# Deep Learning on Geometry Representations

by

Dmitriy Smirnov

B.A., Pomona College (2017)
S.M., Massachusetts Institute of Technology (2019)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2022



Author . . . . . .
Department of Electrical Engineering and Computer Science
May 13, 2022

Certified by . . . . . .
Justin Solomon
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

**Deep Learning on Geometry Representations**

by

Dmitriy Smirnov

Submitted to the Department of Electrical Engineering and Computer Science
on May 13, 2022, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

## Abstract

While deep learning has been successfully applied to many tasks in computer graphics and vision, standard learning architectures often operate on shape representations that are dense and regular, like pixel or voxel grids. On the other hand, decades of computer graphics and geometry processing research have resulted in specialized algorithms and tools that use representations without such regular structure. In this thesis, we revisit conventional approaches in graphics in geometry to propose deep learning pipelines and inductive biases that are directly compatible with common geometry representations, without relying on simple uniform structure.

Thesis Supervisor: Justin Solomon
Title: Associate Professor of Electrical Engineering and Computer Science

# Acknowledgements

The work in this dissertation has a complex dependency graph with a combinatorial explosion of individuals without any of whom none of this would have been possible. Below I highlight just a few of those who helped me become who I am today, both personally and professionally.

I got my first taste of research in computer science during my college years thanks to Ran Libeskind-Hadas at Harvey Mudd. My interest in geometry and topology in particular was spurred by Vin de Silva's *Topics in Geometry and Topology* class as well as our collaborations that followed. I am also very grateful to Dmitriy Morozov for guidance, mentorship, and friendship.

I have been lucky to interact with some remarkable people at MIT both within our lab and outside of it. I am glad to have gotten the chance to collaborate with some of them—Misha Bessmeltsev, Paul Zhang, David Palmer, Lingxiao Li, Mazdak Abulnaga—on research projects, but many others have had an influence via random chats in the hallway or over a meal. Most importantly, I would like to thank my advisor, Justin Solomon, whose enthusiasm, patience, support, and endless knowledge have been invaluable over the past five years.

A significant portion of the work in this dissertation was completed as a result of projects that began during internships at Adobe Research. I am grateful to have had the opportunity to learn from many wonderful collaborators at Adobe including Matt Fisher, Vova Kim, Michaël Gharbi, Alyosha Efros, and Richard Zhang.

None of this would have been possible without the unconditional love and support from my friends and family—on both sides of the country. I am grateful to my friends who managed to keep in touch throughout the last five years: Adam Fisch, Adam Revello, Antony Bello, Igor Berman, Katie Lewis, Kieran McVeigh, Leo Selker, Misha Vysotskiy, Noam Hurwitz, Rachel Zimmerman, Sam Warren, Sarah Gamble, Tal Schuster, Ted Kornish, Zivvy Epstein. Especially, I would like to thank Gabi and her family for giving me a second home on the East Coast and my

parents and grandmother for always keeping the door open to my first on the West.

My PhD research has been supported by the NSF Graduate Research Fellowship under Grant No. 1122374.

# Contents


**Abstract** 2

**Acknowledgements** 3

**1 Introduction** 9

1.1 Motivation . . . 10

1.2 Related work . . . 11

1.2.1 Loss functions . . . 11

1.2.2 Shape representation . . . 12

1.3 Overview . . . 14

**2 Learning from Triangle Meshes** 15

2.1 Introduction . . . 15

2.2 Related work . . . 18

2.2.1 Spectral Shape Analysis . . . 18

2.2.2 Neural Networks on Meshes . . . 20

2.3 Overview . . . 23

2.4 Operator construction . . . 24

2.4.1 Operator . . . 24

2.4.2 Vectorial operators . . . 27

2.5 Differentiable spectral analysis . . . 27

2.5.1 The HodgeNet generalized eigenproblem . . . 28

2.5.2 Derivative formulas . . . 29

2.5.3 Derivation of derivative formulas . . . 30
2.5.4 Derivative approximation . . . 34
2.6 From eigenvectors to features . . . 35
2.7 Additional details and parameters . . . 36
2.8 Experiments . . . 37
2.8.1 Mesh segmentation . . . 37
2.8.2 High-resolution mesh segmentation . . . 38
2.8.3 Mesh classification . . . 38
2.8.4 Dihedral angle prediction . . . 39
2.8.5 Ablation . . . 40
2.9 Discussion . . . 40

**3 Learning Parametric Shapes** **42**
3.1 Introduction . . . 42
3.2 Related work . . . 45
3.3 Preliminaries . . . 48
3.4 Method . . . 49
3.4.1 Learning parametric shapes using distance fields . . . 49
3.4.2 Learning parametric patches . . . 55
3.5 Experiments . . . 60
3.5.1 2D: Font exploration and manipulation . . . 60
3.5.2 3D: Volumetric primitive abstraction . . . 66
3.5.3 3D: Patch-based CAD modeling . . . 67
3.6 Discussion . . . 75

**4 Learning Manifolds with Boundary** **77**
4.1 Introduction . . . 77
4.2 Related work . . . 79
4.2.1 Minimal Surface Computation . . . 79
4.2.2 Deep Learning for Shape Reconstruction . . . 80
4.3 Preliminaries . . . 81

4.3.1 Currents . . . 82
4.3.2 Boundary operator . . . 83
4.3.3 Mass norm . . . 83
4.3.4 Minimal mass problem . . . 83
4.3.5 Representing currents by forms . . . 84
4.4 DeepCurrents . . . 85
4.4.1 Neural representation . . . 85
4.4.2 Modifying the metric . . . 86
4.4.3 Loss functions . . . 88
4.4.4 Network architecture . . . 89
4.5 Experimental results . . . 90
4.5.1 Minimal surfaces . . . 90
4.5.2 Surface reconstruction . . . 91
4.5.3 Latent space learning . . . 92
4.5.4 Ablation study . . . 93
4.6 Discussion . . . 94

**5 Learning Sprites** **96**
5.1 Introduction . . . 96
5.2 Related work . . . 97
5.3 Method . . . 99
5.3.1 Dictionary and sprite generator . . . 100
5.3.2 Layered frame decomposition using sprite anchors . . . 101
5.3.3 Per-anchor sprite selection . . . 102
5.3.4 Local sprite transformations . . . 102
5.3.5 Compositing and reconstruction . . . 103
5.3.6 Training procedure . . . 104
5.4 Experimental results . . . 105
5.4.1 Comparisons . . . 105
5.4.2 Sprite-based game deconstruction . . . 107

5.4.3 Ablation study . . . . . 108
5.4.4 Future directions and limitations . . . . . 109
5.5 Discussion . . . . . 110

**6 Conclusion** **112**

**Bibliography** **114**

# 1

# Introduction

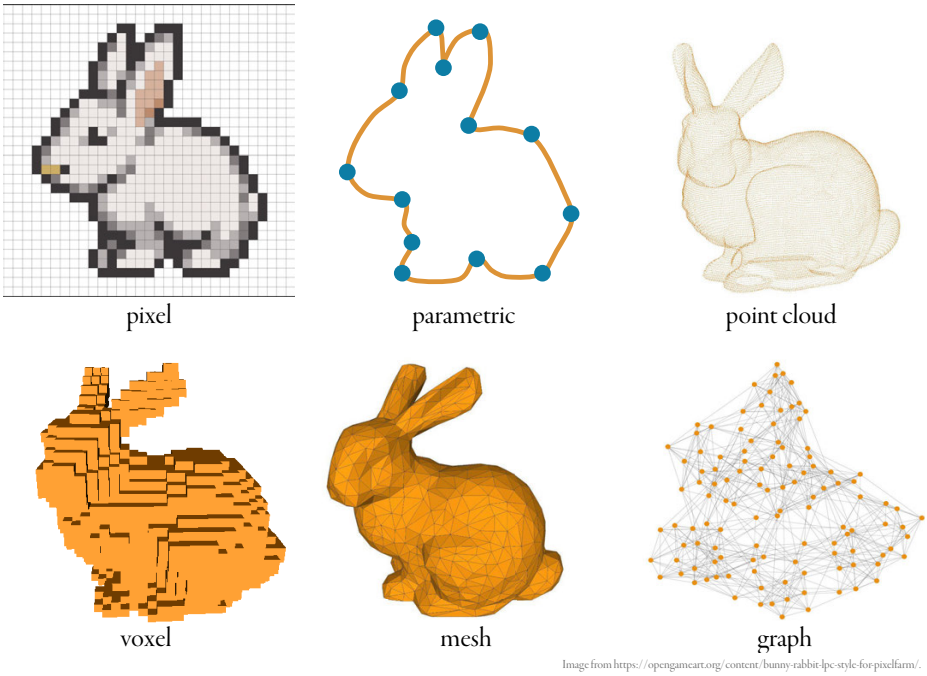



Figure 1-1: Some common representations for 2D and 3D geometry. While each has its advantages and drawbacks, often the representations that are intuitively compatible with standard deep learning architectures (e.g., pixel or voxel grids) are not the ones required by downstream tasks. In this thesis, we consider deep learning–based approaches to processing or producing data in formats that lack this dense regular structure.

## 1.1 Motivation

Over the past decade, the culmination of developments in hardware, large-scale availability of training data, and algorithmic advances have fueled the success of deep learning for a variety of applications. Neural networks are broadly able to process incomplete, messy, and ambiguous input, producing useful and consistent output.

Computer graphics and vision are no exception to the proliferation of deep learning, and data-driven approaches have now become standard for many tasks. Within these fields, the most fruitful deep learning architecture has been the convolutional neural network (CNN). Today CNNs achieve state-of-the-art results in tasks such as image classification [Dai+21], segmentation [Zha+20; Dut+20; Yu+20], object detection [WH21; BWL20], and image-to-image translation [Iso+17].

CNNs, however, operate on *raster* representations (e.g., the pixel and voxel grids in Fig. 1-1). Grid structure is fundamentally built into convolution as a mechanism for information to travel between network layers and between pixels. This structure is leveraged to optimize GPU hardware performance, and many of the readily available datasets are comprised of examples formatted as pixel or voxel grids.

Raster representations are easy to work with for a variety of reasons. They are compatible with standard input and output devices, like camera sensors and screens, and constitute an effective means to represent diverse visual content of varying structure and topology. As such, CNNs leverage a reasonable inductive bias by processing data in a simple *Eulerian* fashion, applying fixed operations to a dense grid.

On the other hand, *Lagrangian* representations use sparse sets of parameters that move with the shape, i.e., control points to express geometry. Such representations offer distinct advantages. By expressing shapes as collections of primitives, we can easily apply transformations and render at arbitrary resolution while storing only a sparse representation. Moreover, parametric representations are effective for high-level reasoning such as discovering common underlying structure and estimating correspondences between shapes, facilitating tools for retrieval, exploration, and style/structure transfer. They are intuitive to edit with conventional software, are resolution-independent, and are efficient to store.

Many tools, algorithms, and mathematical frameworks have been developed for authoring, manipulating, and analyzing such Lagrangian content. Standard CAD and 3D modeling software is used by artists, engineers, and animators. Simulations rely on finite element analysis using meshes. However, by restricting the shape modalities compatible with our machine learning methods to Eulerian grids, we forego the many insights and techniques from these well-established tools.

In this thesis, we propose to design deep learning algorithms by considering shape representations as first class citizens. Rather than taking standard architectures, loss functions, and training algorithms for granted and thus accepting the respective input and output shape modalities, we consider other geometric atomic units that are richer and more suitable for applications than the pixel. This paradigm shift motivates us to modernize conventional approaches and applications that predate deep learning. In particular, we borrow ideas from fields like metric geometry, geometric measure theory, spectral geometry, and animation to develop custom-tailed loss functions, architectures, and training pipelines, making deep learning a more useful tool for practitioners working with visual data.

## 1.2 Related work

Several design choices must be made when developing a deep learning algorithm that either inputs or outputs geometry. Beyond the typical decisions of network architecture, optimizer, and dataset, two particularly important aspects are the shape representation and the loss function. The representation must be both compatible with the available data and useful for the target application, and the loss function must provide a sensible interface between the training data and network output. Many approaches to deep learning on geometric data bridge the gap between these two aspects, and in this thesis we also pay particular attention to their interplay. Below, we summarize some of the recent developments.

### 1.2.1 Loss functions

One popular direction for designing loss functions that operate on geometry employs a differentiable renderer and measures 2D image loss between a rendering of the inferred 3D model and the

input image [KUH18; Wu+16; Yan+16; Rez+16; Wu+17; TEM18]. A notable example is the work by Wu et al. [Wu+17], which learns a mapping from a photograph to a normal map, a depth map, a silhouette, and the mapping from these outputs to a voxelization. They use a differentiable renderer and measure inconsistencies in 2D. Another approach to learning geometry uses 3D loss functions, measuring discrepancies between the predicted and target 3D shapes directly, often via Chamfer or a regularized Wasserstein distance [Wil+19; Man+18; Gro+18; Par+19; Gao+19; HTM19].

### 1.2.2 Shape representation

As noted by Park et al. [Par+19], in deep learning, explicit geometric representations, i.e., those where the shape coordinates are directly specified with an algebraic expression, can be divided into three classes: voxel-, point-, and mesh-based. Voxel-based methods [Del+18; Wu+17; Zha+18; Wu+18], which operate on geometry defined on regular grids, yield dense reconstruction that are limited in resolution, offer no topological guarantees, and cannot represent sharp features. Point-based approaches represent geometry as a point cloud [Yin+18; Man+18; FSG17a; Lun+17; Yan+18], capturing discrete samples of the shape but not manifold connectivity.

Some recent methods represent shapes using meshes [Nas+20; Han+19; WEH20; Wan+18b; Mil+20a; Sha+22]. Typically, these approaches consider the mesh as a graph structure and define a convolutional operator to propagate information. The most recent approaches to mesh-based learning [Sha+22], including the one proposed in this thesis, propose an architecture that learns features on the underlying surface rather than the particular mesh triangulation.

While the content produced by these explicit approaches is generally easy to render and manipulate, it is often restricted in topology and/or resolution, limiting expressiveness.

A different approach circumvents topology and resolution issues by representing 3D shapes *implicitly*, using functions parameterized by neural networks. In DeepSDF, Park et al. [Par+19] learn a field that approximates signed distance to the target geometry, while Mescheder et al. [Mes+19] and Chen and Zhang [CZ19] classify query points as being outside or inside a shape. Others further improve the results by proposing regularizers, loss functions, and training or rendering approaches [Gro+20; AL20; Tak+21; Lip21]. While these works achieve impressive levels of detail in surface reconstruction, they largely suffer from two drawbacks—lack of control and

inability to represent open surfaces, i.e., those with boundary.

Neural implicit learning methods typically overfit to a single target shape or learn a family of shapes parameterized by a high-dimensional latent space. While recent work has shown the possibility of adapting classical geometry processing algorithms to neural implicit geometries [Yan+21], applying targeted manipulations and deformations to learned shapes remains nontrivial. Several papers propose *hybrid representations*, combining the expressive power of neural implicit representations with the control afforded by explicit geometries. Genova et al. [Gen+20] reconstruct shapes by learning multiple implicit representations arranged according to a learned template configuration. In DualSDF [Hao+20], manipulations can be applied to learned implicit shapes by making changes to corresponding explicit geometric primitives. BSP-Net [CTZ20] and CvxNet [Den+20] restrict the class of learned implicit surfaces to half-spaces and convex hulls, respectively.

Because neural implicit shapes are typically level sets of learned functions, they must be closed surfaces. Two notable exceptions are [CMP20], which learns *unsigned* distance functions rather than SDFs, and [Ven+21], which maps an input point to its closest point on the target surface.

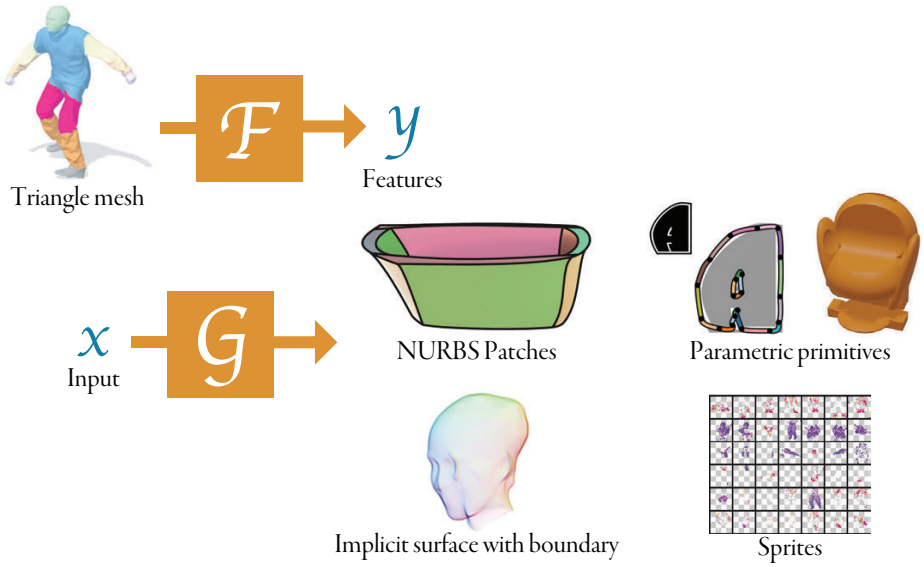



Figure 1-2: An overview of the shape representations explored in this thesis. We consider deep learning pipelines that take as input or produce as output geometry in common and useful formats. Particularly, we propose a system that learns features on triangle meshes as well as pipelines that output parametric patches and primitives, implicit surfaces with boundary, and sprites.

## 1.3 Overview

In this thesis, we propose deep learning architectures, training procedures, and algorithms that make it feasible to train neural networks on datasets of visual data in formats that are intuitive to use and compatible with common downstream tasks and applications, such as design, modeling, simulation, and rendering. In Chapter 2, we describe an encoder that operates on triangle meshes, borrowing ideas from spectral geometry. We consider two approaches, inspired by metric geometry, to produce parametrically-defined shapes, like CAD models, in Chapter 3. Chapter 4 introduces a hybrid shape representation that combines the merits of explicit geometry with the advantages of implicit representations. Finally, in Chapter 5, we propose a self-supervised approach to learning intuitive decompositions for collections of images, e.g., frames of animations or video games, which allow for high-level manipulation, using the learned texture patches as geometric primitives. We illustrate our contributions in Fig. 1-2.

