171–4186, Stroudsburg, PA, USA, 2019. Association for Computational Linguistics.

[dHWCW21] Pim de Haan, Maurice Weiler, Taco Cohen, and Max Welling. Gauge Equivariant Mesh CNNs: Anisotropic Convolutions on Geometric Graphs. In *Proceedings of the International Conference on Machine Learning*, 2021.

[DKBK22] Daniel Dylewsky, Eurika Kaiser, Steven L. Brunton, and J. Nathan Kutz. Principal component Trajectories For Modeling Spectrally Continuous Dynamics as Forced Linear Systems. *Physical Review E*, 105(1):015312, 2022.

[DLG90] Nra Dyn, David Levin, and John A Gregory. A Butterfly Subdivision Scheme for Surface Interpolation with Tension Control. *ACM Transactions on Graphics*, 9(2):160–169, 1990.

[DMG$^+$16] Nat Dilokthanakul, Pedro A. M. Mediano, Marta Garnelo, Matthew C. H. Lee, Hugh Salimbeni, Kai Arulkumaran, and Murray Shanahan. Deep Unsupervised Clustering with Gaussian Mixture Variational Autoencoders. *arXiv preprint arXiv:1611.02648*, 2016.

[DMSB99] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, pages 317–324, New York, New York, USA, 1999. ACM Press.

[DSC18] Soumya Dutta, Han-Wei Shen, and Jen-Ping Chen. In Situ Prediction Driven Feature Analysis in Jet Engine Simulations. In *2018 IEEE Pacific Visualization Symposium (PacificVis)*, pages 66–75. IEEE, 2018.

[DSO20] Nicolas Donati, Abhishek Sharma, and Maks Ovsjanikov. Deep Geometric Functional Maps: Robust Feature Learning for Shape Correspondence. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 8589–8598. IEEE, 2020.

[EDD$^+$95] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution Analysis of Arbitrary Meshes. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, pages 173–182, New York, New York, USA, 1995. ACM Press.

[EK03] Asi Elad and Ron Kimmel. On Bending Invariant Signatures for Surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(10):1285–1295, 2003.

[ETLTC20] Marvin Eisenberger, Aysim Toker, Laura Leal-Taixé, and Daniel Cremers. Deep Shells: Unsupervised Shape Correspondence With Optimal Transport. In *Advances in Neural Information Processing Systems*, volume 33, pages 10491–10502, 2020.

[FCVP17] Jean Feydy, Benjamin Charlier, François-Xavier Vialard, and Gabriel Peyré. Optimal Transport for Diffeomorphic Registration. In *Medical Image Computing and Computer Assisted Intervention*, pages 291–299, 2017.

[FKSC21] Simone Foti, Bongjin Koo, Danail Stoyanov, and Matthew J. Clarkson. 3D Shape Variational Autoencoder Latent Disentanglement via Mini-Batch Feature Swapping for Bodies and Faces. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 18730–18739. IEEE, 2021.

[FLD$^+$23] Yao Feng, Jing Lin, Sai Kumar Dwivedi, Yu Sun, Priyanka Patel, and Michael J. Black. PoseGPT: Chatting about 3D Human Pose. *arXiv preprint arXiv:2311.18836*, 2023.

[FSG17] Haoqiang Fan, Hao Su, and Leonidas Guibas. A Point Set Generation Network for 3D Object Reconstruction from a Single Image. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 605–613. IEEE, 2017.

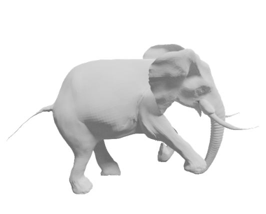
[FSV$^+$19] Jean Feydy, Thibault Séjourné, François-Xavier Vialard, Shun-Ichi Amari, Alain Trouvé, and Gabriel Peyré. Interpolating between Optimal Transport and MMD using Sinkhorn Divergences. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 89, pages 2681–2690, 2019.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016.

[GBP08] Daniela Giorgi, Silvia Biasotti, and Laura Paraboschi. SHape REtrieval Contest 2007: Watertight Models Track. *SHREC competition*, 8, 2008.

[GBS$^+$16] Yulan Guo, Mohammed Bennamoun, Ferdous Sohel, Min Lu, Jianwei Wan, and Ngai Ming Kwok. A Comprehensive Performance Evaluation of 3D Local Feature Descriptors. *International Journal of Computer Vision*, 116:66–89, 2016.

[GCBZ19] Shunwang Gong, Lei Chen, Michael Bronstein, and Stefanos Zafeiriou. SpiralNet++: A Fast and Highly Efficient Mesh Convolution Operator. In *Proceedings of the International Conference on Computer Vision Workshops*, pages 4141–4148. IEEE, 2019.



[GH97] Michael Garland and Paul S Heckbert. Surface Simplification Using Quadric Error Metrics. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, pages 209–216, 1997.

[Gia19] Dimitrios Giannakis. Data-driven Spectral Decomposition and Forecasting of Ergodic Dynamical Systems. *Applied and Computational Harmonic Analysis*, 47(2):338–396, 2019.