# 2

# Learning from Triangle Meshes

Figure 2-1: Mesh segmentation results on the full-resolution MIT Animation Dataset [Vla+08]. Each mesh in the dataset contains 20,000 faces (10,000 vertices). We show an example ground truth segmentation in the bottom-left. In contrast to previous works, which downsample each mesh by more than 10×, we efficiently process dense meshes both at train and test time.

We start by considering a deep learning pipeline that operates on 3D geometry as input. We focus on a shape representation that is common in many applications and downstream tasks and for which large datasets are readily available—the triangle mesh. To this end, we propose an encoder that uses machinery from spectral geometry processing to learn features on meshes.

## 2.1 Introduction

Data-driven algorithms have altered the way we approach problem-solving in computer graphics. Machine learning tools garner top performance for tasks like image editing, user interaction,

This chapter includes material from the following publication: [SS21].

image synthesis, and layout, supported by large, well-curated datasets. Yet, while learning tools for areas like computational photography and rendering are widely adopted, another branch of graphics has been resistant to change: mesh-based shape analysis.

Numerous technical challenges preclude modern learning methods from being adopted for meshes. Deep learning—arguably the most popular recent learning methodology—relies on *regularity* of the data and *differentiability* of the objective function for efficiency. For example, convolutional neural network (CNN) training is built on high-throughput processing of images through convolution and per-pixel computations to obtain gradients with respect to network weights, required for stochastic gradient descent.

Meshes, a primary means of representing geometry in graphics, defy the considerations above. They come as sparse, irregular networks of vertices varying in number; the same piece of geometry easily can be represented by multiple meshes and at multiple resolutions/densities. Advances in graph neural networks (GNNs) have as a byproduct helped advance mesh processing, but typical graphs in geometry processing are fundamentally different from those in network science—vertices have low valence, are related through long chains of edges, can be connected in many roughly-equivalent ways, and can be deformed through rigid motions and isometries.

The end result is that mesh-based learning architectures often contort input data to make it compatible with existing learning toolkits. Restricting to GPU-parallel, regularly-structured computation is a vast limitation for mesh analysis. For example, while geometry processing algorithms frequently rely on inversion and eigenanalysis of sparse matrices, these operations are hardly compatible with deep learning. Instead, mesh-based learning algorithms differ from successful non-learning geometry processing algorithms, relying on easily differentiated/parallelized local operations.

In this chapter, we ask whether we can invert this relationship: Rather than inventing new data streams for geometry processing to suit existing learning algorithms, can we develop learning methodologies from successful geometry processing techniques?

We target applications in shape analysis using a prevalent tool in that domain, spectral geometry. Myriad shape analysis algorithms follow a similar template, building a positive (semi)definite matrix whose sparsity pattern is inherited from the mesh and then using its spectral structure to infer information about meshed geometry. Some examples include the Laplacian operator,

the bilaplacian operator, the Dirac operator, and modal analysis. Our broad goal is to *learn* the entries of this operator as functions of local geometry.

Unlike past work, however, we observe that classical shape analysis relies on *near-zero eigenvalues* of these operators (and the corresponding eigenvectors); high-frequency information is discarded. This is a key reason why classical geometry processing algorithms involve sparse matrix inversion and partial computation of eigenvalues. Partial eigenvector computation from a sparse matrix, however, is incompatible with most existing learning pipelines, so learning algorithms that use low-order eigenanalysis typically precompute the relevant eigenvectors from a fixed operator. Approaches to operator learning work with operator-vector products (rather than inverting the operator), restrict to a pre-computed basis, or compute the full spectrum as a dense matrix, which is prohibitive for large meshes.

In this chapter, we approximately differentiate through sparse operator construction for one class of operators motivated by discrete differential geometry. As a result, we can learn operators whose entries are functions of local geometry, which together modify the spectral structure of the operator—a global computation. Our method is competitive with existing mesh-based learning tools while being implemented from standard components of the geometry processing toolkit, and we show how to handle boundary conditions and vector-valued data.

We make some unconventional design decisions that resemble geometry processing rather than deep learning. For instance, our spectral computation and operator construction are implemented using sparse linear algebra on the CPU, and we implement geometry processing-specific strategies for data augmentation that promote resolution independence. These decisions do not hamper efficiency of our method relative to past work.

*Contributions.* We present a lightweight model for learning from triangle meshes, with or without boundary. Contributions include:

- a learnable class of sparse operators on meshes built from standard constructions in discrete exterior calculus;

- parallelizable algorithms for differentiating eigencomputation from these operators, including approximate backpropagation without sparse computation;

- end-to-end architectures for learning per-element or per-mesh features starting from mesh geometry without additional features;
- simple strategies for data augmentation and other practical techniques to improve performance of our method; and
- experiments demonstrating effectiveness in shape analysis tasks, including the generalization of our model to high-resolution meshes that are too dense to be compatible with related methods.

## 2.2 Related work

Machine learning from geometry is becoming a popular subfield of graphics and vision. Bronstein et al. [Bro+17] provide a broad overview of challenges in this discipline; here, we focus on work directly related to our task of learning from meshed geometry.

### 2.2.1 Spectral Shape Analysis

Our method is built on ideas from spectral geometry, which captures shape properties through the lens of spectral (eigenvalue/eigenvector) problems. Wang and Solomon [WS19] provide a comprehensive introduction to this approach to geometry processing.

The *Laplace–Beltrami* (or, *Laplacian*) operator is ubiquitous in spectral geometry processing. Most relevant to our work, numerous per-vertex and per-mesh features have been built from Laplacian eigenvalues and eigenvectors, including the global point signature [Rus07], the heat kernel signature [SOG09], the wave kernel signature [ASC11], and the heat kernel map [Ovs+10]. These descriptors underlie algorithms for tasks as varied as symmetry detection [OSG08], correspondence [Ovs+12], shape recognition [RWP06; BB10], and shape retrieval [Bro+11]—among others.

The Laplacian is popular given its multiscale sensitivity to intrinsic geometry, but recent work proposes replacements sensitive to other aspects of geometry like extrinsic deformation. Examples include the Dirac operator [LJC17; Ye+18], modal analysis [Hil+12; Hua+09], the Hamiltonian [CPK18], the curvature Laplacian [LZ07], the concavity-aware Laplacian [Au+11; Wan+14], the volumetric Laplacian [Rav+10], and the Dirichlet-to-Neumann operator [Wan+18d]. Other

works add invariances to the Laplacian, e.g., to local scaling [BK10] or affine deformation [Rav+11], while others incorporate local features like photometric information [Kov+11; Spa+12]. Nearly all these algorithms—with the notable exception of volumetric methods [Rav+10; Wan+18d]—follow the same outline: Build an operator matrix whose sparsity pattern is inherited from the edges of a triangle mesh and construct features from its eigenvectors and eigenvalues; a widely-used strategy of *truncation* approximates spectral features using partial eigeninformation, usually the eigenvalues closest to 0.

Other spectral methods use or produce *vectorial* data, working with operators that manipulate tangential fields. Vector diffusion operators move information along a manifold or surface while accounting for parallel transport [SW12; SSC19]. The Killing operator also has been applied to intrinsic symmetry detection [Ben+10], segmentation [Sol+11b], deformation [Sol+11a; Cla+17], and registration/reconstruction [CW13; Sla+17]. These methods again analyze a sparse operator built from local features and mesh structure, although there is less agreement on the discretization of operators acting on vector-valued data [GDT16].

Spectral representations of geometry can be "complete" in the sense that a shape's intrinsic structure or embedding can be reconstructed from the eigenvalues and eigenvectors of certain operators. For example, the discrete Laplacian determines mesh edge lengths [Zen+12], and a modified operator adds the extrinsic information needed to obtain an embedding [Cor+17]. [Bos+15a; Cor+17; Cos+19] solve related inverse problems in practice.

Transitioning to the next section, an early machine learning method by Litman and Bronstein [LB13] uses regression to learn spectral descriptors on meshes through learnable functions of Laplacian eigenvalues. This method does not learn the operator but rather the way per-vertex features are constructed from Laplacian eigenvalues. Henaff, Bruna, and LeCun [HBL15] propose a similar approach on graphs.

We attempt to generalize many of the methods above. Rather than defining a "bespoke" operator and mapping from eigeninformation to features for each new task, however, we learn an operator from data.

### 2.2.2 Neural Networks on Meshes

Many papers propose algorithms for learning from meshes and other geometric representations. Here, we summarize past approaches for learning features from meshes, although specialized methods for mesh-based learning appear in tasks like generative modeling [Liu+20; Her+20], meshing [SO20b], and reconstruction [Gao+20; Han+20].

*Learning from graphs.* Since triangle meshes are structured graphs, algorithms for learning from graphs inspired approaches to learning from meshes. Indeed, graph neural networks (GNNs) [KW17] are often used as baselines for geometric learning.

*Spectral domain.* The graph analog of spectral geometry employs Laplacian matrices that act on per-vertex functions. Graph Laplacians provide a linear model for aggregating information between neighboring vertices. Spectral networks [Bru+14] project per-vertex features onto a low-frequency Laplacian eigenbasis before applying a learned linear operator, followed by a per-vertex nonlinearity in the standard basis; convolution on images can be understood as a spectral filter, so these networks generalize image-based convolutional neural networks (CNNs). Subsequent work accelerated learning and inference from spectral networks, often using matrix functions in lieu of computing a Laplacian eigenbasis, e.g., via Chebyshev polynomials [DBV16], random walks [AT16], or rational functions [Lev+18].

*Spatial domain.* Many mesh-based learning methods operate in the "spatial domain," relating vertices to their neighbors through constructions like local parameterization or tangent plane approximation. These methods often can be understood as GNNs with geometrically-motivated edge weights.

Starting with [Mas+15], many methods define convolution-like operations within local neighborhoods by parameterizing vertices and their neighborhoods. A challenge is orienting the convolution kernel, since the tangent plane is different at every point; strategies include taking a maximum over all possible orientations [Mas+15; Sun+20], dynamically computing weights from neighboring features [VBV18], aligning to principal curvatures [Bos+16], learning pseudo-coordinate functions represented as mixtures of Gaussians [Mon+17], projecting onto tangent planes [Tat+18],

sorting nearby vertices based on feature similarity [Wan+18c], aligning to a 4-symmetry field [Hua+19], and weighting by normal vector similarity [Son+20] or directional curvature [He+20].

These and other methods must also define a means of representing localized convolution kernels. Many choices are available, including localized spectral filters [Bos+15b], B-splines [Fey+18], Zernike polynomials [Sun+20], wavelets [SDL18], and extrinsic Euclidean convolution [Sch+20].

Additional machinery is needed to compute vectorial features or relate tangent kernels at different vertices—a problem related to choosing a canonical orientation per vertex. Parallel transport is a choice motivated by differential geometry [Yan+20], which can be combined with circular harmonics [WEH20] or pooling over multiple coordinates [PO18] to avoid dependence on a local coordinate system. Yang et al. [Yan+20] employ locally flat connections for a similar purpose. Mitchel, Kim, and Kazhdan [MKK21] propose to aggregate features by considering each neighbor's own coordinate frame rather than picking a single local parameterization.

Simple GNN layers like [KW17] communicate information only among neighboring vertices. This small receptive field—inherited by several methods above—is a serious challenge for learning from meshes, which are sparse graphs for which a single such layer becomes more and more local as resolution increases. This issue creates dependency of performance on mesh resolution.

*Mesh-based constructions.* While it is valid to interpret meshes as graphs, this perspective neglects the fact that meshes are highly-structured relative to graphs in other disciplines; a few learning algorithms leverage this additional structure to engineer mesh-specific convolutional-style layers. The popular MeshCNN architecture [Han+19] learns edge features and performs pooling based on edge collapse operations. PD-MeshNet [Mil+20a] augments the graph of mesh edges with the graph of dual edges capturing triangle adjacency, with pooling techniques inspired by mesh simplification and dynamic local aggregation using attention.

*Global parameterization.* Surface parameterization is a standard technique for texture mapping; some methods parameterize meshes into an image domain on which standard CNNs can be used for learning and inference from pushed-forward features. Sinha, Bai, and Ramani [SBR16] pioneered this approach using geometry images [GGH02] for parameterization. Maron et al. [Mar+17] use seamless toric covers, conformally mapping four copies of a surface into a flat torus; this work was extended by Haim et al. [Hai+19] to general covers to reduce distortion. Rendering-

based techniques can also be understood as simple parameterizations onto the image plane, e.g., using panoramic [Shi+15; STP17] or multi-view [Su+15; Wei+16; Kal+17] projections.

*Fixed operator methods.* Some methods use operators on surfaces to construct convolution-like operations. Surface Networks [Kos+18b] use discrete Laplacian and Dirac operators as edge weights in GNNs. Yi et al. [Yi+17] define kernels in Laplacian eigenbases, including spectral parameterizations of dilated convolutional kernels and transformer networks. Qiao et al. [Qia+20] use Laplacian spectral clustering to define neighborhoods for pooling.

*Learned operators.* Some past methods learn relevant differential operators to a geometric learning task. Closest to ours, Wang et al. [Wan+19] learn a parameterized sparse operator for geometry processing; see Section 2.4 for comparison of our operator to theirs. Their layers simulate iterations of algorithms, like the conjugate gradient method, by applying their operator, limiting its receptive field to the number of layers. In contrast, we explicitly perform eigendecomposition in our differentiable pipeline, allowing us to engineer the inductive bias inspired by the Hodge Laplacian. Similar discretizations are found in methods like [ET20] for learning PDEs from data; this method uses algebraic multigrid to increase the receptive field.

Sharp et al. [Sha+22] propose a "learned diffusion layer" in which features are diffused along a geometric domain via the isotropic heat equation with learned amount of diffusion; they include diffusion time as a learnable parameter. Similarly to [Bru+14], their diffusion is implemented in a fixed low-frequency Laplacian eigenbasis, computed during learning/inference. Additional features incorporate anisotropy via inner products of spatial gradients. Unlike our work, they use a prescribed Laplacian operator.

*Other.* We mention a few other methods for learning from meshes that do not fall into the categories above. Xu, Dong, and Zhong [XDZ17] present a pipeline that combines purely local and mesh-wise global features; Feng et al. [Fen+19] also propose extracting purely local features. Lim et al. [Lim+18] apply recurrent neural networks (RNNs) to compute vertex features after unrolling local neighborhoods into prescribed spiral patterns. Deep functional maps [Lit+17] largely rely on precomputed features for geometric information, although some recent efforts bring this correspondence method closer to end-to-end [DSO20; SO20a].

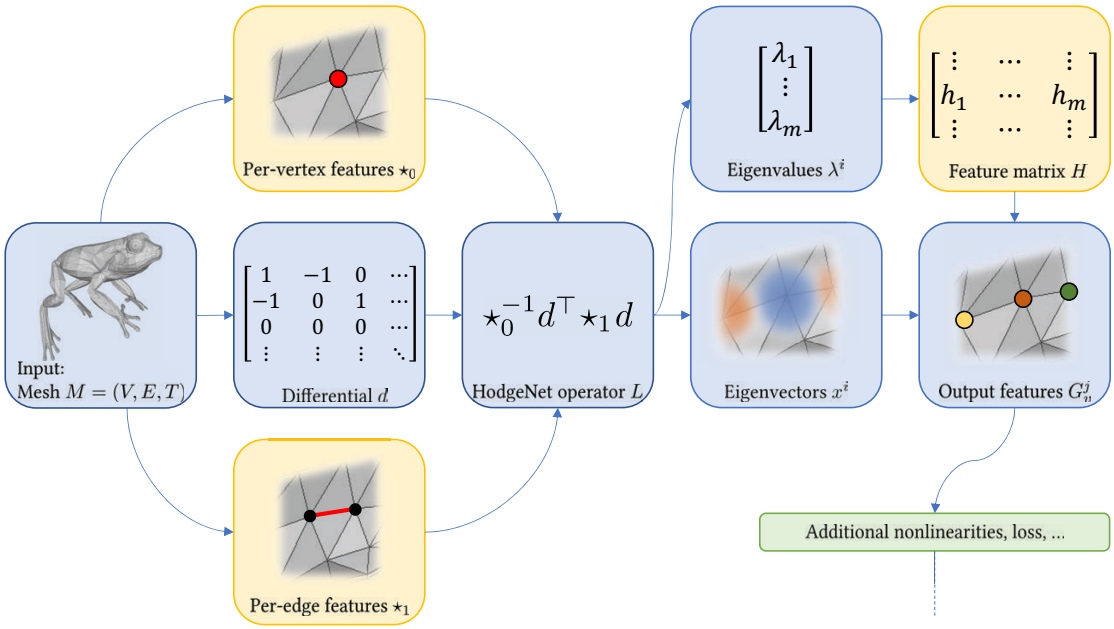



Figure 2-2: Data flow in HodgeNet; yellow boxes contain learnable parameters.

## 2.3 Overview

Figure 2-2 gives an overview of our HodgeNet architecture for learning from triangle meshes. The boxes highlighted in yellow have learnable parameters, while the remaining boxes are fixed computations.

Our goal is to learn an operator and associated spectral descriptor for a given learning-from-meshes task. As with most methods, the learning stage uses stochastic gradient descent to optimize for model parameters, which are fixed during inference.

Our model inputs a triangle mesh $M = (V, E, T)$ and constructs three objects:

- a combinatorial *differential* matrix $d \in \{-1, 0, 1\}^{|E| \times |V|}$,
- a diagonal *0-form Hodge star* matrix $\star_0 \in \mathbb{R}^{|V| \times |V|}$, and
- a diagonal *1-form Hodge star* matrix $\star_1 \in \mathbb{R}^{|E| \times |E|}$.

The matrix $d$ is a fixed function of $M$, while $\star_0, \star_1$ are learnable functions of the local geometry around mesh vertices.

HodgeNet then computes the $k$ eigenvectors $x^i$ of the semidefinite Laplacian-type matrix $L = \star_0^{-1} d^\top \star_1 d$ whose eigenvalues $\lambda^i$ are closest to zero. Finally, per-vertex or per-mesh features are gathered from $\{(x^i, \lambda^i)\}$ using learnable formulas that generalize the form of popular spectral features like the heat kernel signature.

During training, we need to differentiate our loss function through the steps above. Most of the operations above are simple nonlinearities that can be differentiated using standard backpropagation methods. We show in Section 2.5 how to obtain approximate derivatives of the eigenproblem efficiently.

## 2.4 Operator construction

HodgeNet relies on a parameterized class of learnable operators whose entries are functions of local geometry. The basis of our construction, designed to encapsulate operator constructions in spectral geometry, resembles that proposed by Wang et al. [Wan+19], with the key difference that we expose per-edge and per-vertex features using diagonal Hodge star operators; this difference greatly simplifies our backpropagation procedure in Section 2.5. In Section 2.4.2, we also show how to generalize this construction for vectorial operators.

### 2.4.1 Operator

Given an oriented manifold mesh $\mathcal{M} = (V, E, T)$ (optionally with boundary) with vertices $V \subset \mathbb{R}^3$, edges $E \subset V \times V$, and oriented triangles $T \subset V \times V \times V$, HodgeNet constructs a positive (semi)definite operator matrix $L \in \mathbb{R}^{|V| \times |V|}$ whose spectral structure will be used for a mesh-based learning task.

Inspired by the factorization of the Laplacian in discrete exterior calculus [Des+05], we parameterize $L$ as a product:

$$L = \star_0^{-1} d^\top \star_1 d. \tag{2.1}$$

Here, $d \in \{-1, 0, 1\}^{|E| \times |V|}$ is the differential operator given by

$$d_{ev} = \begin{cases} 1 & \text{if } v = v_2 \\ -1 & \text{if } v = v_1 \\ 0 & \text{otherwise,} \end{cases}$$

where $e = (v_1, v_2)$ is an oriented edge.

While $d$ is determined by mesh topology, the diagonal Hodge star matrices $\star_0 \in \mathbb{R}^{|V| \times |V|}$ and $\star_1 \in \mathbb{R}^{|E| \times |E|}$ are learnable functions of local mesh geometry. To construct $\star_0, \star_1$, we input $D$ per-vertex features $F \in \mathbb{R}^{|V| \times D}$. In our experiments, we use positions and normals as the per-vertex features, except when noted otherwise. We detail the construction of these operators $\star_0(F), \star_1(F)$ from $F$ below.

*Per-vertex features* $\star_0$. Our construction of $\star_0(F)$ imitates area weight computation for discrete Laplacians. It takes place in two steps. First, we compute *per-triangle* features using a learnable function $f_\Phi : \mathbb{R}^{3D} \to \mathbb{R}$, where $\Phi$ contains the parameters of our model. To ensure positive (semi)definiteness for $\star_0(F)$ we square $f_\Phi$. Finally, we gather features from triangles to vertices by summing and optionally adding a small constant $\varepsilon$ (in practice, $\varepsilon = 10^{-4}$) to improve conditioning of the eigensystem. Overall, we can write our expression as follows:

$$(\star_0(F))_{vv} = \varepsilon + \sum_{t \sim v} f_\Phi(F_{v_1}, F_{v_2}, F_{v_3})^2 \tag{2.2}$$

where $t = (v_1, v_2, v_3) \in T$ is a triangle with vertices $v_1, v_2, v_3$ in counterclockwise order. This sum over $t$ has a potentially different number of terms for each vertex, equal to the valence.

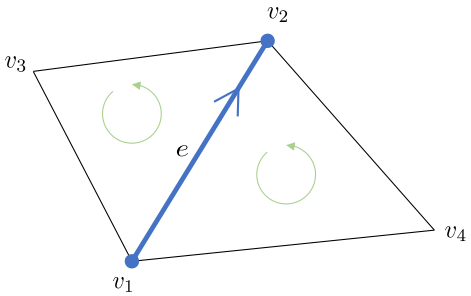
If $\varepsilon = 0$ and $f_\Phi^2$ measures triangle area scaled by $1/3$, then $\star_0$ becomes the barycentric area weights matrix often used in finite elements and discrete exterior calculus. We give the details of our choice of functions $f_\Phi$ in Section 2.7. Squaring the inner part of (2.2) is one of many ways to make sure $(\star_0)_{vv} \geq 0$ and could be replaced, e.g., by ReLU activation, but we found empirically that this simple expression led to the best performance.

*Per-edge features* $\star_1$. The diagonal matrix $\star_1(F)$ contains per-edge features on its diagonal. Unlike (2.2), to compute $\star_1(F)$ we do not need to gather features from a variable-sized ring. Instead,

we learn a function $g_\Phi : \mathbb{R}^{4D} \to \mathbb{R}$ and, for an interior edge $e = (v_1, v_2)$, compute

$$(\star_1(F))_{ee} = \varepsilon + g_\Phi(F_{v_1}, F_{v_2}, F_{v_3}, F_{v_4})^2, \tag{2.3}$$

where $v_3$ and $v_4$ are *opposite* the edge $e$ as shown to the right. We order $v_3$ and $v_4$ so that $(v_1, v_2, v_3)$ and $(v_2, v_1, v_4)$ are all consistently oriented. We learn a separate function $\bar{g}_\Phi(v_1, v_2, v_3)$ for boundary edges, since there is only one opposite vertex in this case.




If $\varepsilon = 0$ and $g_\Phi^2$ gives the sum of interior angle cotangents at $v_3$ and $v_4$, then $\star_1$ will be the famous cotangent Laplacian matrix common in geometry processing. While we have chosen to square the function $g$, thanks to conjugation by $d$ in (2.1) this is sufficient but not necessary for positive (semi)definiteness of $L$, and indeed this design choice prevents us from exactly reproducing the cotangent Laplacian in the presence of obtuse triangles. Our architecture could easily be adjusted to allow for negative $(\star_1)_{ee}$ values and hence to reproduce the cotangent Laplacian operator, but the stability and ease of squaring $g_\Phi$ to ensure that $L$ has no negative eigenvalues outweighed this largely theoretical consideration.

*Discussion.* Our parameterizations of $L$, $\star_0$, and $\star_1$ imitate the flow of information used to construct discrete Laplacian operators and related objects. They are readily incorporated into geometry processing pipelines and have familiar sparsity patterns encountered in this discipline.

It is worth acknowledging a few design decisions intended to simplify our framework at the cost of mathematical structure:

- Squaring $g_\Phi$ in (2.3) means we cannot reproduce the cotangent Laplacian operator for poorly-conditioned meshes with negative cotangent weights.
- We arbitrarily choose one of three possible cyclic orderings of the inputs to $f_\Phi$ in (2.2).
- Similarly, we arbitrarily choose among two orderings of the inputs to $g_\Phi$ in (2.3): $(v_1, v_2, v_3, v_4)$ and $(v_2, v_1, v_4, v_3)$.

All three items above could be addressed at the cost of increasing the complexity of $f_\Phi, g_\Phi$, but

building more general semidefiniteness conditions and/or order invariance did not bring practical benefit.

### 2.4.2 Vectorial operators

$L$ discretizes operators that act on functions discretized using one value per vertex of a triangle mesh. We also can discretize operators acting on *vector-valued* functions with a value in $\mathbb{R}^k$ per vertex by adjustmenting our construction. For example, for planar triangle meshes and $k = 2$ we can reproduce the Killing operator described in [Sol+11a; Cla+17]; for $k = 4$ we can mimic the Dirac operator used for shape analysis by Liu, Jacobson, and Crane [LJC17].

To extend to vectorial operators, we use a $k|E| \times k|V|$ block version of $d$ whose blocks are given as follows:

$$d_{ev} = \begin{cases} I_{k\times k} & \text{if } v = v_2 \\ -I_{k\times k} & \text{if } v = v_1 \\ 0 & \text{otherwise,} \end{cases}$$

where $I_{k\times k}$ denotes the $k \times k$ identity matrix.

We generalize $f_\Phi : \mathbb{R}^{3D} \to \mathbb{R}^{k\times k}$ and $g_\Phi : \mathbb{R}^{4D} \to \mathbb{R}^{k\times k}$ to output $k \times k$ matrices. Then, we compute $\star_0$ and $\star_1$ as block diagonal matrices whose elements are as follows:

$$(\star_0)_{vv} = \varepsilon I_{k\times k} + \sum_{t\sim v} f_\Phi(F_{v_1}, F_{v_2}, F_{v_3})^\top f_\Phi(F_{v_1}, F_{v_2}, F_{v_3}) \tag{2.4}$$

$$(\star_1)_{ee} = \varepsilon I_{k\times k} + g_\Phi(F_{v_1}, F_{v_2}, F_{v_3}, F_{v_4})^\top g_\Phi(F_{v_1}, F_{v_2}, F_{v_3}, F_{v_4}). \tag{2.5}$$

These definitions generalize our scalar construction for the case of $k = 1$ and still lead to a semidefinite matrix $L = \star_0^{-1} d^\top \star_1 d \in \mathbb{R}^{k|V|\times k|V|}$.

## 2.5 Differentiable spectral analysis

Now that we have determined the form of our operator $L$, we turn the task of using its low-order eigenvalues and eigenvectors for learning tasks. The key challenge will be to differentiate through eigenvalue/eigenvector computation, a task we consider below. While general eigencomputation

is extremely difficult for learning, we show how our particular form (2.1) for $L$ facilitates backpropagation and reduces dependence on random-access computation.

Recall that a step of training requires evaluating the loss function $\mathscr{L}$ and its gradients with respect to the parameters $\Phi$ of the model. The loss function is evaluated in a *forward pass*, and the gradients are evaluated during *backpropagation*. We will perform both the forward and backward pass of our model on the CPU so as to take advantage of a sparse solver to compute a set of eigenvalues/eigenvectors for the operator $L$ efficiently. While the computation of our features for our learning problem as well as the entirety of backpropogation could be efficiently computed on the GPU, our model has sufficiently few parameters that we find it unnecessary to transfer data between GPU and CPU.

### 2.5.1 The HodgeNet generalized eigenproblem

Our architecture outputs features built from eigenvectors of $L$ in (2.1). Recall that $L$—and, in particular, the Hodge stars $\star_0, \star_1$—is built from a matrix $F \in \mathbb{R}^{|V| \times D}$ of per-vertex features:

$$L = \star_0(F)^{-1} d^\top \star_1(F) d.$$

Hence, our features are built from eigenvectors $x^i \in \mathbb{R}^{k|V|}$ satisfying

$$Lx^i = \lambda^i x^i \iff d^\top \star_1 dx^i = \lambda^i \star_0 x^i. \tag{2.6}$$

By construction, $d^\top \star_1 d \geq 0$ and $\star_0 \geq$, so $\lambda^i \geq 0$. By convention, we normalize eigenvectors to satisfy the condition $(x^i)^\top \star_0 x^j = \delta_{ij}$, possible thanks to symmetry of our operators.

To differentiate our per-vertex features, we need to differentiate the eigenvectors $x^i$ and eigenvalues $\lambda^i$ with respect to the parameters $\Phi$ of our learned functions $f_\Phi, g_\Phi$. The expressions in Section 2.4 for $(\star_0)_{vv}(F)$ and $(\star_1)_{ee}(F)$ are readily differentiated. Hence, for compatibility with the backpropagation algorithm for differentiation, we need to solve the following problem involving our loss function $\mathscr{L}$:

**Given the partial derivatives $\partial\mathscr{L}/\partial\lambda^i$ and $\partial\mathscr{L}/\partial x^i_j$ for all $i, j$, compute the partial derivatives $\partial\mathscr{L}/\partial(\star_0)_{vv}$ and $\partial\mathscr{L}/\partial(\star_1)_{ee}$ for all $v \in V, e \in E$.**

In words, given derivatives of the loss function with respect to the eigenvalues and eigenvectors of $L$, compute the derivatives of the loss function with respect to the Hodge stars.

In general, differentiating through eigenvalue problems is expensive. Libraries like TensorFlow and PyTorch allow for differentiation of computing the *full* spectrum of a matrix, but their implementations (1) cannot account for the sparsity structure of our mesh and (2) cannot target a few eigenvalues close to 0, which are typically the meaningful eigenvalues to compute in geometry processing applications. Solving the full eigenvalue problem is extremely expensive computationally, and storing a $k|V| \times k|V|$ matrix of eigenvectors is prohibitive.

Our pipeline addresses the issues above. We rely on CPU-based sparse eigensolvers during the forward pass of our network, solving (2.6) only for a subset of eigenvalues. This alleviates dependence on $k|V| \times k|V|$ dense matrices and instead only stores the $O(k|V|)$ nonzero entries.

### 2.5.2 Derivative formulas

The vectorial operator $L$ operates on vectors in $\mathbb{R}^k$ per vertex on a mesh. Following Section 2.4.2, we will use $x^i_{v\ell}$ to refer to the $\ell$-th element ($\ell \in \{1, \dots, k\}$) of entry $v$ ($v \in V$) of the $i$-th eigenvector of $L$. We use $\star_{0v\ell m}$ to refer to the element $(\ell, m)$ of the $k \times k$ block of $\star_0$ at vertex $v \in V$. More generally, we will use subscripts to refer to matrix elements and superscripts to index over eigenvalues.

Define the following tensors:

$$y^i_{e\ell} := \sum_{v \in V} d_{ev} x^i_{v\ell}$$

$$M_{ij} := \begin{cases} (\lambda^i - \lambda^j)^{-1} & \text{if } i \neq j \\ 0 & \text{otherwise.} \end{cases}$$

$$N_{ij} := \begin{cases} \lambda^i / (\lambda^j - \lambda^i) & \text{if } i \neq j \\ -1/2 & \text{otherwise.} \end{cases}$$

We compute $y$ during the forward pass as $dx$ and cache the result for use during backpropagation, since $d$ is a sparse matrix.

Our algorithm relies on the following proposition:

**Proposition 1** *We can backpropagate derivatives of our loss function as follows:*

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial \star_{0w\ell m}} &= -\sum_i \frac{\partial \mathscr{L}}{\partial \lambda^i} \lambda^i x^i_{w\ell} x^i_{wm} + \sum_{ivnj} \frac{\partial \mathscr{L}}{\partial x^i_{vn}} N_{ij} x^j_{w\ell} x^i_{wm} x^j_{vn} \\ \frac{\partial \mathscr{L}}{\partial \star_{1e\ell m}} &= \sum_i \frac{\partial \mathscr{L}}{\partial \lambda^i} y^i_{e\ell} y^i_{em} + \sum_{ivnj} \frac{\partial \mathscr{L}}{\partial x^i_{vn}} M_{ij} x^j_{vn} y^j_{e\ell} y^i_{em}\end{aligned} \tag{2.7}$$

*Here, $i, j$ index over the eigenvectors of $L$; $\ell, n, m$ index over vector elements from 1 to $k$; $v, w$ are vertices of the mesh; and $e$ is an edge.*

We provide a proof in the following section.

The expressions in (2.7) may appear complicated, but in reality they are efficiently computable. We have eliminated all sparse matrices and inverses from these formulas, which are readily implemented using a one-line call to Einstein summation functions in deep learning toolkits (e.g., `einsum` in PyTorch).

### 2.5.3 Derivation of derivative formulas

As input, assume we have a matrix $d \in \mathbb{R}^{|E|\times|V|}$ that subtracts per-vertex values along mesh edges. Our goal is to learn two operators $\star_0 \in \mathbb{R}^{|V|\times k\times k}$ and $\star_1 \in \mathbb{R}^{|E|\times k\times k}$, where $k$ is the dimensionality of the vectorial data.

Suppose $x \in \mathbb{R}^{|V|\times k}$ is a $k$-valued function on our domain. Then, the weak form of our Laplace-type operator acting on $x$ can be written $x \in \mathbb{R}^{|V|\times k} \mapsto \mathscr{L}[x] \in \mathbb{R}^{|V|\times k}$ where

$$\mathscr{L}[x]_{v\ell} = \sum_{emw} d_{ev} \star_{1e\ell m} d_{ew} x_{wm}. \tag{2.8}$$

Similarly, we can define a mass operator as

$$\mathscr{M}[x]_{v\ell} = \sum_m \star_{0v\ell m} x_{vm}. \tag{2.9}$$

Throughout this document, we will assume the symmetry properties

$$\star_{0v\ell m} = \star_{0vm\ell} \qquad \text{and} \qquad \star_{1e\ell m} = \star_{1em\ell}. \tag{2.10}$$

In a forward pass, we compute a set of eigenvectors $x^i$ and eigenvalues $\lambda^i$ satisfying

$$\mathscr{L}[x^i] = \lambda^i \mathscr{M}[x^i]. \tag{2.11}$$

Or equivalently, plugging in (2.8) and (2.9), for all $i, v, \ell$ we have

$$\sum_{emw} d_{ev} \star_{1e\ell m} d_{ew} x^i_{wm} = \lambda^i \sum_m \star_{0v\ell m} x^i_{vm} \tag{2.12}$$

Because our operators are self-adjoint, we have the following orthogonality relationship:

$$\sum_{v\ell} x^j_{v\ell} \mathscr{M}[x^i]_{v\ell} = \delta_{ij}. \tag{2.13}$$

Or equivalently, plugging in (2.9),

$$\sum_{v\ell m} x^j_{v\ell} \star_{0v\ell m} x^i_{vm} = \delta_{ij}. \tag{2.14}$$

Suppose our learning problem as an loss function $\mathscr{L}$. During the process of back propagation, we require derivatives

$$\frac{\partial \mathscr{L}}{\partial x^i_{vj}} \quad \text{and} \quad \frac{\partial \mathscr{L}}{\partial \lambda^i}. \tag{2.15}$$

In this document, we show how to obtain

$$\frac{\partial \mathscr{L}}{\partial \star_{0v\ell m}} \quad \text{and} \quad \frac{\partial \mathscr{L}}{\partial \star_{1e\ell m}}. \tag{2.16}$$

*Useful identities.* We start by deriving some general formulas. We will use prime to denote differentiation with respect to a scalar we will determine later; note $\mathscr{L}$ and $\mathscr{M}$ are functions of our unknowns in the learning problem so we need to be careful taking derivatives.