[GLY$^+$21] Lin Gao, Yu-Kun Lai, Jie Yang, Ling-Xiao Zhang, Shihong Xia, and Leif Kobbelt. Sparse Data Driven Mesh Deformation. *IEEE Transactions on Visualization and Computer Graphics*, 27(3):2085–2100, 2021.

[GPAM$^+$20] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[GSCO07] Ran Gal, Ariel Shamir, and Daniel Cohen-Or. Pose-Oblivious Shape Signature. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):261–271, 2007.

[GSR$^+$17] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural Message Passing for Quantum Chemistry. In *Proceedings of the International Conference on Machine Learning*, volume 70, pages 1263–1272, 2017.

[GVSS00] Igor Guskov, Kiril Vidimče, Wim Sweldens, and Peter Schröder. Normal Meshes. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, pages 95–102, New York, New York, USA, 2000. ACM Press.

[GWH$^+$21] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep Learning for 3D Point Clouds: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(12):4338–4364, 2021.

[GYH$^+$20] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit Geometric Regularization for Learning Shapes. In *Proceedings of Machine Learning and Systems*, pages 3569–3579, 2020.

[HAGO19] Ruqi Huang, Panos Achlioptas, Leonidas Guibas, and Maks Ovsjanikov. Limit Shapes – A Tool for Understanding Shape Differences and Variability in 3D Model Collections. *Computer Graphics Forum*, 38(5):187–202, 2019.

[HAGO23] Sara Hahner, Souhaib Attaiki, Jochen Garcke, and Maks Ovsjanikov. Unsupervised Representation Learning for Diverse Deformable Shape Collections. *arXiv preprint arXiv:2310.18141. Accepted at International Conference on 3D Vision 2024*, pages 1–10, 2023.

[HFLK03] John R. Hutchinson, Dan Famini, Richard Lair, and Rodger Kram. Are fast-moving elephants really running? *Nature*, 422(6931):493–494, 2003.

[HHF$^+$19] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. MeshCNN: A Network with an Edge. *ACM Transactions on Graphics*, 38(4):1–12, 2019.

[HITG20] Sara Hahner, Rodrigo Iza-Teran, and Jochen Garcke. Analysis and Prediction of Deforming 3D Shapes Using Oriented Bounding Boxes and LSTM Autoencoders. In *Artificial Neural Networks and Machine Learning*, pages 284–296. Springer International Publishing, 2020.

[HJZS20] Wenchong He, Zhe Jiang, Chengming Zhang, and Arpan Man Sainju. CurvaNet: Geometric Deep Learning based on Directional Curvature for 3D Shape Analysis. In *Proceedings of the International Conference on Knowledge Discovery & Data Mining*, pages 2214–2224, New York, NY, USA, 2020. ACM.

[HKG22] Sara Hahner, Felix Kerkhoff, and Jochen Garcke. Transfer Learning Using Spectral Convolutional Autoencoders on Semi-Regular Surface Meshes. In *Proceedings of the First Learning on Graphs Conference*, volume 198, pages 18:1–18:19. PMLR, 2022.

[HKR93] D.P. Huttenlocher, G.A. Klanderman, and W.J. Rucklidge. Comparing Images using the Hausdorff Distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863, 1993.

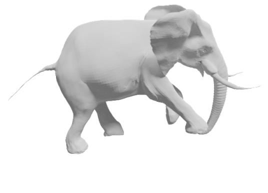
[HLG$^+$22] Shi-Min Hu, Zheng-Ning Liu, Meng-Hao Guo, Jun-Xiong Cai, Jiahui Huang, Tai-Jiang Mu, and Ralph R. Martin. Subdivision-Based Mesh Convolution Networks. *ACM Transactions on Graphics*, 41(3):1–16, 2022.

[HLRK19] Oshri Halimi, Or Litany, Emanuele Rodoì, and Ron Kimmel. Unsupervised Learning of Dense Shape Correspondence. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*. IEEE, 2019.

[Hop96] Hugues Hoppe. Progressive Meshes. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, pages 99–108, New York, New York, USA, 1996. ACM Press.

[HPCW18] Emiel Hoogeboom, Jorn W.T. Peters, Taco S. Cohen, and Max Welling. HEXACONV. *arXiv preprint arXiv:1803.02108*, pages 1–11, 2018.

[HRA$^+$19] Ruqi Huang, Marie-Julie Rakotosaona, Panos Achlioptas, Leonidas Guibas, and Maks Ovsjanikov. OperatorNet: Recovering 3D Shapes From Difference Operators. In *Proceedings of the International Conference on Computer Vision*, pages 8587–8596. IEEE, 2019.



[HSA$^+$23] Florine Hartwig, Josua Sassen, Omri Azencot, Martin Rumpf, and Mirela Ben-Chen. An Elastic Basis for Spectral Shape Correspondence. In *SIGGRAPH Conference Paper*, volume 11. ACM, 2023.

[HSBH$^+$19] Niv Haim, Nimrod Segol, Heli Ben-Hamu, Haggai Maron, and Yaron Lipman. Surface Networks via General Covers. In *Proceedings of the International Conference on Computer Vision*, pages 632–641. IEEE, 2019.