We'll first differentiate (2.12). For all $i, v, \ell$ we have

$$\begin{aligned} 0 &= \left( \sum_{emw} d_{ev} \star_{1e\ell m} d_{ew} x^i_{wm} - \lambda^i \sum_m \star_{0v\ell m} x^i_{vm} \right)' \\ &= \sum_{emw} \left( d_{ev} \star'_{1e\ell m} d_{ew} x^i_{wm} + d_{ev} \star_{1e\ell m} d_{ew} (x^i_{wm})' \right) - \sum_m \left( (\lambda^i)' \star_{0v\ell m} x^i_{vm} + \lambda^i \star'_{0v\ell m} x^i_{vm} + \lambda^i \star_{0v\ell m} (x^i_{vm})' \right) \end{aligned}$$

We multiply both sides by $x^j_{v\ell}$ and sum over $v, \ell$ to obtain the following:

$$
\begin{aligned}
0 &= \sum_{emwv\ell} x^j_{v\ell} \left( d_{ev} \star'_{1e\ell m} d_{ew} x^i_{wm} + d_{ev} \star_{1e\ell m} d_{ew} (x^i_{wm})' \right) \\
&\quad - \sum_{mv\ell} x^j_{v\ell} \left( (\lambda^i)' \star_{0v\ell m} x^i_{vm} + \lambda^i \star'_{0v\ell m} x^i_{vm} + \lambda^i \star_{0v\ell m} (x^i_{vm})' \right) \\
&= \sum_{emwv\ell} x^j_{v\ell} \left( d_{ev} \star'_{1e\ell m} d_{ew} x^i_{wm} + d_{ev} \star_{1e\ell m} d_{ew} (x^i_{wm})' \right) \\
&\quad - (\lambda^i)' \delta_{ij} - \lambda^i \sum_{mv\ell} x^j_{v\ell} \left( \star'_{0v\ell m} x^i_{vm} + \star_{0v\ell m} (x^i_{vm})' \right) \text{ by (2.14)} \\
&= \sum_{emwv\ell} x^j_{v\ell} d_{ev} \star'_{1e\ell m} d_{ew} x^i_{wm} + \sum_{v\ell} (x^i_{v\ell})' \sum_{emw} d_{ev} \star_{1e\ell m} d_{ew} x^j_{wm} \\
&\quad - (\lambda^i)' \delta_{ij} - \lambda^i \sum_{mv\ell} x^j_{v\ell} \left( \star'_{0v\ell m} x^i_{vm} + \star_{0v\ell m} (x^i_{vm})' \right) \text{ reindexing/reshuffling the second term} \\
&= \sum_{emwv\ell} x^j_{v\ell} d_{ev} \star'_{1e\ell m} d_{ew} x^i_{wm} + \lambda^j \sum_{v\ell m} (x^i_{v\ell})' \star_{0v\ell m} x^j_{vm} \\
&\quad - (\lambda^i)' \delta_{ij} - \lambda^i \sum_{mv\ell} x^j_{v\ell} \left( \star'_{0v\ell m} x^i_{vm} + \star_{0v\ell m} (x^i_{vm})' \right) \text{ by (2.12)} \\
&= \sum_{emwv\ell} x^j_{v\ell} d_{ev} \star'_{1e\ell m} d_{ew} x^i_{wm} + (\lambda^j - \lambda^i) \sum_{v\ell m} (x^i_{v\ell})' \star_{0v\ell m} x^j_{vm} \\
&\quad - (\lambda^i)' \delta_{ij} - \lambda^i \sum_{mv\ell} x^j_{v\ell} \star'_{0v\ell m} x^i_{vm} \text{ after reindexing the last term} \\
&= \sum_{em\ell} y^j_{e\ell} \star'_{1e\ell m} y^i_{em} + (\lambda^j - \lambda^i) \sum_{v\ell m} (x^i_{v\ell})' \star_{0v\ell m} x^j_{vm} - (\lambda^i)' \delta_{ij} - \lambda^i \sum_{mv\ell} x^j_{v\ell} \star'_{0v\ell m} x^i_{vm}, \qquad (2.17)
\end{aligned}
$$

where we define

$$
y^j_{e\ell} := \sum_v d_{ev} x^j_{v\ell}. \qquad (2.18)
$$

This leads us to a useful identity:

$$
(\lambda^i)' \delta_{ij} + (\lambda^i - \lambda^j) \langle (x^i)', x^j \rangle_0 = \sum_{ew} y^j_{e\ell} \star'_{1e\ell m} y^i_{em} - \lambda^i \sum_{v\ell m} x^j_{v\ell} \star'_{0v\ell m} x^i_{vm}, \qquad (2.19)
$$

where $\langle \cdot, \cdot \rangle_0$ denotes an inner product with respect to $\star_0$.

For one more convenient formula, we can differentiate (2.14) in the case $i = j$:

$$\begin{aligned} 0 &= \sum_{v\ell m} (x^i_{v\ell} \star_{0v\ell m} x^i_{vm})' \\ &= \sum_{v\ell m} (x^i_{v\ell} \star'_{0v\ell m} x^i_{vm} + 2x^i_{v\ell} \star_{0v\ell m} (x^i_{vm})') \text{ by symmetry of } \star_0 \\ \implies \langle x^i, (x^i)' \rangle_0 &= -\frac{1}{2} \sum_{v\ell m} x^i_{v\ell} \star'_{0v\ell m} x^i_{vm} \end{aligned} \tag{2.20}$$

*Derivatives w.r.t. $\star_1$.* We start with the $\star_1$ operator. By the chain rule, we know

$$\frac{\partial \mathscr{L}}{\partial \star_{1e\ell m}} = \sum_i \frac{\partial \mathscr{L}}{\partial \lambda^i} \frac{\partial \lambda^i}{\partial \star_{1e\ell m}} + \sum_{ivn} \frac{\partial \mathscr{L}}{\partial x^i_{vn}} \frac{\partial x^i_{vn}}{\partial \star_{1e\ell m}} \tag{2.21}$$

Consider (2.19) with $i = j$ and differentiating with respect to $\star_{1e\ell m}$. We find:

$$\frac{\partial \lambda^i}{\partial \star_{1e\ell m}} = y^i_{e\ell} y^i_{em} \tag{2.22}$$

Differentiating the eigenvectors is more difficult. We use a projection formula:

$$\begin{aligned} \frac{\partial x^i_{vn}}{\partial \star_{1e\ell m}} &= \sum_j \left\langle \frac{\partial x^i}{\partial \star_{1e\ell m}}, x^j \right\rangle_0 x^j_{vn} \text{ by writing in the } x \text{ basis} \\ &= \sum_{j \neq i} \left\langle \frac{\partial x^i}{\partial \star_{1e\ell m}}, x^j \right\rangle_0 x^j_{vn} \text{ by (2.20)} \\ &= \sum_j M_{ij} x^j_{vn} y^j_{e\ell} y^i_{em} \text{ by (2.19) with } i \neq j, \end{aligned} \tag{2.23}$$

where

$$\boxed{M_{ij} = \begin{cases} (\lambda_i - \lambda_j)^{-1} & \text{if } i \neq j \\ 0 & \text{otherwise.} \end{cases}} \tag{2.24}$$

Plugging these expressions into (2.21),

$$\boxed{\frac{\partial \mathscr{L}}{\partial \star_{1e\ell m}} = \sum_i \frac{\partial \mathscr{L}}{\partial \lambda^i} y^i_{e\ell} y^i_{em} + \sum_{ivnj} \frac{\partial \mathscr{L}}{\partial x^i_{vn}} M_{ij} x^j_{vn} y^j_{e\ell} y^i_{em}} \tag{2.25}$$

*Derivatives w.r.t. $\star_0$.* Now, we consider the $\star_0$ operator. By the chain rule, we have

$$\frac{\partial \mathscr{L}}{\partial \star_{0w\ell m}} = \sum_i \frac{\partial \mathscr{L}}{\partial \lambda^i} \frac{\partial \lambda^i}{\partial \star_{0w\ell m}} + \sum_{ivn} \frac{\partial \mathscr{L}}{\partial x^i_{vn}} \frac{\partial x^i_{vn}}{\partial \star_{0w\ell m}}. \tag{2.26}$$

Consider (2.19) with $i = j$ and differentiating with respect to $\star_{0w\ell m}$. We find:

$$\frac{\partial \lambda^i}{\partial \star_{0w\ell m}} = -\lambda^i x^i_{w\ell} x^i_{wm} \tag{2.27}$$

Differentiating the eigenvectors is again more difficult. We use a projection formula:

$$\begin{aligned}
\frac{\partial x^i_{vn}}{\partial \star_{0w\ell m}} &= \sum_j \left\langle \frac{\partial x^i}{\partial \star_{0w\ell m}}, x^j \right\rangle_0 x^j_{vn} \text{ by writing in the } x \text{ basis} \\
&= -\frac{1}{2} x^i_{w\ell} x^i_{wm} x^i_{vn} + \sum_{j \neq i} \left\langle \frac{\partial x^i}{\partial \star_{0w\ell m}}, x^j \right\rangle_0 x^j_{vn} \text{ by (2.20)} \\
&= -\frac{1}{2} x^i_{w\ell} x^i_{wm} x^i_{vn} - \sum_{j \neq i} M_{ij} \lambda^i x^j_{w\ell} x^i_{wm} x^j_{vn} \text{ by (2.19) with } i \neq j \\
&= \sum_j N_{ij} x^j_{w\ell} x^i_{wm} x^j_{vn}
\end{aligned} \tag{2.28}$$

where

$$N_{ij} = \begin{cases} -\frac{\lambda^i}{\lambda^i - \lambda^j} & \text{if } i \neq j \\ -\frac{1}{2} & \text{otherwise.} \end{cases} \tag{2.29}$$

Plugging these expressions into (2.26),

$$\frac{\partial \mathscr{L}}{\partial \star_{0w\ell m}} = -\sum_i \frac{\partial \mathscr{L}}{\partial \lambda^i} \lambda^i x^i_{w\ell} x^i_{wm} + \sum_{ivnj} \frac{\partial \mathscr{L}}{\partial x^i_{vn}} N_{ij} x^j_{w\ell} x^i_{wm} x^j_{vn} \tag{2.30}$$

### 2.5.4 Derivative approximation

Here we briefly address one challenge using Proposition 1 to differentiate HodgeNet. Recall from Section 2.5.1 that we compute an incomplete set of eigenvectors of $L$, far fewer than the largest possible number. This choice is reasonable for constructing a loss function, which will only depend on this low-order eigenstructure. However, (2.7) requires *all* eigenvectors of $L$ to evaluate the sums over $j$.

We use a simple strategy to address this issue. During the forward pass we compute and cache more eigenvalues/eigenvectors than are needed to evaluate $\mathscr{L}$; in practice, we use 2× (see Section 2.8.5 for validation). Then, in backpropagation we truncate the sums over $j$ in (2.7) to include only these terms. A straightforward argument reveals that the resulting gradient approximation still yields a descent direction for $\mathscr{L}$.

The first term in each sum is computable from exclusively the partial set of eigenvalues, implying we can exactly differentiate $\mathscr{L}$ with respect to the eigenvalues $\lambda^i$; our approximation is only relevant to the eigenvectors.

## 2.6 From eigenvectors to features

Recall that our broad task is to design a learnable mapping from meshes to task-specific features. So far, we have designed a learnable operator from mesh geometry and provided a means of differentiating through its eigenvectors/eigenvalues. It is tempting to use the eigenvectors as per-vertex features, but this is not a suitable choice: The choice of sign $\pm x_i$ for each eigenvector is arbitrary.

We return to geometry processing for inspiration. Classical shape descriptors built from operator eigenfunctions circumvent the sign issue by *squaring* the Laplacian eigenfunctions pointwise. For instance, the heat kernel signature [SOG09], wave kernel signature [ASC11], and general learned kernels [LB13] take the form

$$\sum_i f(\bar{\lambda}^i)\psi^i(p)^2,$$

where $\bar{\lambda}^i$ is the $i$-th eigenvalue and $\psi^i$ is the $i$-ith eigenvector of the Laplacian. The fact that $\psi^i$ is squared alleviates sign dependence. Similarly, for eigenfunctions of the vectorial *vector Laplacian*, sign-agnostic features can be computed from the outer product of the pointwise vector and itself $\psi^i(p)\psi^i(p)^\top \in \mathbb{R}^{k\times k}$ (see, e.g., [SW12, eq. (3.13)]).

Generalizing the hand-designed features above, we construct a sign-agnostic learnable per-vertex feature as follows. Take $m$ to be the number of eigenvectors of $L$ we will use to compute features, and take $n$ to be the number of output features. We learn a function $h_\Phi : \mathbb{R} \to \mathbb{R}^n$ and construct a matrix $H \in \mathbb{R}^{m\times n}$ whose columns are $h(\lambda^i)$ for $i \in \{1, \dots, m\}$. Then, for $j \in \{1, \dots, n\}$,

the $j$-th output feature at vertex $v \in V$, notated $G_v^j$, is given by:

$$G_v^j := \sum_i H_{ij} \cdot (x_v^i)(x_v^i)^\top,$$

where $x_v^i$ denotes the $i$-th eigenvector of $L$ evaluated at vertex $v$ as a $k \times 1$ column vector. We omit the 0 eigenvalue corresponding to the constant eigenfunction. We give our form for $h_\Phi$ in Section 2.7.

Having computed per-vertex features $G_v$, we optionally follow a standard max pooling approach to obtain per-face features

$$G_f = \max_{v \sim f} G_v,$$

or per-mesh features

$$G_M = \max_{v \in V} G_v$$

depending on the learning task at hand. We map these features to the desired output dimensions $d$ using a learned function $o_\Phi : \mathbb{R}^n \to \mathbb{R}^d$.

## 2.7 Additional details and parameters

We model each of $f_\Phi, g_\Phi, h_\Phi, o_\Phi$ as an MLP with batch normalization and Leaky ReLU [MHN13] before each hidden layer. $f_\Phi$ and $g_\Phi$ each consist of four hidden layers, each of size 32; $h_\Phi$ consists of four hidden layers, each of size $n$; and $o_\Phi$ consists of two hidden layers, each of size 32, except for the classification task, where the layers have 64 units. In all experiments, we set vector dimensionality $k = 4$, output feature size $n = 32$, and number of eigenpairs used $m = 32$. We use an additional 32 eigenpairs for improved derivative approximation, as described in Section 2.5.4.

We train our network using the optimizer AdamW [LH19] with a batch size of 16 and learning rate of 0.0001. We use gradient clipping with maximum norm of 1.0 to stabilize training. We implement our pipeline in PyTorch, using SciPy `eigsh` with ARPACK for solving our sparse eigenproblem and `libigl` for mesh processing. We train our models on 128 2.5 GHz CPUs.

## 2.8 Experiments

We demonstrate the efficacy of our method on several shape analysis tasks and provide experiments justifying some of our parameter and design choices. We also compare to state-of-the-art methods developed for learning on meshes. Other geometric deep learning approaches tend to use GNNs, ignoring the mesh structure and relying on multiple layers to aggregate global data, whereas our method uses spectral geometry to infer global information from local features.

### 2.8.1 Mesh segmentation

We train our network for mesh segmentation on four datasets—the Human Body dataset [Mar+17] and the vase, chair, and alien categories of the Shape COSEG dataset [Wan+12]—optimizing cross entropy loss. We use the same version of the Human Body dataset as in [Han+19; Mil+20a], which is downsampled to 2000 faces per mesh. We evaluate on the test set of the Human Body dataset, and generate a random 85%-15% train-test split for each Shape COSEG category, as in [Han+19; Mil+20a]. We train for 100 epochs (~ 3 hours), randomly decimating each input mesh to a resolution of 1000-2000 faces and randomly applying anisotropic scaling of up to 5% in each dimension. We then fine-tune by training for 100 more epochs without decimation or scaling. In the case of the Human Body dataset, where meshes are not canonically rotated, we also apply random rotations as data augmentation. We center each mesh about the vertex center of mass and rescale to fit inside the unit sphere.

Figure 2-3: Segmentation results on the Shape COSEG dataset. Meshes shown are randomly selected from the test set for each category.

We report segmentation accuracies in Tables 2.1 and 2.3 and area-weighted segmentation accuracies in Tables 2.2 and 2.4. For fair comparison, as in [Mil+20a], we report accuracies based on "hard" ground-truth segmentation face labels for MeshCNN [Han+19] rather than "soft" edge labels; see [Mil+20a, Supplementary Material, Section H] for details regarding the segmen-

Table 2.1: Segmentation accuracy on the Human Body test set.

| Method | # Parameters | Accuracy |
|---|---|---|
| Ours | 31,720 | 85.03% |
| PD-MeshNet [Mil+20a] | 173,728 | 85.61% |
| MeshCNN [Han+19] | 2,279,720 | 85.39% |

Table 2.2: Area-weighted segmentation accuracy on Human Body test set.

| Method | Accuracy |
|---|---|
| Ours | 86.48% |
| PD-MeshNet [Mil+20a] | 86.45% |

tation metrics. Our method obtains results comparable to state-of-the-art for each dataset while requiring significantly fewer learnable parameters. We also show our learned segmentations on the Human Body dataset in Fig. 2-4 and on Shape COSEG in Fig. 2-3.

### 2.8.2 High-resolution mesh segmentation

In contrast to earlier works, our method is capable of training on dense, non-decimated mesh data. We demonstrate this by training a segmentation model on the MIT Animation Dataset [Vla+08], where each mesh contains 20,000 faces. We pre-initialize our model with the segmentation model trained on the Human Body dataset above and train for an additional 30 epochs (~ 4 hours). The pre-initialization allows us to avoid training a model from scratch: our model trained on low-resolution meshes captures some triangulation-invariant features, making this transfer learning possible. We train on five random 95%-5% train-test splits and achieve 89.34% mean accuracy. We report accuracies for each split in Table 2.5 and render Split 1 in Fig. 2-1 and Split 2 in Fig. 2-5.

Table 2.5: Segmentation accuracies for random splits of the full-resolution MIT Animation Dataset [Vla+08].

| Split # | Accuracy |
|---|---|
| 1 | 90.57% |
| 2 | 86.90% |
| 3 | 90.02% |
| 4 | 89.07% |
| 5 | 90.15% |

### 2.8.3 Mesh classification

We evaluate our method on mesh classification on the downsampled version of the SHREC dataset [Lia+11], as in [Han+19; Mil+20a], optimizing the standard cross entropy loss. We report

Table 2.3: Test segmentation accuracy on Shape COSEG.

| Method | Vases | Chairs | Aliens |
|---|---|---|---|
| Ours | 90.30% | 95.68% | 96.03% |
| PD-MeshNet [Mil+20a] | 95.36% | 97.23% | 98.18% |
| MeshCNN [Han+19] | 92.36% | 92.99% | 96.26% |

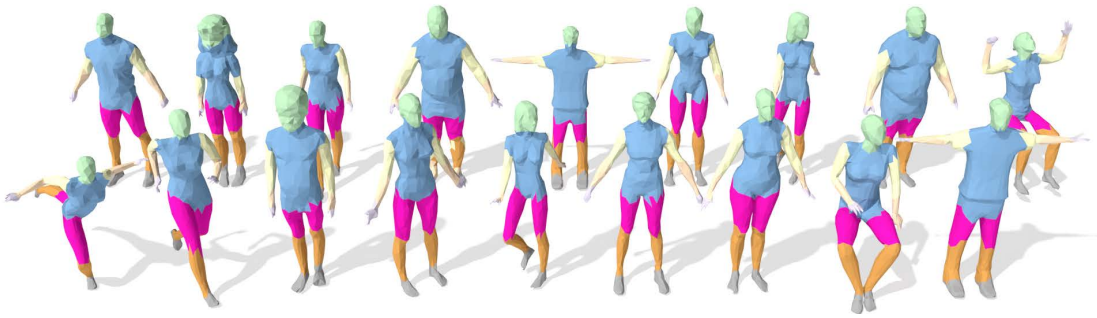


Figure 2-4: Mesh segmentation results on the Human Body test set.

our results on two different splits of the dataset—*Split 10*, where each of the 30 shape categories is randomly split into 10 test and 10 training examples, and *Split 16*, where each category is split into 4 test and 16 training examples—in Table 2.6. We train for 100 epochs using decimation to 400-500 faces, anisotropic scaling, and random rotations as data augmentation and then fine-tune for another 100 epochs for *Split 16* and 200 epochs for *Split 10*.