[HSKK01] Masaki Hilaga, Yoshihisa Shinagawa, Taku Kohmura, and Tosiyasu L. Kunii. Topology Matching for Fully Automatic Similarity Estimation of 3D Shapes. In *Proceedings of the annual conference on Computer graphics and interactive techniques*, pages 203–212, New York, NY, USA, 2001. ACM.

[HWM$^+$14] Jonathan H. Tu, Clarence W. Rowley, Dirk M. Luchtenburg, Steven L. Brunton, and J. Nathan Kutz. On Dynamic Mode Decomposition: Theory and Applications. *Journal of Computational Dynamics*, 1(2):391–421, 2014.

[ITG19] Rodrigo Iza-Teran and Jochen Garcke. A Geometrical Method for Low-Dimensional Representations of Simulations. *SIAM/ASA Journal on Uncertainty Quantification*, 7(2):472–496, 2019.

[JKK20] Ameya D. Jagtap, Ehsan Kharazmi, and George Em Karniadakis. Conservative Physics-informed Neural Networks on Discrete Domains for Conservation Laws: Applications to Forward and Inverse Problems. *Computer Methods in Applied Mechanics and Engineering*, 365:113028, 2020.

[KB15] Diederik P. Kingma and Jimmy Lei Ba. Adam: A Method for Stochastic Optimization. In *Proceedings of the International Conference on Learning Representations*, pages 1–15, 2015.

[KBBP16] J. Nathan Kutz, Steven L. Brunton, Bingni W. Brunton, and Joshua L. Proctor. *Dynamic Mode Decomposition.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 2016.

[KDSG23] David Kracker, Revan Kumar Dhanasekaran, Axel Schumacher, and Jochen Garcke. Method for Automated Detection of Outliers in Crash Simulations. *International Journal of Crashworthiness*, 28(1):96–107, 2023.

[KG00] Zachi Karni and Craig Gotsman. Spectral Compression of Mesh Geometry. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH '00*, pages 279–286, New York, USA, 2000. ACM Press.

[KGE$^+$21] Christopher P. Kohar, Lars Greve, Tom K. Eller, Daniel S. Connolly, and Kaan Inal. A Machine Learning Framework for Accelerating the Design

Process using CAE Simulations: An Application to Finite Element Analysis in Structural Crashworthiness. *Computer Methods in Applied Mechanics and Engineering*, 385:114008, 2021.

[KKBK20] Mason Kamb, Eurika Kaiser, Steven L. Brunton, and J. Nathan Kutz. Time-Delay Observables for Koopman: Theory and Applications. *SIAM Journal on Applied Dynamical Systems*, 19(2):886–917, 2020.

[KKL$^+$21] George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

[KN32] Bernard Osgood Koopman and John von Neumann. Dynamical Systems of Continuous Spectra. *Proceedings of the National Academy of Sciences*, 18(3):255–263, 1932.

[Koo31] Bernard Osgood Koopman. Hamiltonian Systems and Transformation in Hilbert Space. *Proceedings of the National Academy of Sciences*, 17(5):315–318, 1931.

[KPF$^+$20] Dawar Khan, Alexander Plopski, Yuichiro Fujimoto, Masayuki Kanbara, Gul Jabeen, Yongjie Zhang, Xiaopeng Zhang, and Hirokazu Kato. Surface Remeshing: A Systematic Literature Review of Methods and Research Directions. *IEEE Transactions on Visualization and Computer Graphics*, 2020.

[KS98] R. Kimmel and J. A. Sethian. Computing Geodesic Paths on Manifolds. *Proceedings of the National Academy of Sciences*, 95(15):8431–8435, 1998.

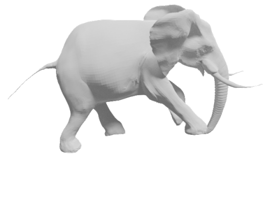
[KSS00] Andrei Khodakovsky, Peter Schröder, and Wim Sweldens. Progressive Geometry Compression. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, pages 271–278, 2000.

[KW14] Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. In *Proceedings of the International Conference on Learning Representations*, 2014.

[KW16] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv preprint arXiv:1609.02907*, pages 1–14, 2016.

[Laf04] Stéphane S Lafon. *Diffusion Maps and Geometric Harmonics.* Phd thesis, Yale University, 2004.

[LB14] Roee Litman and Alexander M. Bronstein. Learning Spectral Descriptors for Deformable Shape Correspondence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(1):171–180, 2014.



[LBBM18] Or Litany, Alex Bronstein, Michael Bronstein, and Ameesh Makadia. Deformable Shape Completion with Graph Convolutional Autoencoders. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 1886–1895. IEEE, 2018.

[LBD$^+$18] Yves Le Guennec, Jean-Patrick. Brunet, Fatima Zohra Daim, Ming Chau, and Yves Tourbier. A Parametric and Non-Intrusive Reduced Order Model of Car Crash Simulation. *Computer Methods in Applied Mechanics and Engineering*, 338:186–207, 2018.

[LDCK18] Isaak Lim, Alexander Dielen, Marcel Campen, and Leif Kobbelt. A Simple Approach to Intrinsic Correspondence Learning on Unstructured 3D Meshes. In *Proceedings of the European Conference on Computer Vision Workshops*, 2018.