### 2.8.4 Dihedral angle prediction

As a stress test, we demonstrate that our method is capable of learning an operator that is sensitive to extrinsic geometry. To this end, we propose a synthetic dataset for dihedral angle regression. Previous methods that rely on computing a Laplacian would necessarily fail at this task, as they are only aware of intrinsic structure.

We take a regular mesh of a flat square consisting of 100 faces and crease it down the center at a random angle $\theta \in [0, 2\pi]$, as shown. Our network learns a two-dimensional vector per mesh, and we optimize cosine distance to the ground truth $\theta$. We use the same hyperparameters as for the other experiments with a batch size of 32. For this experiment, we only use vertex positions as the input features—we do not

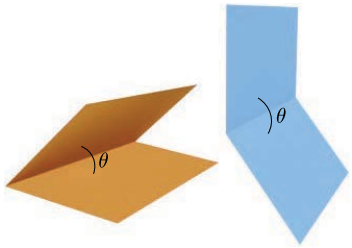


Table 2.4: Area-weighted test segmentation accuracy on Shape COSEG.

| Method | Vases | Chairs | Aliens |
|---|---|---|---|
| Ours | 94.38% | 99.22% | 97.97% |
| PD-MeshNet [Mil+20a] | 97.49% | 97.86% | 98.66% |

Figure 2-5: Test set mesh segmentations of Split 2 of the full-resolution MIT Animation Dataset [Vla+08].

provide normals. After training for just 15 minutes, we are able to predict the angle with an average error of 0.17°.

### 2.8.5 Ablation

We perform an ablation study to justify some the design and parameter choices in our architecture. In Table 2.7, we report test accuracy on the Shape COSEG vases dataset after 100 epochs of training (without fine-tuning). The accuracy degrades when we do not provide normals as part of the input mesh features, when we do not cache any additional eigenpairs for improved derivative approximation, when we reduce the vector dimensionality $k$, when we reduce the learned feature size $n$, or when we use fewer eigenpairs $m$ for feature computation.

Table 2.7: Ablation study of our parameter choices on segmentation of the Shape COSEG vases dataset.

| Model | Accuracy |
|---|---|
| full | 87.78% |
| no normals | 86.26% |
| no additional eig. | 87.44% |
| $k = 2$ | 79.02% |
| $m = 16$ | 86.34% |
| $n = 8$ | 87.08% |

## 2.9 Discussion

HodgeNet has many features that make it an attractive alternative for learning from meshes. During inference, its structure resembles that of most spectral geometry processing algorithms: con-

Table 2.6: Classification accuracy on the SHREC test set.

| Method | Split 16 | Split 10 |
|---|---|---|
| Ours | 99.17% | 94.67% |
| PD-MeshNet [Mil+20a] | 99.7% | 99.1% |
| MeshCNN [Han+19] | 98.6% | 91.0% |

struct a useful operator and compute features from its spectrum. Our model is lightweight in the sense that the learnable functions act only on local neighborhoods, yet our model has a global receptive field thanks to the eigenvector computation. It has relatively few parameters and can be evaluated efficiently on the CPU.

This exploratory work suggests many avenues for future research. The most obvious next step is to extend our model to tetrahedral meshes for volumetric problems; we do not anticipate any major issues with this extension. We also can use our method's connection to DEC to make learnable versions of other discrete differential operators, e.g. ones acting on $k$-forms for $k \geq 1$, and we can consider other applications of learning on meshes like generative modeling.

Our work also reveals some insight into other learning problems. Our architecture could easily be applied to graphs rather than triangle meshes by mildly changing the parameterization of $\star_1$ and taking $\star_0$ to be the identity matrix; we hence anticipate that there may be some applications to network analysis and other graph learning problems. Our lightweight differentiation strategy for eigenvectors may also prove useful in other contexts demanding eigenstructure of large matrices.

From the broadest perspective, our work demonstrates one of many potential applications of differentiable sparse linear algebra. While our derivative approximations and specially-formulated operator provide one way to circumvent development of a general framework for combining deep learning and linear algebra, a framework coupling sparse linear algebra to deep learning toolkits would enable a vast set of modeling choice and applications currently hamstrung by available architectures.

# 3

# Learning Parametric Shapes

We now move on to deep learning architectures that *output* interesting and useful geometry. In this chapter, we consider *parametric shapes*, which are particularly convenient for applications like CAD or modeling thanks to their intuitive and sparse representation. We show how borrowing intuition from metric geometry can motivate simple objective functions, useful in training such deep networks.

## 3.1 Introduction

The creation, modification, and rendering of parametric shapes, such as in vector graphics, is a fundamental problem of interest to engineers, artists, animators, and designers. Such representations offer distinct advantages. By expressing shapes as collections of primitives, we can easily apply transformations and render at arbitrary resolution while storing only a sparse representation. Moreover, generating parametric representations that are *consistent* across inputs enables us to learn common underlying structure and estimate correspondences between shapes, facilitating tools for retrieval, exploration, style/structure transfer, and so on.

It is often useful to generate parametric models from data that do not directly correspond to the target geometry and contain imperfections or missing parts. These artifacts be the results of noise, corruption, or human-generated input; often, an artist intends to create a precise geometric

This chapter includes material from the following publications: [Smi+20; SBS21].

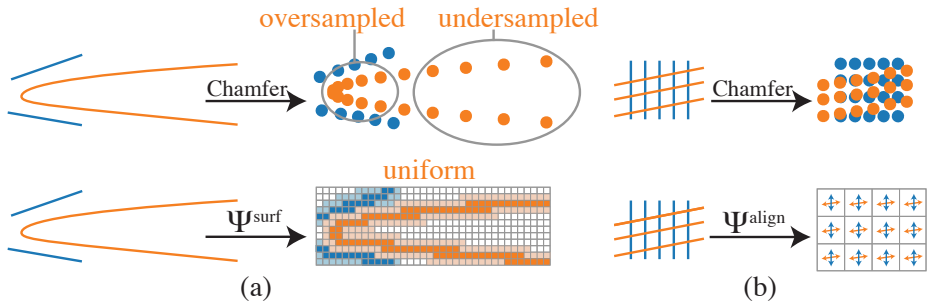


Figure 3-1: Drawbacks of Chamfer distance (above) fixed by our losses (below). In a, sampling uniformly in the parameter space of a Bèzier curve (orange) yields oversampling at the high-curvature area, resulting in a low Chamfer distance to the segments (blue). Our method yields a spatially uniform representation. In b, two sets of nearly-orthogonal line segments have near-zero Chamfer distance despite misaligned normals. We explicitly measure normal alignment.

object but produces one that is "sketchy" and ambiguous. Hence, we turn to machine learning methods, which have shown success in inferring structure from noisy data.

Convolutional neural networks (CNNs) achieve state-of-the-art results in vision tasks such as image classification [KSH12], segmentation [LSD15], and image-to-image translation [Iso+17]. CNNs, however, operate on *raster* representations. Grid structure is fundamentally built into convolution as a mechanism for information to travel between network layers. This structure is leveraged to optimize GPU performance. Recent deep learning pipelines that output vector shape primitives have been significantly less successful than pipelines for analogous tasks on raster images or voxelized volumes.

A challenge in applying deep learning to parametric geometry is the combination of Eulerian and Lagrangian representations. CNNs process data in an *Eulerian* fashion, applying fixed operations to a dense grid; Eulerian shape representations like indicator functions come as values on a fixed grid. Parametric shapes, on the other hand, use sparse sets of parameters like control points to express geometry. In contrast to stationary Eulerian grids, this *Lagrangian* representation moves with the shape. Mediating between Eulerian and Lagrangian geometry is key to any learning pipeline for the problems above, a task we consider in detail.

We propose two learning frameworks for predicting parametric shapes, addressing the aforementioned issues.

First, by analytically computing a distance field to the primitives during training, we formulate an Eulerian version of Chamfer distance, a common metric for geometric similarity [Tul+17;

FSG17b; LF04; Gro+18]. Our metric does not require samples from the predicted or target shapes, eliminating artifacts that emerge due to nonuniform sampling. Additionally, our distance field enables alternative loss functions that are sensitive to specific geometric qualities like alignment. We illustrate the advantages of our method over Chamfer distance in Fig. 3-1.

We apply our new framework in 2D to a diverse dataset of fonts, training a network that takes in a raster image of a glyph and outputs a collection of Bézier curves. This embedding effectively maps glyphs onto a common set of parameters that can be traversed intuitively. We use this embedding for font exploration and retrieval, correspondence, and interpolation in a completely self-supervised setting, without need for human labeling or annotation.

We also show that our approach works in 3D. With surface primitives in place of curves, we perform abstraction on ShapeNet [Cha+15], outputting parametric primitives to approximate each input. Our method can produce consistent shape segmentations, outperforming state-of-the-art deep cuboid fitting of Tulsiani et al. [Tul+17] on semantic segmentation.

While our distance field–based approach is effective for simple parametric primitives, it cannot be applied to more complex geometry, like parametric patches. To this end, we present a second method that operates explicitly on the geometry rather than in the parametric domain or on a sampling of surrounding space.

To test our system, we choose sketch-based modeling as a target application. Converting rough, incomplete 2D input into a clean, complete 3D shape is extremely ill-posed, requiring hallucination of missing parts and interpretation of noisy signal. To cope with these ambiguities, existing systems either rely on hand-designed priors, severely limiting applications, or learn the shapes from data, implicitly inferring relevant priors [Del+18; Wan+18c; Lun+17]. However, the output of the latter methods often lacks resolution and sharp features necessary for high-quality 3D modeling.

In industrial design, man-made shapes are typically modeled as collections of smooth parametric patches (e.g., NURBS surfaces) whose boundaries form the sharp features. To learn such shapes effectively, we use a deformable parametric template [JZD98]—a manifold surface composed of patches, each parameterized by control points (Fig. 3-6a). This representation enables the model to control the smoothness of each patch and introduce sharp edges between patches where necessary.

Compared to traditional representations, deformable parametric templates have numerous benefits for our task. They are intuitive to edit with conventional software, are resolution-independent, and can be meshed to arbitrary accuracy. Since only boundary control points are needed, our representation has relatively few parameters. Finally, this structure admits closed-form expressions for normals and other geometric features, which can be used for loss functions that improve reconstruction quality (Section 3.4.2).

Training a model for such representations faces three major challenges: detection of non-manifold surfaces, structural variation within shape categories, and lack of data. We address them as follows:

*Contributions.* We present techniques for predicting parametric shapes from 2D and 3D raster data, including:

- a *general distance field loss function* motivating several self-supervised losses based on a common formulation;
- loss functions for fitting a patch-based shape representation, inspired by machinery from metric geometry;
- application to 2D *font glyph vectorization*, with application to correspondence, exploration, retrieval, and repair;
- application to 3D *surface abstraction*, with results for different primitives and constructive solid geometry (CSG) as well as application to segmentation;
- application to sketch-based modeling of man-made shapes, producing usable CAD-style patch primitives.

## 3.2 Related work

*Deep shape reconstruction.* Reconstructing geometry from one or more viewpoints is crucial in applications like robotics and autonomous driving [FRR15; Sei+06; Su+15]. Recent deep networks can produce point clouds or voxel occupancy grids given a single image [FSG17a; Cho+16], but their output suffers from fixed resolution.

Learning signed distance fields defined on a voxel grid [DQN17; SG18] or directly [Par+19] allows high-resolution rendering but requires surface extraction; this representation is neither sparse nor modular. Liao, Donné, and Geiger [LDG18] address the rendering issue by incorporating marching cubes into a differentiable pipeline, but the lack of sparsity remains problematic, and predicted shapes are still on a voxel grid.

Parametric shapes offer a sparse, non-voxelized solution. Methods for converting point clouds to geometric primitives achieve high-quality results but require supervision, either relying on existing labeled data [NLX18; Mo+19; Gao+19] or prescribed templates [Gan+18]. Groueix et al. [Gro+18] output primitives at any resolution, but their primitives are not naturally parameterized or sparsely represented. Genova et al. [Gen+19] represent geometry as isosurfaces of axis-aligned Gaussians. Others [Sun+19; PUG19] develop tailored primitives but use standard Chamfer distance as the loss objective. We demonstrate and address the issues inherent in Chamfer distance.

*Font exploration and manipulation.* Designing or even finding a font can be tedious using generic vector graphics tools. Certain geometric features distinguish letters from one another across fonts, while others distinguish fonts from one another. Due to these difficulties and the presence of large font datasets, font exploration, design, and retrieval have emerged as challenging problems in graphics and learning.

Previous exploration methods organize fonts via crowdsourced attributes [ODo+14] or embed fonts on a manifold using purely geometric features [CK14; Bal+18]. Instead, we leverage deep vectorization to automatically generate a sparse representation for each glyph. This enables exploration on the basis of general shape rather than fine detail.

Automatic font generation methods usually fall into two categories. Rule-based methods [SI10; PFC15] use engineered decomposition and reassembly of glyphs into parts. Deep learning approaches [Aza+18; USB16] produce raster images, with limited resolution and potential for image-based artifacts, making them unfit for use as glyphs. We apply our method to edit existing fonts while retaining vector structure and show vectorization of glyphs from noisy partial data.

*Parametric shape collections.* As the number of publicly-available 3D models grows, methods for organizing, classifying, and exploring models become crucial. Many approaches decompose models into modular parametric components, commonly relying on prespecified templates or la-

beled collections of specific parts [Kim+13; She+12; Ovs+11]. Such shape collections prove useful in domain-specific applications in design and manufacturing [Sch+17; UIM12]. Our deep learning pipeline allows generation of parametric shapes to perform these tasks. It works quickly on new inputs at test time and is generic, handling a variety of modalities without supervision and producing different output types.

*Sketch-based 3D shape modeling* 3D reconstruction from sketches has a long history in graphics. A survey is beyond the scope of this paper; see [Del+18] or surveys by Ding and Liu [DL16] and Olsen et al. [Ols+09].

Unlike incremental sketch-based 3D modeling, where users progressively add new strokes [Che+05; GIZ09; Che+13; IMT99], our method interprets complete sketches, eliminating training for artists and enabling 3D reconstruction of legacy sketches.

Some systems interpret complete sketches without extra information. This input is extremely ambiguous thanks to occlusions and inaccuracies. Hence, reconstruction algorithms rely on strong 3D shape priors. These priors are typically manually created, e.g., for humanoids, animals, and natural shapes [Bes+15; Ent+15; IMT99]. Our work focuses on man-made shapes, which have characteristic sharp edges and are only piecewise smooth. Rather than relying on expert-designed priors, we *automatically* learn category-specific shape priors.

A few deep learning approaches address sketch-based modeling. Nishida et al. [Nis+16] and Huang et al. [Hua+17] train networks to predict *procedural model parameters* that yield detailed shapes from a sketch. These methods produce complex high-resolution models but only for shapes that can be procedurally generated. Lun et al. [Lun+17] use a CNN-based architecture to predict multi-view depth and normal maps, later converted to point clouds; Li et al. [Li+17] improve on their results by first predicting a flow field from an annotated sketch. In contrast, we output a deformable parametric template, which can be converted to a manifold mesh without post-processing. Wang et al. [Wan+18a] learn from unlabeled databases of sketches and 3D models with no correspondence using an adverserial training approach. Another inspiration for our research is the work of Delanoy et al. [Del+18], which reconstructs a 3D object as voxel grids; we compare to this work in Fig. 3-24.

## 3.3 Preliminaries

Let $A, B \subset \mathbb{R}^n$ be two measurable shapes. Let $X$ and $Y$ be two point sets sampled uniformly from $A$ and $B$. The *directed Chamfer distance* between $X$ and $Y$ is

$$\mathrm{Ch}_{\mathrm{dir}}(X, Y) = \frac{1}{|X|} \sum_{x \in X} \min_{y \in Y} \|x - y\|_2^2, \tag{3.1}$$

and the *symmetric Chamfer distance* is defined as

$$\mathrm{Ch}(X, Y) = \mathrm{Ch}_{\mathrm{dir}}(X, Y) + \mathrm{Ch}_{\mathrm{dir}}(Y, X). \tag{3.2}$$

These were proposed for computational applications in [Bor84] and have been used as loss functions assessing geometric similarity in learning [Tul+17; FSG17a; LF04; Gro+18].

To relate our proposed loss to Chamfer distance, we define the *variational directed Chamfer distance* as

$$\mathrm{Ch}_{\mathrm{dir}}^{\mathrm{var}}(A, B) = \frac{1}{\mathrm{Vol(A)}} \int_A \inf_{y \in B} \|x - y\|_2^2 \, \mathrm{d}V(x), \tag{3.3}$$

with *variational symmetric Chamfer distance* $\mathrm{Ch}(A, B)^{\mathrm{var}}$ defined analogously, extending (3.1) and (3.2) to smooth objects.

If points are sampled uniformly, under relatively weak assumptions, $\mathrm{Ch}(X, Y) \to 0$ if and only if $A = B$ as the number of samples grows, making it a reasonable shape matching metric. Chamfer distance, however, has fundamental drawbacks:

- It is highly dependent on the sampled points and sensitive to non-uniform sampling, as in Fig. 3-1a.
- It is agnostic to normal alignment. As in Fig. 3-1b, Chamfer distance between a dense set of vertical lines and a dense set of horizontal lines approaches zero.
- It is slow to compute. For each $x$ sampled from $A$, it is necessary to find the closest $y$ sampled from $B$, a quadratic-time operation when implemented naïvely. Efficient structures like $k$-d trees are not well-suited to GPUs.

Our method does not suffer from these disadvantages.

## 3.4 Method

We describe two approaches to designing deep network architectures that output geometry in the form of parametric shapes. The first (Section 3.4.1) relies on computing distance fields to the predicted geometry during training. The distance field serves as a convenient intermediate object, motivating several loss functions. While this approach is effective for learning simple primitives like Bézier curves or cuboids, computing distance fields becomes infeasible for more expressive geometry like parametric patches. Thus, in Section 3.4.2 we propose an alternative way for formulating objective functions, relying on a formulation using metric geometry.

### 3.4.1 Learning parametric shapes using distance fields

We introduce a framework for formulating loss functions suitable for learning parametric shapes in 2D and 3D; our formulation not only generalizes Chamfer distance but also leads to stronger loss functions that improve performance on a variety of tasks. We start by defining a general loss on distance fields and propose two specific losses.

#### General distance field loss

Given $A, B \subseteq \mathbb{R}^n$, let $\mathrm{d}_A, \mathrm{d}_B : \mathbb{R}^n \to \mathbb{R}_+$ measure distance from each point in $\mathbb{R}^n$ to $A$ and $B$, respectively, $\mathrm{d}_A(x) := \inf_{y \in A} \|x - y\|_2$. In our experiments, $n \in \{2, 3\}$. Let $S \subseteq \mathbb{R}^n$ be a bounded set with $A, B \subseteq S$. We define a *general distance field loss* as

$$\mathscr{L}_{\Psi}[A, B] = \frac{1}{\mathrm{Vol}(S)} \int_{x \in S} \Psi_{A,B}(x) \; \mathrm{d}V(x), \tag{3.4}$$

for some measure of discrepancy $\Psi$. Note that we represent $A$ and $B$ only by their respective distance functions, and the loss is computed over $S$.

Let $\Phi \in \mathbb{R}^p$ be a collection of parameters defining a shape $S_\Phi \subseteq \mathbb{R}^n$. For instance, if $S_\Phi$ consists of Bézier curves, $\Phi$ contains a list of control points. Given a target shape $T \subseteq \mathbb{R}^n$, we formulate fitting a parametric shape to approximate $T$ with respect to $\Psi$ as minimizing

$$f_\Psi(\Phi) = \mathscr{L}_\Psi[S_\Phi, T]. \tag{3.5}$$

For optimal shape parameters, $\hat{\Phi} := \arg\min_{\Phi} f_{\mathcal{Y}}(\Phi)$. We propose two discrepancy measures, providing loss functions that capture different geometric features.

### Surface loss

We define surface discrepancy to be

$$\mathcal{Y}^{\text{surf}}_{A,B}(x) = \delta\{\ker \mathrm{d}^2_A\}(x)\mathrm{d}^2_B(x) + \delta\{\ker \mathrm{d}^2_B\}(x)\mathrm{d}^2_A(x) \tag{3.6}$$

where $\delta\{X\}$ is the Dirac delta defined uniformly on $X$, and $\ker f$ denotes the zero level-set of $f$. $\mathcal{Y}^{\text{surf}} > 0$ iff the shapes do not match, making it sensitive to local geometry:

**Proposition 1** *The symmetric variational Chamfer distance between $A, B \subseteq \mathbb{R}^n$ is equal to the surface loss between $A$ and $B$, i.e.,* $\mathrm{Ch}^{\text{var}}(A, B) = \mathcal{L}_{\mathcal{Y}^{surf}_{A,B}}$.

Unlike Chamfer distance, the discrete version of our surface loss can be approximated efficiently without sampling points from either the parametric or target shape via evaluation over a regular grid, as we show in Section 3.4.1.

### Normal alignment loss

We define normal alignment discrepancy to be

$$\mathcal{Y}^{\text{align}}_{A,B}(x) = 1 - \langle \nabla \mathrm{d}_A(x), \nabla \mathrm{d}_B(x) \rangle^2. \tag{3.7}$$

Minimizing $f_{\mathcal{Y}^{\text{align}}}$ aligns the predicted primitives' normals to those of the target. Following Fig. 3-1b, if $A$ contains vertical lines, and $B$ contains horizontal lines, $\mathcal{L}_{\mathcal{Y}^{\text{align}}_{A,B}} \gg 0$ while $\mathrm{Ch}(A, B) \approx 0$.

### Final loss function

The general distance field loss and proposed discrepancy measures are differentiable with respect to the shape parameters $\Phi$, as long as $\mathrm{d}_{S_\Phi}$ is differentiable with respect to $\Phi$. Thus, they are well-suited to be optimized by a deep network predicting parametric shapes. We discretize (3.4):

$$\mathcal{L}_{\mathcal{Y}}[A, B] \approx \frac{1}{|G|} \sum_{x \in G} \mathcal{Y}_{A,B}(x), \tag{3.8}$$

where $G$ is a 2D or 3D grid.

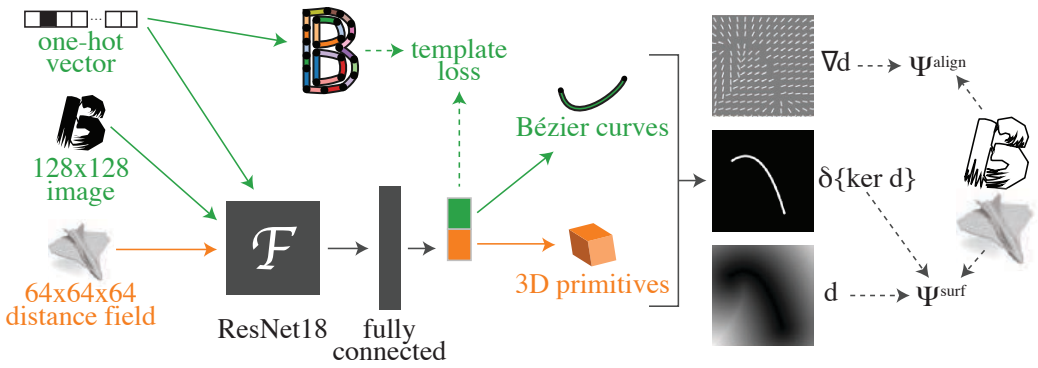



Figure 3-2: An overview of our pipelines—font vectorization and 3D abstraction.

While we use a voxel grid to compute the integrals in our loss function, the resolution of the voxel grid only affects quadrature without limiting the resolution of our representation. The grid dictates how we sample distance values; the values themselves are derived from a continuous parametric representation. A small subvoxel change in the geometry will affect the distance value at multiple discrete voxels. This property is in distinct contrast to representations that only consider the occupancy grid of a shape—the resolution of such representations is strictly limited by the grid resolution.

For $\Psi^{\text{surf}}$, we use Smootherstep$(1 - \mathrm{d}_A^2/\gamma^2)$ (with Smootherstep defined as in [EM03]) as a smooth version of $\delta\{\ker \mathrm{d}_A^2\}$ to evaluate the expression on a grid and to avoid discontinuities, enabling smooth gradients in our optimization. We set $\gamma$ to twice the diameter of a voxel. For $\Psi^{\text{align}}$, we approximate gradients using finite differences.

We minimize $f_\Psi = f_{\Psi^{\text{surf}}} + \alpha^{\text{align}} f_{\Psi^{\text{align}}}$, determining $\alpha^{\text{align}} = 0.01$ for all experiments using cross-validation.

### Network architecture and training

The network takes a 128×128 image or a 64×64×64 distance field as input and outputs a parametric shape. We encode our input to a $\mathbb{R}^{256}$ latent space using a ResNet-18 [He+16] architecture. We then use a fully connected layer with 256 units and ReLU nonlinearity followed by a fully connected layer with number of units equal to the dimension of the target parameterization. We pass the output through a sigmoid and rescale it depending on the parameters being predicted. Our pipeline is illustrated in Fig. 3-2. We train each network on a single Tesla GeForce GTX Titan X

GPU for approximately one day, using Adam [KB14] with learning rate $10^{-4}$ and batch size 32 for 2D and 16 for 3D.

### 2D: Bèzier curve networks

We first describe our choice of primitives as well as the computation of their distance fields. We introduce a template-based approach to allow our network to better handle multimodal data (different letters) and test several applications.

*Primitives.* We wish to use a 2D parametric shape primitive that is sparse and expressive and admits an analytic distance field. Our choice is the *quadratic Bèzier curve* (which we refer to as *curve*), parameterized by control points $a, b, c \in \mathbb{R}^2$ and defined by $\gamma(t) = (1-t)^2 a + 2(1-t)tb + t^2 c$, for $0 \leq t \leq 1$. We represent 2D shapes as the union of $n$ curves parameterized by $\Phi = \{a_i, b_i, c_i\}_{i=1}^n \subseteq \mathbb{R}^{3n}$.

**Proposition 2** *Given a curve $\gamma$ parameterized by $a, b, c \in \mathbb{R}^2$ and a point $p \in \mathbb{R}^2$, the $\hat{t} \in \mathbb{R}$ such that $\gamma(\hat{t})$ is the closest point on the curve to $p$ satisfies the following:*

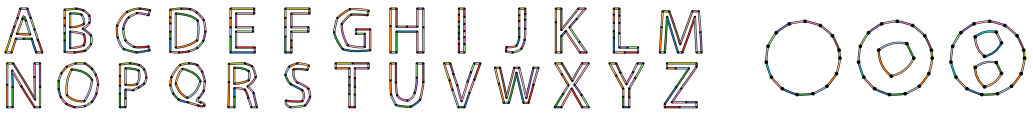
$$\langle B, B\rangle \hat{t}^3 + 3\langle A, B\rangle \hat{t}^2 + (2\langle A, A\rangle + \langle B, a-p\rangle)\hat{t} + \langle A, a-p\rangle = 0, \tag{3.9}$$

*where $A = b - a$ and $B = c - 2b + a$.*

Thus, evaluating the distance to a single curve $\mathrm{d}_{\gamma_i}(p) = \|p - \gamma_i(\hat{t})\|_2$ requires finding the roots of a cubic [QMK06], which we can do analytically. To compute distance to the union of the curves, we take a minimum: $\mathrm{d}_\Phi(p) = \min_{i=1}^n \mathrm{d}_{\gamma_i}(p)$.

In addition to the control points, we predict a stroke thickness for each curve. We use this parameter when computing the loss by "lifting" the predicted distance field, thus thickening the curve—if curve $\gamma$ has thickness $s$, we set $d_\gamma^s(p) = \min(d_\gamma(p) - s, 0)$. While we do not visualize stroke thickness in our experiments, this approach allows the network to thicken curves to better match high-frequency filigree (see Fig. 3-3). This thickening is a simple operation in our distance field representation; sampling-based methods do not provide a natural way to thicken predicted geometry.

Figure 3-3: Glyphs with predicted boundary curves rendered with predicted stroke thickness. The network thickens curves to account for stylistic details at the glyph boundaries.



(a) Lettter templates (a) Simple templates

Figure 3-4: Font glyph templates. These determine the connectivity and initialize the placement of the predicted curves.

*Templates.* Our training procedure is self-supervised, as we do not have ground truth curve annotations. To better handle the multimodal nature of our entire dataset with a single network, we label each training example with its letter, passed as additional input. This allows us to condition on input class by concatenating a 26-dimensional one-hot vector to the input, a common technique for conditioning [Zhu+17].

We choose a "standard" curve representation per letter, capturing each letter's distinct geometric and topological features, by designing 26 templates from a shared set of control points. A *template* of type $\ell \in \{\mathrm{A}, \ldots, \mathrm{Z}\}$ is a collection of points $T_\ell = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^{2n}$ with corresponding *connectivity* determining how the points define curves. Since our curves form closed loops, we reuse endpoints.

For glyph boundaries of uppercase English letters, there are three connectivity types—one loop (e.g., "C"), two loops (e.g., "A"), and three loops ("B"). In our templates, the first loop has 15 curves and the other loops have 4 curves each. We will show that while letter templates (Fig. 3-4a) better specialize to the boundaries of each glyph, we still achieve good results with simple templates (Fig. 3-4b). Even without letter-specific templates, our system learns a consistent geometric representation, establishing cross-glyph correspondences purely using self-supervision.

We use predefined templates together with our labeling of each training example for two purposes. First, connectivity is used to compute curve control points from the network output. Sec-

ond, they provide a *template loss*:

$$\mathscr{L}^{\text{template}}(\ell, x) = \alpha^{\text{template}} e^{(t/s)} \|T_\ell - h^t(x)\|_2^2, \tag{3.10}$$

where $s \in \mathbb{Z}_+, \gamma \in (0, 1)$, $t$ is the iteration number, $x$ is the input image, and $h^t(x)$ is the network output at iteration $t$. This initializes the network output, such that an input of type $\ell$ initially maps to template $\ell$. As this term decays, the other loss terms take over. We set $\alpha^{\text{template}} = 10$ and $s = 500$, though other choices of parameters for which the template term initially overpowers the rest of the loss also work.

### 3D: Cuboids and rounded cuboids

We reconstruct 3D surfaces out of various primitives, which allow our model to be expressive, sparse, and abstract.

Our first primitive is a *cuboid*, parameterized by $\{b, t, q\}$, where $b = (w, h, d), t \in \mathbb{R}^3$ and $q \in \mathbb{S}^4$ a quaternion, i.e., an origin-centered (hollow) rectangular prism with dimensions $2b$ to which we apply rotation $q$ and then translation $t$.

**Proposition 3** *Let $C$ be a cuboid with parameters $\{b, t, q\}$ and $p \in \mathbb{R}^3$ a point. Then, the signed distance between $p$ and $C$ is*

$$\mathrm{d}_C(p) = \|\max(d, 0)\|_2 + \min(\max(d_x, d_y, d_z), 0), \tag{3.11}$$

*where $p' = q^{-1}(p - t)q$ using the Hamilton product and $d = (|p'_x|, |p'_y|, |p'_z|) - b$.*

Inspired by [PUG19], we additionally use a *rounded cuboid* primitive by introducing a radius parameter $r$ and computing the signed distance by $\mathrm{d}_{RC}(p) = \mathrm{d}_C(p) - r$.

A unique advantage of our distance field representation is the ability to perform CSG boolean operations. Since our distances are *signed*, we can compute the distance to the union of $n$ primitives by taking a minimum over distance fields. With sampling-based methods such as Chamfer distance optimization, care must be taken to avoid sampling interior faces that are not part of the outer surface.

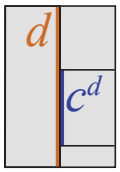
### 3.4.2 Learning parametric patches

For learning primitives whose distance field cannot be easily computed, we propose an alternative appraoch. Our learning pipeline outputs a parametrically-defined 3D surface. We describe our geometric representation (Section 3.4.2), define our loss (Section 3.4.2), and specify our architecture and training procedure (Section 3.4.2).

#### Representation

*Patches.* Our surfaces are collections of *Coons patches* [Coo67], a commonly used and rich subset of NURBS surfaces. A patch is specified by four boundary cubic Bézier curves sharing endpoints (Fig. 3-6a). Each curve has control points $p_1, \ldots, p_4 \in \mathbb{R}^3$. A Bézier curve $c : [0,1] \to \mathbb{R}^3$ is $c(\gamma) = p_1(1-\gamma)^3 + 3p_2\gamma(1-\gamma)^2 + 3p_3\gamma^2(1-\gamma) + p_4\gamma^3$, and a Coons patch $P : [0,1]^2 \to \mathbb{R}^3$ is

$$\begin{aligned} P(s,t) =& (1-t)c_1(s) + tc_3(1-s) + sc_2(t) + (1-s)c_4(1-t) \\ &- \big(c_1(0)(1-s)(1-t) + c_1(1)s(1-t) + c_3(1)(1-s)t + c_3(0)\big)\ st. \end{aligned} \tag{3.12}$$




*Templates.* Templates specify patch connectivity. A template consists of the minimal number of control points necessary to define the patches; shared control points are reused (Figs. 3-6b and 3-6c). We allow the edge of one patch to be contained *within* the edge of another using *junction curves*. A *junction curve* $c^d$ is constrained to a lie along a parent curve $d$ and is thus parameterized by $s, t \in [0,1]$, such that $c(0) = d(s)$ and $c(1) = d(t)$.

A template provides hard topological constraints for our surfaces, an initialization of their geometry, and, optionally, a means for regularization. Templates are crucial in ensuring that the patches have consistent topology—an approach without templates would result in unstructured, non-manifold patch collections. While our method works using a generic sphere template, we can define templates per shape category to incorporate category-specific priors. These templates capture only coarse geometric features and approximate scale. We outline an algorithm for obtaining templates below.

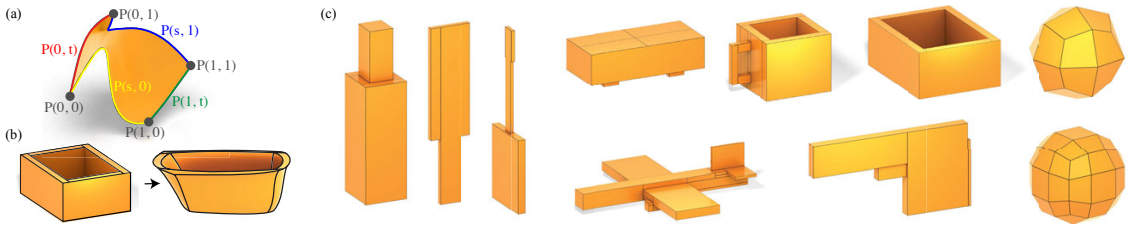



Figure 3-6: Our representation is composed of Coons patches (a) organized into a deformable template (b). We use the following templates (c, top to bottom, left to right): bottle, knife, guitar, car, airplane, coffee mug, gun, bathtub, 24-patch sphere, 54-patch sphere.

*Algorithmic construction of templates.* We design a simple system to construct a template given as input any collection of cuboids. Such a collection can be computed automatically for a shape category, e.g., given a segmentation or using self-supervised methods such as [Smi+20; Tul+17; Sun+19], or easily produced using standard CAD software. We show templates algorithmically computed from pre-segmented shapes—we obtain a collection of cuboids by taking the bounding box around each connected component of each segmentation class.