[LDLD22] Clément Lemeunier, Florence Denis, Guillaume Lavoué, and Florent Dupont. Representation Learning of 3D Meshes using an Autoencoder in the Spectral Domain. *Computers & Graphics*, 107:131–143, 2022.

[lea23] learn, v. In *Oxford English Dictionary.* Oxford University Press, 2023.

[LH19] Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *Proceedings of the International Conference on Learning Representations*, 2019.

[LJB$^+$89] Yann LeCun, Lionel D. Jackel, Brian Boser, John S. Denker, Henry P. Graf, Isabelle Guyon, Don Henderson, Richard E. Howard, and William Hubbard. Handwritten Digit Recognition: Applications of Neural Network Chips and Automatic Learning. *IEEE Communications Magazine*, 27(11):41–46, 1989.

[LKB18] Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. Deep Learning for Universal Linear Embeddings of Nonlinear Dynamics. *Nature Communications*, 9(1), 2018.

[LKC$^+$20] Hsueh Ti Derek Liu, Vladimir G. Kim, Siddhartha Chaudhuri, Noam Aigerman, and Alec Jacobson. Neural Subdivision. *ACM Transactions on Graphics*, 39(4):1–16, 2020.

[LL20] Yulong Lu and Jianfeng Lu. A Universal Approximation Theorem of Deep Neural Networks for Expressing Probability Distributions. In *Advances in Neural Information Processing Systems*, pages 3094–3105. Curran Associates, Inc., 2020.

[LMS97] Mark Lanthier, Anil Maheshwari, and Jörg-Rüdiger Sack. Approximating Weighted Shortest Paths on Polyhedral Surfaces. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, pages 274–283, New York, USA, 1997. ACM Press.

[Loo87] Charles Loop. *Smooth Subdivision Surfaces Based on Triangles.* PhD thesis, The University of Utah, Masters Thesis, 1987.

[LSS$^+$98] Aaron W.F. Lee, Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. MAPS: Multiresolution Adaptive Parameterization of Surfaces. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, pages 95–104, 1998.

[LT20] Alon Lahav and Ayellet Tal. MeshWalker: Deep Mesh Understanding by Random Walks. *ACM Transactions on Graphics*, 39(6), 2020.

[LWL21a] Kevin Lin, Lijuan Wang, and Zicheng Liu. End-to-End Human Pose and Mesh Reconstruction with Transformers. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 1954–1963. IEEE, 2021.

[LWL21b] Kevin Lin, Lijuan Wang, and Zicheng Liu. Mesh Graphormer. In *Proceedings of the International Conference on Computer Vision*, pages 12919–12928. IEEE, 2021.

[Mal89] Stephane G. Mallat. A Theory for Multiresolution Signal Decomposition: The Wavelet Representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, 1989.

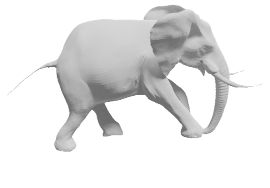
[MBBV15] Jonathan Masci, Davide Boscaini, Michael M. Bronstein, and Pierre Vandergheynst. Geodesic Convolutional Neural Networks on Riemannian Manifolds. In *Proceedings of the International Conference on Computer Vision*, pages 832–840. IEEE, 2015.

[MBM$^+$17] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 5115–5124. IEEE, 2017.

[MDSB03] Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan H Barr. Discrete Differential-Geometry Operators for Triangulated 2-Manifolds. In *Visualization and Mathematics II*, pages 35–57. Springer, Berlin, Heidelberg, 2003.

[Mez05] Igor Mezić. Spectral Properties of Dynamical Systems, Model Reduction and Decompositions. *Nonlinear Dynamics*, 41(1-3):309–325, 2005.

[MGA$^+$17] Haggai Maron, Meirav Galun, Noam Aigerman, Miri Trope, Nadav Dym, Ersin Yumer, Vladimir G. Kim, and Yaron Lipman. Convolutional Neural Networks on Surfaces via Seamless Toric Covers. *ACM Transactions on Graphics*, 36(4):1–10, 2017.



[Mit97] Tom M. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[MLDH15] Adrien Maglo, Guillaume Lavoué, Florent Dupont, and Céline Hudelot. 3D Mesh Compression: Survey, Comparisons, and Emerging Trends. *ACM Computing Surveys*, 47(3):1–41, 2015.

[MLR$^+$20] Francesco Milano, Antonio Loquercio, Antoni Rosinol, Davide Scaramuzza, and Luca Carlone. Primal-Dual Mesh Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, pages 952–963, 2020.

[MMP87] Joseph S. B. Mitchell, David M. Mount, and Christos H. Papadimitriou. The Discrete Geodesic Problem. *SIAM Journal on Computing*, 16(4):647–668, 1987.

[MRR$^+$19] Simone Melzi, Jing Ren, Emanuele Rodolà, Abhishek Sharma, Peter Wonka, and Maks Ovsjanikov. ZoomOut: Spectral upsampling for efficient shape correspondence. *ACM Transactions on Graphics*, 38(6):1–14, 2019.

[MSC$^+$13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed Representations of Words and Phrases and their Compositionality. In C J Burges, L Bottou, M Welling, Z Ghahramani, and K Q Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[Mur22] Kevin P. Murphy. *Probabilistic Machine Learning: An Introduction.* MIT Press, 2022.

[Mur23] Kevin Murphy. *Probabilistic Machine Learning: Advanced Topics.* MIT Press, 2023.

[Nat] National Crash Analysis Center (NCAC). Finite Element Model Archive (`http://web.archive.org/web/20160110143219/www.ncac.gwu.edu/vml/models.html`). Accesed on: 2016-01-10.

[NGEB20] Charlie Nash, Yaroslav Ganin, S. M. Ali Eslami, and Peter W. Battaglia. PolyGen: An Autoregressive Generative Model of 3D Meshes. In *Proceedings of the International Conference on Machine Learning*, 2020.

[NISA06] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. Laplacian Mesh Optimization. In *Proceedings of the international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, pages 381–389, New York, NY, USA, 2006. ACM.

[NO17] Dorian Nogneng and Maks Ovsjanikov. Informative Descriptor Preservation via Commutativity for Shape Matching. *Computer Graphics Forum*, 36(2):259–267, 2017.

[OBBG09] Maks Ovsjanikov, Alexander M. Bronstein, Michael M. Bronstein, and Leonidas J. Guibas. Shape Google: A Computer Vision Approach to Isometry Invariant Shape Retrieval. In *Proceedings of the International Conference on Computer Vision Workshops*, pages 320–327. IEEE, 2009.

[OBCS$^+$12] Maks Ovsjanikov, Mirela Ben-Chen, Justin Solomon, Adrian Butscher, and Leonidas Guibas. Functional Maps: A Flexible Representation of Maps Between Shapes. *ACM Transactions on Graphics*, 31(4), 2012.

[OCB$^+$17] Maks Ovsjanikov, Etienne Corman, Michael Bronstein, Emanuele Rodolà, Mirela Ben-Chen, Leonidas Guibas, Frederic Chazal, and Alex Bronstein. Computing and Processing Correspondences with Functional Maps. In *ACM SIGGRAPH 2017 Courses*, pages 1–62, New York, NY, USA, 2017. ACM.

[OR19] Samuel E. Otto and Clarence W. Rowley. Linearly Recurrent Autoencoder Networks for Learning Dynamics. *SIAM Journal on Applied Dynamical Systems*, 18(1):558–593, 2019.

[PC19] Gabriel Peyré and Marco Cuturi. Computational Optimal Transport. *Foundations and Trends in Machine Learning*, 11(5-6):355–607, 2019.

[PD20] Shaowu Pan and Karthik Duraisamy. Physics-Informed Probabilistic Learning of Linear Embeddings of Nonlinear Dynamics with Guaranteed Stability. *SIAM Journal on Applied Dynamical Systems*, 19(1):480–509, 2020.

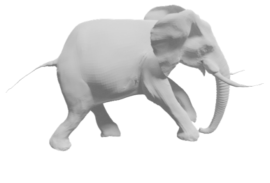
[Pea01] Karl Pearson. On Lines and Planes of Closest Fit to Systems of Points in Space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[Pey08] Gabriel Peyré. Image Processing with Nonlocal Spectral Bases. *Multiscale Modeling & Simulation*, 7(2):703–730, 2008.

[Pey09] Gabriel Peyré. Manifold Models for Signals and Images. *Computer Vision and Image Understanding*, 113(2):249–260, 2009.

[PFS$^+$19] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 165–174. IEEE, 2019.

[PGM$^+$19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative



Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, volume 32, pages 8026–8037, 2019.

[PNI$^+$18] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep Contextualized Word Representations. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1*, pages 2227–2237, Stroudsburg, PA, USA, 2018. Association for Computational Linguistics.

[PP93] Ulrich Pinkall and Konrad Polthier. Computing Discrete Minimal Surfaces and Their Conjugates. *Experimental Mathematics*, 2(1):15–36, 1993.

[PPGS21] Thomas Pierrot, Nicolas Perrin-Gilbert, and Olivier Sigaud. First-Order and Second-Order Variants of the Gradient Descent in a Unified Framework. In *Artificial Neural Networks and Machine Learning*, pages 197–208. Springer International Publishing, 2021.

[PRS15] Frédéric Payan, Céline Roudet, and Basile Sauvage. Semi-Regular Triangle Remeshing: A Comprehensive Study. *Computer Graphics Forum*, 34(1):86–102, 2015.

[PWT$^+$22] Yatian Pang, Wenxiao Wang, Francis E. H. Tay, Wei Liu, Yonghong Tian, and Li Yuan. Masked Autoencoders for Point Cloud Self-supervised Learning. In *Proceedings of the European Conference on Computer Vision*, pages 604–621, 2022.

[QLP$^+$22] Guocheng Qian, Yuchen Li, Houwen Peng, Jinjie Mai, Hasan Abed Al Kader Hammoud, Mohamed Elhoseiny, and Bernard Ghanem. PointNeXt: Revisiting PointNet++ with Improved Training and Scaling Strategies. In *Advances in Neural Information Processing Systems*, pages 23192–23204, 2022.