A generic cuboid decomposition cannot be used as a template, since individual cuboids may overlap. We snap cuboids to an integer lattice, split each face at grid coordinates, and remove overlapping and interior faces to obtain a manifold quad mesh. This mesh typically consists of many faces, and so, we simplify it. We merge adjacent quads with a greedy agglomerative algorithm, iterating over each quad in order of descending area and merging with an adjacent quad as long as the merge does not result in ill-defined junction curves. We show an example of this process in Fig. 3-5. Given decompositions of multiple shapes in a category, we use the median model with respect to Chamfer distance. Since models within a category are aligned, the median provides a rough approximation of the typical geometry.

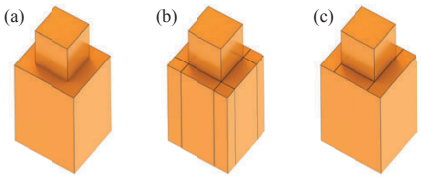



Figure 3-5: Summary of our template algorithm. Given a collection of cuboids (a), we form a quad mesh (b), and merge faces to get a template (c).

*Structural variation using templates.* For category-specific templates, we use the fact that template patches are consistently placed

on semantically meaningful parts to account for structural variation. For instance, certain airplane models contain turbines, while others do not. We note which template patches come from

cuboids corresponding to turbines, and, during training, only use turbine patches for models that contain turbines. This allows training on the entire airplane shape category, effectively using two distinct templates. At test time, the user can toggle turbines on or off for any output.

### Loss

We fit a collection of patches $\{P_i\}$ to a target mesh $M$ by optimizing a differentiable loss function. Below, we describe each term—a reconstruction loss generalizing Chamfer distance, a normal alignment loss, a self-intersection regularizer, a patch flatness regularizer, and two template priors.

*Area-weighted Chamfer distance.* While it is difficult to both to sample uniformly from patches as well as to compute distance fields, we can sample easily from their parametric domain (the unit square). We again start with variational Chamfer distance, as defined in (3.3), and perform a change of variables to get the following:

$$\begin{aligned}
\mathrm{Ch}^{\mathrm{var}}_{\mathrm{dir}}(P, M) &= \frac{1}{\mathrm{Area}(P)} \int_P \inf_{y \in M} \mathrm{d}(x, y)\, \mathrm{d}x && (3.13)\\
&= \frac{1}{\mathrm{Area}(P)} \int_0^1 \int_0^1 \inf_{y \in M} \mathrm{d}\left(P(s,t), y\right) |J(s,t)|\, \mathrm{d}s\, \mathrm{d}t && (3.14)\\
&= \frac{1}{\mathrm{Area}(P)} \cdot \frac{1}{\mathrm{Area}(\square)} \int_0^1 \int_0^1 \inf_{y \in M} \mathrm{d}\left(P(s,t), y\right) |J(s,t)|\, \mathrm{d}s\, \mathrm{d}t && (3.15)\\
&= \frac{1}{\mathrm{Area}(P)}\, \mathbb{E}_{(s,t)\sim \mathcal{U}_\square} \left[ \inf_{y \in M} \mathrm{d}(P(s,t), y) |J(s,t)| \right] && (3.16)\\
&= \frac{\mathbb{E}_{(s,t)\sim \mathcal{U}_\square} \left[ \inf_{y \in M} \mathrm{d}(P(s,t), y) |J(s,t)| \right]}{\mathbb{E}_{(s,t)\sim \mathcal{U}_\square} \left[ |J(s,t)| \right]}, && (3.17)
\end{aligned}$$

where $J(s,t)$ is the Jacobian of $P$. We approximate this expression via Monte Carlo integration.

Since we can precompute uniformly sampled random points from the target mesh, we do not need to use area weights for $\mathrm{Ch}^{\mathrm{var}}_{\mathrm{dir}}(M, P)$. Thus, our area-weighted Chamfer distance is

$$\mathscr{L}_{\mathrm{Ch}}(\cup P_i, M) = \frac{\sum_i \sum_{(s,t)\in U_\square} \min_{y \in M} \mathrm{d}(P(s,t), y) |J_i(s,t)|}{\sum_i \sum_{(s,t)\in U_\square} |J_i(s,t)|} + \frac{\sum_{x \in M} \min_{y \in \cup P_i} \mathrm{d}(x, y)}{|M|}, \quad (3.18)$$

where $U_\square \sim \mathcal{U}_{[0,1]^2}$. We compute $J_i(u,v)$ for a patch given its control points in closed-form.

*Normal alignment.* The normal alignment term is computed analogously to $\mathrm{Ch}_{\mathrm{dir}}$, except that instead of Euclidean distance, we use $\mathrm{d}_N(x, y) = \|n_x - n_y\|_2^2$, where $n_x$ is the unit normal at $x$. For each point $y$ sampled from our predicted surface, we compare $n_y$ to $n_x$, where $x \in M$ is closest to $y$, and, symmetrically, for each $x' \in M$, we compare $n_{x'}$ to to $n_{y'}$, where $y' \in \cup P_i$ is closest to $x'$:

$$\begin{aligned}\mathscr{L}_{\mathrm{normal}}(\cup P_i, M) = &\frac{\sum_i \sum_{(u,v)\in U_\square} \mathrm{d}_N\left(\mathrm{NN}_M(P_i(u,v)), P_i(u,v)\right) |J_i(u,v)|}{\sum_i \sum_{(u,v)\in U_\square} |J_i(u,v)|} \\ &+ \frac{\sum_{x\in M} \mathrm{d}_N\left(x, \mathrm{NN}_{\cup P_i}(x)\right)}{|M|},\end{aligned} \tag{3.19}$$

where $\mathrm{NN}_Y(x)$ is the nearest neighbor to $x$ in $Y$ under Euclidean distance.

*Intersection regularization.* We introduce a *collision detection loss* to detect patch intersections:

$$\mathscr{L}_{\mathrm{coll}}(\{P_i\}) = \sum_{i\neq j} \exp\left(-\left(\min(\mathrm{d}(\mathscr{T}^i, P^j), \mathrm{d}(\mathscr{T}^j, P^i))/\varepsilon\right)^2\right), \tag{3.20}$$

where $\mathscr{T}^i$ is a triangulation of patch $P_i$, and $P^i$ is a set of points sampled from patch $P_i$. To triangulate a patch, we take a fixed triangulation of the parameter space (a unit square) and compute the image of each vertex under the Coons patch map, keeping the original connectivity. With a small $\varepsilon = 10^{-6}$, this expression is a smooth indicator for when two patches are intersecting. For a pair of adjacent patches or those that share a junction, we truncate one patch at the adjacency before evaluating the loss.

*Patch flatness regularization.* To help patches to align to smooth regions and sharp creases to fall on patch boundaries, we define a *patch flatness regularizer*, which discourages excessive curvature by regularizing each Coons patch map $P : [0,1] \times [0,1] \to \mathbb{R}^3$ to be close to a linear map. For each patch, we sample points $U_\square$ in parameter space, compute their image $P(U_\square)$, and fit a linear function using least-squares. Thus, $\hat{P}(U_\square) = AU_\square + b \approx P(U_\square)$ for some $A, b$. We define the patch flatness loss as

$$\mathscr{L}_{\mathrm{flat}}(\{P_i\}) = \frac{\sum_i \sum_{(u,v)\in U_\square} \|\hat{P}_i(u,v) - P_i(u,v)\|_2^2 |J_i(u,v)|}{\sum_i \sum_{(u,v)\in U_\square} |J_i(u,v)|}. \tag{3.21}$$

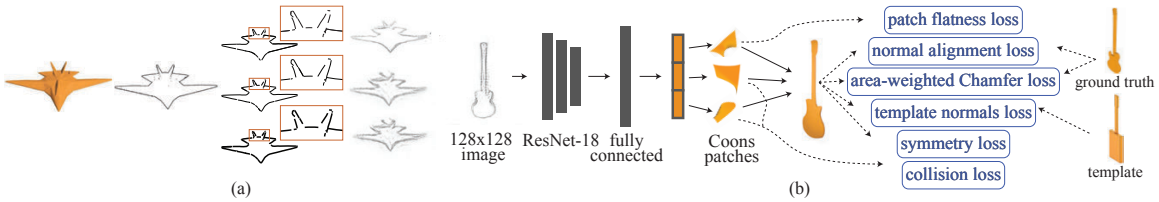



Figure 3-7: An overview of our data generation and augmentation (a) and learning (b) pipelines.

*Template normals regularization.* For categories where a template is available, we not only initialize the network with the template geometry but also regularize the output using template normals. This favorably positions patch seams and prevents patches from sliding over high-curvature regions:

$$\mathscr{L}_{\text{template}}(\{P_i\},\{T_i\}) = \frac{\sum_i \sum_{(u,v)\in U_\square} \|n_{P_i(u,v)} - n_{T_i}\|_2^2 |J_i(u,v)|}{\sum_i \sum_{(u,v)\in U_\square} |J_i(u,v)|}, \tag{3.22}$$

where $n_{T_i}$ is the normal vector of the $i$th template patch.

*Global symmetry.* Man-made shapes often exhibit global bilateral symmetries. Enforcing symmetry is difficult in, e.g., meshes or implicit surfaces. In contrast, after computing a template's symmetry planes, we enforce symmetric positions of the corresponding control points as an additional loss term:

$$\mathscr{L}_{\text{sym}}(\cup P_i) = \frac{1}{|S|} \sum_{(i,j)\in S} \|(P_x^i - a, P_y^i, P_z^i) - (a - P_x^j, P_y^j, P_z^j)\|_2^2, \tag{3.23}$$

where $S$ contains index pairs of symmetric control points $(P^i, P^j)$. In the formula, we assume, without loss of generality, symmetry plane $x = a$. We use this to enforce symmetrical reconstruction of airplanes and cars.

### Deep learning pipeline

The final loss that we optimize is

$$\begin{aligned}\mathscr{L}(\{P_i\}, M) = {} & \mathscr{L}_{\text{Ch}}(\cup P_i\, M) + \alpha_{\text{normal}}\mathscr{L}_{\text{normal}}(\cup P_i\, M) \\ & + \alpha_{\text{flat}}\mathscr{L}_{\text{flat}}(\{P_i\}) + \alpha_{\text{coll}}\mathscr{L}_{\text{coll}}(\{P_i\}) + \alpha_{\text{template}}\mathscr{L}_{\text{template}}(\{P_i\},\{T_i\}) + \alpha_{\text{sym}}\mathscr{L}_{\text{sym}}(\cup P_i).\end{aligned} \tag{3.24}$$

For models scaled to fit in a unit sphere, we use $\alpha_{\text{normal}} = 0.008$, $\alpha_{\text{flat}} = 2$, and $\alpha_{\text{coll}} = 0.00001$ for all experiments, and $\alpha_{\text{template}} = 0.0001$ and $\alpha_{\text{sym}} = 1$ for experiments that use those regularizers.

Our network inputs one or more $128 \times 128$ images and outputs patch parameters. The architecture is ResNet-18 [He+16] followed by hidden layers with 1024, 512, and 256 units, and an output layer with size equal to the output dimension. Final layer weights are initialized to zero with bias equal to the template parameters. For multi-view input, we encode each image and do max pooling over the latent codes. We use ReLU and batch normalization after each layer except the last. We train each network for 24 hours on a Tesla V100 GPU, using Adam [KB14] and batch size 8 with learning rate 0.0001. At each iteration, we sample 7,000 points from the predicted and target shapes. Our pipeline is illustrated in Fig. 3-7b.

## 3.5 Experiments

We demonstrate results of learning parametric shape representations using deep neural networks. In Sections 3.5.1 and 3.5.2, we use our distance field–based loss functions to output Bézier curves and simple 3D primitives, respectively. Then, in Section 3.5.3 we use our metric-based approach to learn expressive patch-based representations.

### 3.5.1 2D: Font exploration and manipulation

We demonstrate our method in 2D for font glyph vectorization. Given a raster image of a glyph, our network outputs control points defining a collection of quadratic Bézier curves that approximate its outline. We produce nearly exact vector representations of glyphs from simple (non-decorative) fonts. From a decorative glyph with fine-grained detail, however, we recover a good approximation of the glyph's shape using a small number of Bézier primitives and a consistent structure. This process can be interpreted as projection onto a common latent space of control points.

We train our network on the 26 uppercase English letters extracted from nearly 10,000 fonts. The input is a raster image of a letter, and the target distance field to the boundary of the original vector representation is precomputed.

*Ablation study.* We demonstrate the benefit of our loss over Chamfer distance as well as the contribution of each of our loss terms. While having 26 unique templates helps achieve better results, it is not crucial—we evaluate a network trained with three "simple templates" (Fig. 3-4b), which capture the three topology classes of our data.

Table 3.1: Comparison between subsets of our full model as well as standard Chamfer distance and AtlasNet. Average error is Chamfer distance (in pixels on a 128×128 image) between ground truth and uniformly sampled predicted curves.

| Model | Average error |
|---|---|
| **Full model (ours)** | **0.509** |
| No surface term (ours) | 1.613 |
| No alignment term (ours) | 0.642 |
| Simple templates (ours) | 0.641 |
| Chamfer (with letter templates) | 0.623 |
| AtlasNet [Gro+18] | 5.154 |

For the Chamfer loss experiment, we use the same hyperparameters as for our method and sample 5,000 points from the source and target geometry. We initialize the model output to the full letter templates, like in our full model.

We also evaluate on 20 sans-serif fonts, computing Chamfer distance between our predicted curves and ground truth geometry, sampling uniformly (average error in Table 3.1). Uniform sampling is a computationally-expensive and non-differentiable procedure only for evaluation *a posteriori*—not suitable for training. While it does not correct all of Chamfer distance's shortcomings, we use it as a baseline to evaluate quality. We limit to sans-serif fonts since we do not expect to faithfully recover local geometry. Our full loss outperforms Chamfer loss, and both our loss terms are necessary. Figure 3-8 shows test set qualitative results.

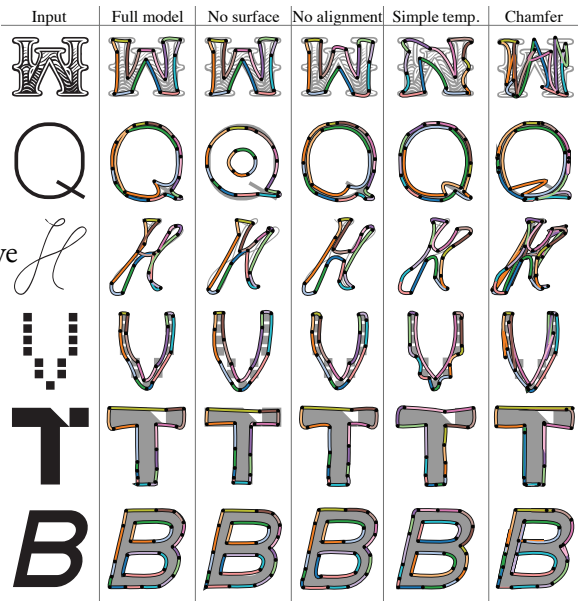


Figure 3-8: Ablation study.

We demonstrate robustness in Fig. 3-9 by quantizing our loss values and plotting the number of examples for each value. High loss outliers are generally caused by noisy data—they are either not uppercase English letters or have fundamentally uncommon structure, e.g., a *B* glyph comprised of just a single loop.

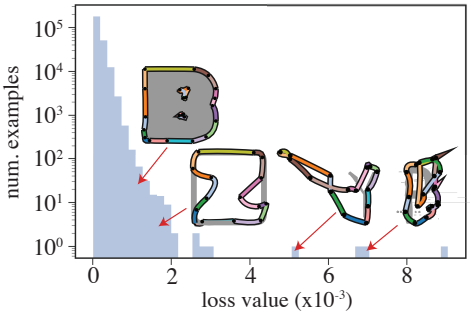
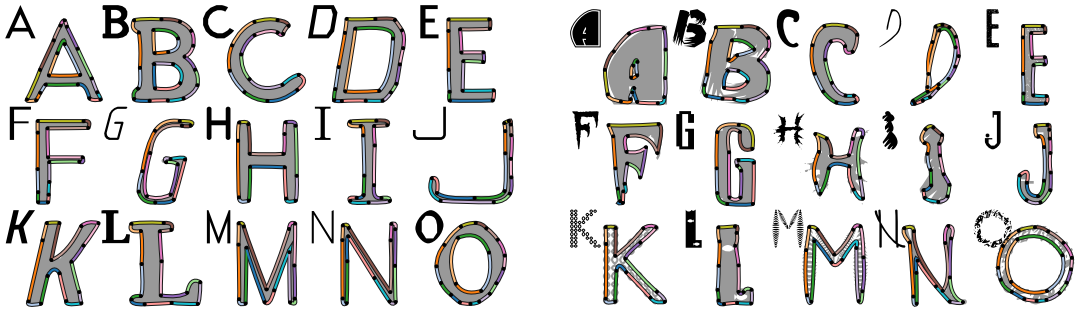



Figure 3-9: Number of examples per quantized loss value.



(a) Plain font glyphs

(b) Decorative font glyphs

Figure 3-10: Vectorization of various glyphs. For each we show the raster input (top left,black) along with the vectorization (colored curves) superimposed. When the input has simple structure (a), we recover an accurate vectorization. For fonts with decorative details (b), our method places curves to capture overall structure. Results are taken from the test dataset.

*Comparison to AtlasNet.* In AtlasNet [Gro+18], geometry is reconstructed by training implicit decoders, which map a point in the unit square to a point on the target surface, optimizing Chamfer distance. We modify the AtlasNet system to our task and demonstrate that our method method proposes a more effective geometry representation and loss.

AtlasNet represents shapes as points in a learned high dimensional space, which does not obviously correlate to geometric features. Thus, in contrast to our explicit representation as a collection of control points, it does not facilitate geometric interpretability. Additionally, their representations makes it difficult to impose geometric priors—it is unclear how to initialize AtlasNet to predefined templates, as we do in Section 3.4.1.

For a fair comparison, we train an AtlasNet model that maps points from the boundary of a circle (rather than the interior of a square) into 2D. We only train on letters with single loop topology (*C, E, F,* etc.) and sample 5,000 points. Thus, this setting is comparable to the *simple templates* experiment from our ablation.

We show results in Fig. 3-11. Although AtlasNet recovers overall structure of the input, it suffers from self-intersections and imprecision not exhibited by our method, even with simple templates. Likely, these artifacts are due to the fact that AtlasNet exhibits the drawbacks of Chamfer distance identified in Section 3.3, i.e., non-uniform sampling and lack of sensitivity to normal alignment. We include a quantitative comparison in Table 3.1. Our method outperforms AtlasNet even based on on a uniformly-sampled Chamfer distance metric.