[QSMG17] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 77–85. IEEE, 2017.

[QYSG17] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[QYW$^+$19] Can Qin, Haoxuan You, Lichen Wang, C.-C Jay Kuo, and Yun Fu. PointDAN: A Multi-Scale 3D Domain Adaption Network for Point Cloud Representation. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[RBG$^+$09] Martin Reuter, Silvia Biasotti, Daniela Giorgi, Giuseppe Patanè, and Michela Spagnuolo. Discrete Laplace–Beltrami Operators for Shape Analysis and Segmentation. *Computers & Graphics*, 33(3):381–390, 2009.

[RBSB18] Anurag Ranjan, Timo Bolkart, Soubhik Sanyal, and Michael J. Black. Generating 3D Faces Using Convolutional Mesh Autoencoders. In *Proceedings of the European Conference on Computer Vision*, pages 725–741, 2018.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention - MICCAI*, pages 234–241. Springer International Publishing, 2015.

[RMB$^+$09] Clarence W. Rowley, Igor Mezić, Shervin Bagheri, Philipp Schlatter, and Dan S. Henningson. Spectral Analysis of Nonlinear Flows. *Journal of Fluid Mechanics*, 641:115–127, 2009.

[RMW14] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In *Proceedings of the International Conference on Machine Learning*, volume 32, pages 1278–1286, 2014.

[ROA$^+$13] Raif M. Rustamov, Maks Ovsjanikov, Omri Azencot, Mirela Ben-Chen, Frédéric Chazal, and Leonidas Guibas. Map-based Exploration of Intrinsic Shape Differences and Variability. *ACM Transactions on Graphics*, 32(4):1–12, 2013.

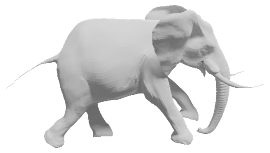
[RRN$^+$20] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3D Deep Learning with PyTorch3D. *arXiv preprint arXiv:2007.08501*, 2020.

[RSO19] Jean-Michel Roufosse, Abhishek Sharma, and Maks Ovsjanikov. Unsupervised Deep Learning for Structured Shape Matching. In *Proceedings of the International Conference on Computer Vision*, pages 1617–1627. IEEE, 2019.

[RTG00] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The Earth Mover's Distance as a Metric for Image Retrieval. *International Journal of Computer Vision*, 40:99–121, 2000.

[RWP06] Martin Reuter, Franz-Erich Wolter, and Niklas Peinecke. Laplace–Beltrami Spectra as 'Shape-DNA' of Surfaces and Solids. *Computer-Aided Design*, 38(4):342–366, 2006.

[SAA$^+$19] Yawar Siddiqui, Antonio Alliegro, Alexey Artemov, Tatiana Tommasi, Daniele Sirigatti, Vladislav Rosov, Angela Dai, and Matthias Nießner.



MeshGPT: Generating Triangle Meshes with Decoder-Only Transformers. *arXiv preprint arXiv:2311.15475*, 2019.

[SACO22] Nicholas Sharp, Souhaib Attaiki, Keenan Crane, and Maks Ovsjanikov. DiffusionNet: Discretization Agnostic Learning on Surfaces. *ACM Transactions on Graphics*, 41(3):1–16, 2022.

[SBR16] Ayan Sinha, Jing Bai, and Karthik Ramani. Deep Learning 3D Shape Surfaces Using Geometry Images. In *Proceedings of the European Conference on Computer Vision*, pages 223–240, 2016.

[SC20] Nicholas Sharp and Keenan Crane. A Laplacian for Nonmanifold Triangle Meshes. *Computer Graphics Forum*, 39(5), 2020.

[Sch10] Peter J Schmid. Dynamic Mode Decomposition of Numerical and Experimental Data. *Journal of Fluid Mechanics*, 656:5–28, 2010.

[Sch22] Peter J Schmid. Dynamic Mode Decomposition and Its Variants. *Annual Review of Fluid Mechanics*, 54(1):225–254, 2022.

[SCZZ20] Shiliang Sun, Zehui Cao, Han Zhu, and Jing Zhao. A Survey of Optimization Methods from a Machine Learning Perspective. *IEEE Transactions on Cybernetics*, 50(8):3668–3681, 2020.

[SFC$^+$21] Zhengyang Shen, Jean Feydy, Ariel Hernán Curiale, Ruben San, José Estépar, Raúl San, and Marc Niethammer. Accurate Point Cloud Registration with Robust Optimal Transport. In *Advances in Neural Information Processing Systems*, 2021.

[SH19] Constantin Steppa and Tim L. Holch. HexagDLy—Processing Hexagonally Sampled Data with CNNs in PyTorch. *SoftwareX*, 9:193–198, 2019.

[SMDH13] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the International Conference on Machine Learning*, pages 1139–1147, 2013.

[SO20] Abhishek Sharma and Maks Ovsjanikov. Weakly Supervised Deep Functional Map for Shape Matching. In *Advances in Neural Information Processing Systems*, volume 33, pages 19264–19275, 2020.

[SOCG10] Primoz Skraba, Maks Ovsjanikov, Frederic Chazal, and Leonidas Guibas. Persistence-based Segmentation of Deformable Shapes. In *Proceedings of the Conference on Computer Vision and Pattern Recognition Workshops*, pages 45–52. IEEE, 2010.

[SOG09] Jian Sun, Maks Ovsjanikov, and Leonidas Guibas. A Concise and Provably Informative Multi-Scale Signature Based on Heat Diffusion. *Computer Graphics Forum*, 28(5):1383–1392, 2009.

[SP04] Robert W. Sumner and Jovan Popović. Deformation Transfer for Triangle Meshes. *ACM Transactions on Graphics*, 23(3):399–405, 2004.

[SS21] Dmitriy Smirnov and Justin Solomon. HodgeNet: Learning Spectral Geometry on Triangle Meshes. *ACM Transactions on Graphics*, 40(4):1–11, 2021.

[SSK+05] Vitaly Surazhsky, Tatiana Surazhsky, Danil Kirsanov, Steven J Gortler, and Hugues Hoppe. Fast Exact and Approximate Geodesics on Meshes. *ACM Transactions on Graphics*, 24(3):553–560, 2005.

[SUHR17] Ayan Sinha, Asim Unmesh, Qixing Huang, and Karthik Ramani. SurfNet: Generating 3D Shape Surfaces Using Deep Residual Networks. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 791–800. IEEE, 2017.

[SWL+20] Yongbin Sun, Yue Wang, Ziwei Liu, Joshua E Siegel, and Sanjay E Sarma. PointGrow: Autoregressively Learned Point Cloud Generation with Self-Attention. In *Proceedings of the Winter Conference on Applications of Computer Vision*, pages 61–70. IEEE, 2020.

[Tak81] Floris Takens. Detecting Strange Attractors in Turbulence. In *Dynamical Systems and Turbulence, Warwick 1980*, pages 366–381, 1981.

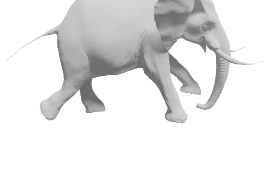
[TdSL00] Joshua B Tenenbaum, Vin de Silva, and John C Langford. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500):2319–2323, 2000.

[TGLX18] Qingyang Tan, Lin Gao, Yu Kun Lai, and Shihong Xia. Variational Autoencoders for Deforming 3D Mesh Models. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 5841–5850. IEEE, 2018.

[TKY17] Naoya Takeishi, Yoshinobu Kawahara, and Takehisa Yairi. Learning Koopman Invariant Subspaces for Dynamic Mode Decomposition. In *Advances in Neural Information Processing Systems*, pages 1130–1140, 2017.

[TLI+23] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971*, 2023.

[TPZ18] Maxim Tatarchenko, Jaesik Park, and Qian-Yi Zhou. Tangent Convolutions for Dense Prediction in 3D. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 3887–3896. IEEE, 2018.



[VBMP08] Daniel Vlasic, Ilya Baran, Wojciech Matusik, and Jovan Popović. Articulated Mesh Animation from Multi-view Silhouettes. *ACM Transactions on Graphics*, 27(3):1–9, 2008.

[vdMH08] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[VL08] B. Vallet and B. Lévy. Spectral Geometry Processing with Manifold Harmonics. *Computer Graphics Forum*, 27(2):251–260, 2008.

[vRMS$^+$20] Laura von Rueden, Sebastian Mayer, Rafet Sifa, Christian Bauckhage, and Jochen Garcke. Combining Machine Learning and Simulation to a Hybrid Modelling Approach: Current and Future Directions. In *Advances in Intelligent Data Analysis XVIII*, pages 548–560. Springer International Publishing, 2020.

[VSP$^+$17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *Advances in Neural Information Processing Systems*, pages 5999–6009, 2017.

[WBBP11] Chaohui Wang, Michael M. Bronstein, Alexander M. Bronstein, and Nikos Paragios. Discrete Minimum Distortion Correspondence Problems for Non-rigid Shape Matching. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pages 580–591. Springer, Berlin, Heidelberg, 2011.

[WEH20] Ruben Wiersma, Elmar Eisemann, and Klaus Hildebrandt. CNNs on Surfaces Using Rotation-Equivariant Features. *ACM Transactions on Graphics*, 39(4), 2020.

[WFVW21] Maurice Weiler, Patrick Forré, Erik Verlinde, and Max Welling. Coordinate Independent Convolutional Networks – Isometry and Gauge Equivariant Convolutions on Riemannian Manifolds. *arXiv preprint arXiv:2106.06020*, pages 1–271, 2021.

[WGS$^+$18] Pengyu Wang, Yuan Gan, Panpan Shui, Fenggen Yu, Yan Zhang, Songle Chen, and Zhengxing Sun. 3D Shape Segmentation via Shape Fully Convolutional Networks. *Computers & Graphics*, 76:182–192, 2018.

[WHG13] Fan Wang, Qixing Huang, and Leonidas J. Guibas. Image Co-Segmentation via Consistent Functional Maps. In *Proceedings of the International Conference on Computer Vision*, pages 849–856. IEEE, 2013.

[WLL$^+$23] Jindong Wang, Cuiling Lan, Chang Liu, Yidong Ouyang, Tao Qin, Wang Lu, Yiqiang Chen, Wenjun Zeng, and Philip Yu. Generalizing to Unseen Domains: A Survey on Domain Generalization. *IEEE Transactions on Knowledge and Data Engineering*, 35(8):8052 – 8072, 2023.

[WNEH22] Ruben Wiersma, Ahmad Nasikun, Elmar Eisemann, and Klaus Hildebrandt. DeltaConv: Anisotropic Operators for Geometric Deep Learning on Point Clouds. *ACM Transactions on Graphics*, 41(4), 2022.

[Won23] Chi-Chong Wong. Heat Diffusion based Multi-scale and Geometric Structure-aware Transformer for Mesh Segmentation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 4413–4422. IEEE, 2023.

[WPC$^+$21] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

[WPZ$^+$21] Tong Wu, Liang Pan, Junzhe Zhang, Tai Wang, Ziwei Liu, and Dahua Lin. Density-aware Chamfer Distance as a Comprehensive Metric for Point Cloud Completion. In *Advances in Neural Information Processing Systems*, 2021.

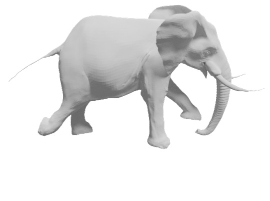
[XHG$^+$23] Aoran Xiao, Jiaxing Huang, Dayan Guan, Xiaoqin Zhang, Shijian Lu, and Ling Shao. Unsupervised Point Cloud Representation Learning with Deep Neural Networks: A Survey. *Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[XLH$^+$21] Wenjun Xia, Zexin Lu, Yongqiang Huang, Zuoqiang Shi, Yan Liu, Hu Chen, Yang Chen, Jiliu Zhou, and Yi Zhang. MAGIC: Manifold and Graph Integrative Convolutional Network for Low-Dose CT Reconstruction. *IEEE Transactions on Medical Imaging*, 40(12):3459–3472, 2021.

[YDP$^+$23] Jiachen Yang, Tarik Dzanic, Brenden Petersen, Jun Kudo, Ketan Mittal, Vladimir Tomov, Jean-Sylvain Camier, Tuo Zhao, Hongyuan Zha, Tzanio Kolev, Robert Anderson, and Daniel Faissol. Reinforcement Learning for Adaptive Mesh Refinement. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 206, pages 5997–6014, 2023.

[YFST18] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. FoldingNet: Point Cloud Auto-encoder via Deep Grid Deformation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 206–215. IEEE, 2018.

[YGT$^+$23] Jie Yang, Lin Gao, Qingyang Tan, Yi-Hua Huang, Shihong Xia, and Yu-Kun Lai. Multiscale Mesh Deformation Component Analysis With Attention-Based Autoencoders. *IEEE Transactions on Visualization and Computer Graphics*, 29(2):1301–1317, 2023.



[YLY$^+$20] Yu-Jie Yuan, Yu-Kun Lai, Jie Yang, Qi Duan, Hongbo Fu, and Lin Gao. Mesh Variational Autoencoders with Edge Contraction Pooling. In *Conference on Computer Vision and Pattern Recognition Workshops*, pages 1105–1112. IEEE, 2020.

[Yos23] Yusuke Yoshiyasu. Deformable Mesh Transformer for 3D Human Mesh Recovery. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 17006–17015. IEEE, 2023.

[YXC$^+$21] Haotian Ye, Chuanlong Xie, Tianle Cai, Ruichen Li, Zhenguo Li, and Liwei Wang. Towards a Theoretical Framework of Out-of-Distribution Generalization. In *Advances in Neural Information Processing Systems*, pages 23519–23531, 2021.

[YZZ$^+$21] Yuan Yuan, Kaiwen Zhou, Wenwu Zhou, Xin Wen, and Yingzheng Liu. Flow Prediction Using Dynamic Mode Decomposition with Time-Delay Embedding Based on Local Measurement. *Physics of Fluids*, 33(9), 2021.

[ZBDT19] Yongheng Zhao, Tolga Birdal, Haowen Deng, and Federico Tombari. 3D Point Capsule Networks. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 1009–1018. IEEE, 2019.

[ZCAK23] Haoliang Zhang, Samuel Cheng, Christian El Amm, and Jonghoon Kim. Efficient Pooling Operator for 3D Morphable Models. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–9, 2023.

[ZLQ$^+$23] Kaiyang Zhou, Ziwei Liu, Yu Qiao, Tao Xiang, and Chen Change Loy. Domain Generalization: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):4396 – 4415, 2023.

[Zor07] Denis Zorin. Subdivision on Arbitrary Meshes: Algorithms and Theory. In *Mathematics and Computation in Imaging Science and Information Processing*, pages 1–46. World Scientific, 2007.

[ZWL$^+$20] Yi Zhou, Chenglei Wu, Zimo Li, Chen Cao, Yuting Ye, Jason Saragih, Hao Li, and Yaser Sheikh. Fully Convolutional Mesh Autoencoder using Efficient Spatially Varying Kernels. In *Advances in Neural Information Processing Systems*, volume 33, pages 9251–9262, 2020.