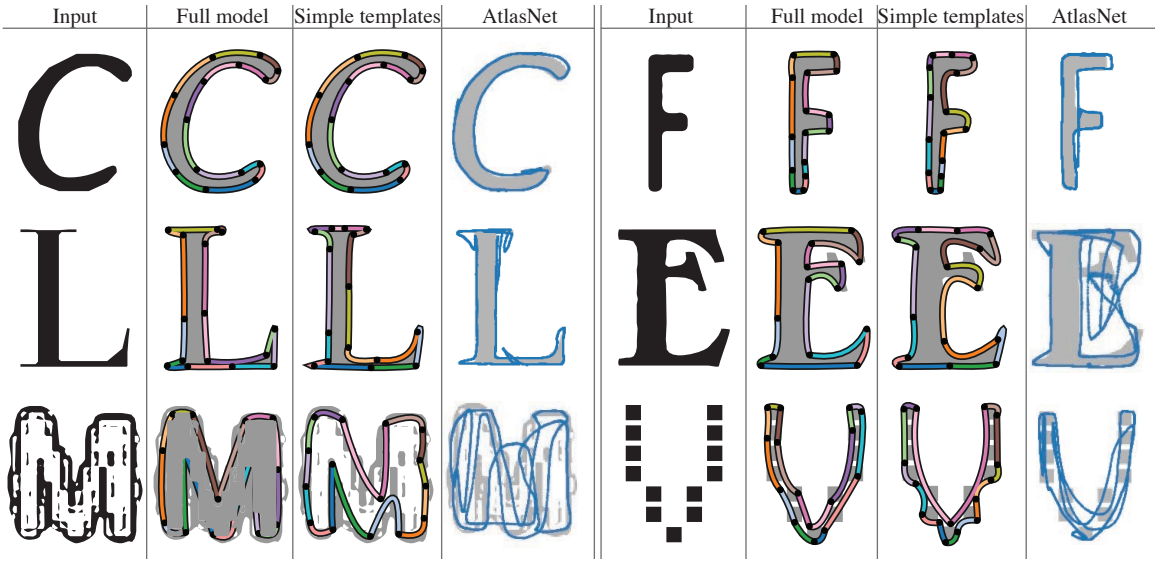



Figure 3-11: Comparison to AtlasNet [Gro+18] with a closed loop start shape to our simple templates and full models. We only train (and test) AtlasNet on letters with a single loop.

*Vectorization.* For any font glyph, our method generates a consistent sparse vector representation, robustly and accurately describing the glyph's structure while ignoring decorative and noisy details. For simple fonts, our representation is a near-perfect vectorization, as in Fig. 3-10a. For decorative glyphs, our method produces a meaningful abstraction. While a true vectorization would contain many curves with a large number of connected components, we succinctly capture the glyph's overall structure (Fig. 3-10b).

Our method preserves semantic correspondences. The same curve is consistently used for the boundary of, e.g., the top of an "I". These correspondences persist *across* letters with both full and simple templates—see, e.g., the "E" and "F" in Figs. 3-10a and 3-10b, and "simple templates" in Fig. 3-8.

*Retrieval and exploration.* Our sparse representation can be used to explore the space of glyphs, useful for artists and designers, without the need for manual labelling or annotation. Treating control points as a metric space, we can perform Euclidean nearest-neighbor lookups for font retrieval.

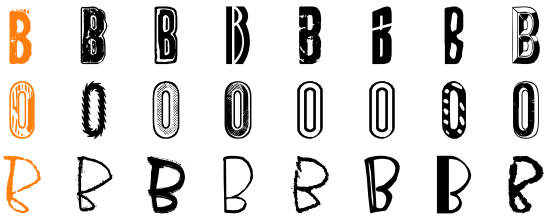


Figure 3-12: Nearest neighbors for a glyph in curve space, sorted by proximity. The query glyph is in orange.

In Fig. 3-12, for each query glyph, we compute its curve representation and retrieve seven nearest neighbors in curve space. Because our representation captures geometric structure, we find glyphs that are similar structurally, despite decorative and stylistic differences.

We can also consider a path in curve space starting at the curves for one glyph and ending at those for another. By sampling nearest neighbors along this trajectory, we "interpolate" between glyphs. As in Fig. 3-14, this produces meaningful collections of fonts for the same letter and reasonable results when the start and end glyphs are different letters.

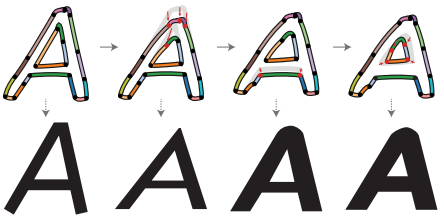


Figure 3-13: User-guided font exploration. At each edit, the nearest glyph is displayed below.

Nearest-neighbor lookups in curve space also can help find a font matching desired geometric characteristics. A possible workflow is in Fig. 3-13—through incremental refinements of the curves the user can quickly find a font.

*Style and structure mixing.* Our sparse curve representation describes geometric structure, ignoring stylistic and decorative details. We leverage this property to warp a glyph with desired style to the structure of another glyph (Fig. 3-15).

We first generate the sparse curve representation for source and target glyphs. Since our representation uses the same set of curves, we can estimate dense correspondences and use them to warp original vectors of the


Figure 3-15: Mixing of style (columns) and structure (rows) of the *A* glyph.

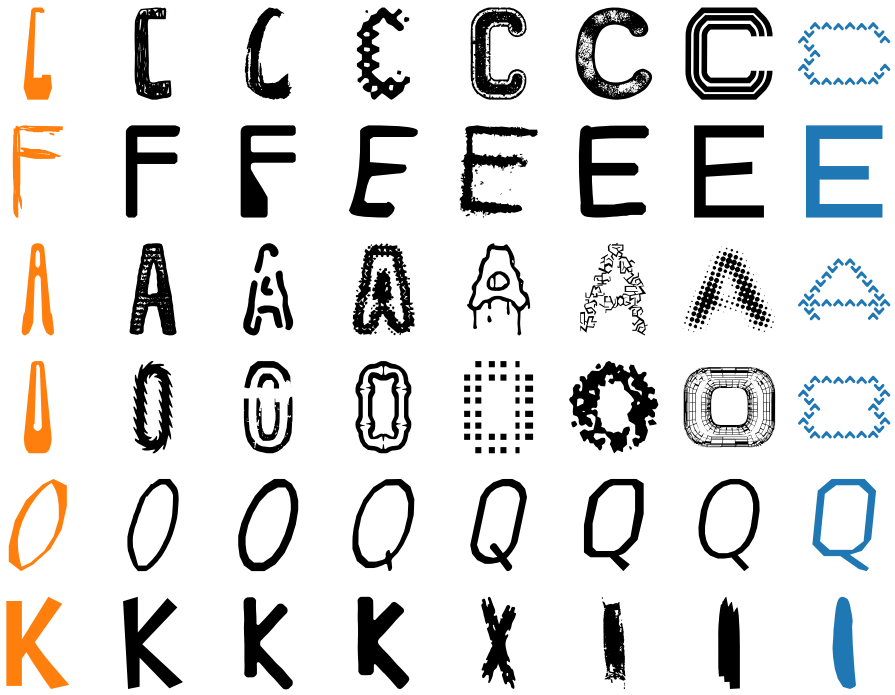


Figure 3-14: Interpolating between fonts in curve space. The start and end are in orange and blue, respectively, and the nearest glyphs to linear interpolants are shown in order.

source glyph to conform to the shape of the target. For each point on the source, we apply a translation that is a weighted sum of the translations from the sparse curve control points in the source glyph to those in the target glyph.

*Repair.* Our system learns a strong prior on glyph shape, allowing us to robustly handle noisy input. In [Aza+18], a generative adversarial network (GAN) generates novel glyphs. The outputs, however, are raster images, often with noise and missing parts. Figure 3-16 shows how our method can simultaneously vectorize and repair GAN-generated glyphs. Compared to a vectorization tool like Adobe Illustrator Live Trace, we infer missing data based on learned priors, making the glyphs usable starting points for font design.

*Other glyphs.* Our method generalizes to more complex input than uppercase English glyphs. We demonstrate this by training a model to vectorize a Chinese character, which has significant geometric and topological complexity. We use a template that roughly captures the structure of the character. Results on several fonts are shown in Fig. 3-17.

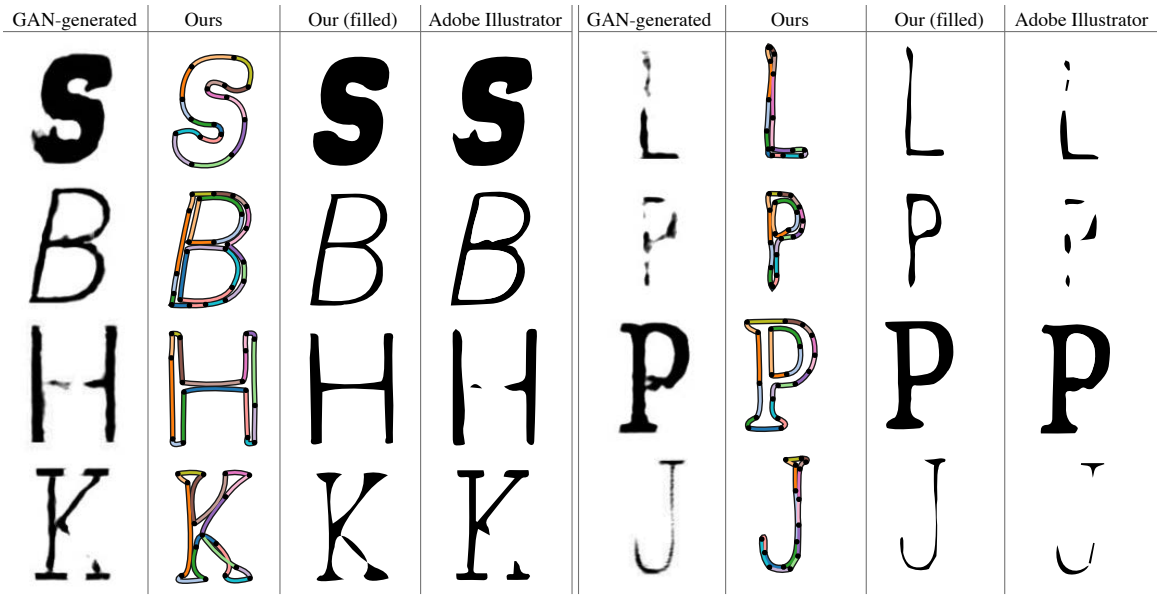



Figure 3-16: Vectorized GAN-generated fonts from [Aza+18].

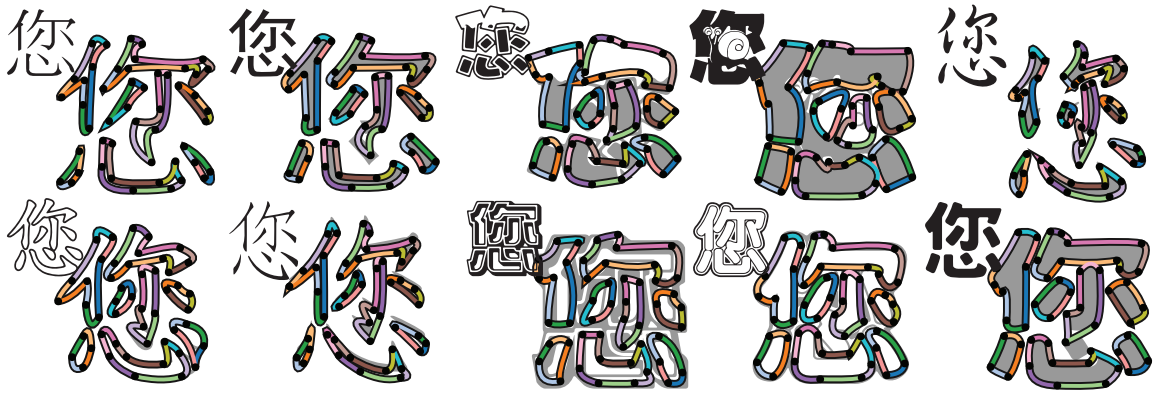


Figure 3-17: Vectorization of a Chinese character.

### 3.5.2 3D: Volumetric primitive abstraction

We train on the airplane and chair categories of ShapeNet Core V2 [Cha+15], taking as input a distance field. Thus, our method is fully self-supervised.

*Surface abstraction.* In Fig. 3-19, for each ShapeNet chair, we show the our cuboid abstraction, our rounded cuboid abstraction, and the abstraction of [Tul+17]. We show our cuboid abstractions of ShapeNet airplanes in Fig. 3-18. Each of our networks outputs 16 prim-

Figure 3-18: Cuboid abstractions of airplanes.

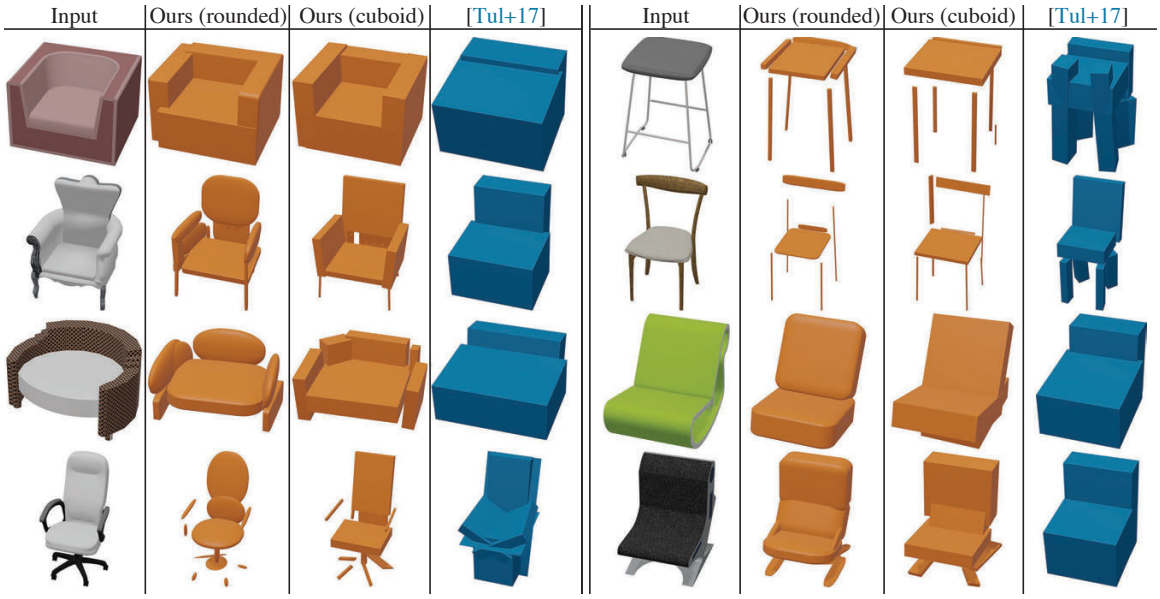



Figure 3-19: Abstractions of test set chairs using our method and the method of [Tul+17].

itives, and we discard cuboids with high overlap using the method of [Tul+17]. The resulting abstractions capture high-level structures of the input.

*Segmentation.* Because we place cuboids consistently, we can use them for segmentation. Following [Tul+17], we demonstrate on the COSEG chair dataset. We first label each cuboid predicted by our network (trained on ShapeNet chairs) with a segmentation class (seat, back, legs). Then, we generate a cuboid decomposition of each chair and segment according to the nearest cuboid. We achieve a mean accuracy of 94.6%, exceeding the 89.0% accuracy of [Tul+17].

*CSG operations.* In Fig. 3-20, we show results of a network that outputs parameters for the union of eight rounded cuboids minus eight rounded cuboids. For inputs compatible with this template, we get good results. It is unclear how to achieve unsupervised CSG predictions using Chamfer loss.

### 3.5.3 3D: Patch-based CAD modeling

We introduce a pipeline for generating sketch data and train a network that coverts a sketch image to a patch-based 3D representation. We show results on synthetic and human-drawn sketches, demonstrate interpolation and quad meshing, compare to prior work, and do an ablation study.

Figure 3-20: Chair CSG abstractions using rounded cuboids.

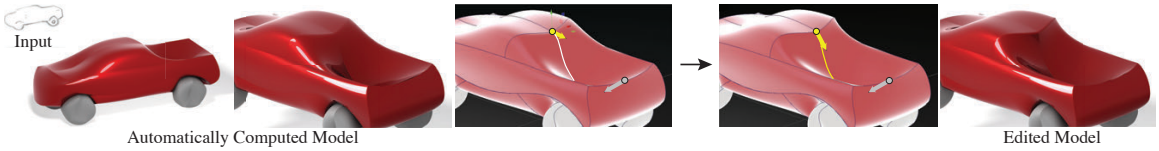



Figure 3-21: Editing a 3D model produced by our method. Because we output 3D geometry as a collection of consistent, well-placed NURBS patches, edits can be made in conventional CAD software by simply moving control points. Here, we refine the trunk of a car with just a few clicks.

### Data preparation

While there exist some annotated datasets of 3D models and corresponding hand-drawn sketches [Gry+19], such data are unavailable at the deep learning scale. Instead, we generate synthetic data. Guided by [Col+08], we first render occluding contours and sharp edges using the Arnold Toon Shader in Autodesk Maya for each model from representative camera views. Although the contour images capture the main features of the 3D model, they lack some of the ambiguities of rough hand-drawn sketches [LRS18]. To this end, we vectorize the images using [BS19] and augment the set of contours by stochastically splitting or truncating curves. Finally, we rasterize each contour image using different stroke widths and pass it through the pencil drawing generation model of Simo-Serra, Iizuka, and Ishikawa [SII18]. We illustrate this pipeline in Fig. 3-7a.

Thus, we obtain sketch images paired with 3D models. We train on the airplane, bathtub, guitar, bottle, car, mug, gun, and knife categories of ShapeNet Core (v2) [Cha+15]. We make the meshes watertight using [HSG18] and normalize them to fit an origin-centered unit sphere.

### Real and synthetic sketch reconstruction

We pick a random 10%-90% test-train split for each category and evaluate in Fig. 3-22.

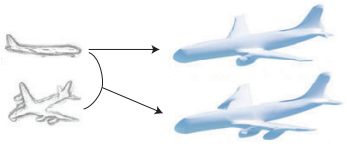
The templates for airplanes, guitars, guns, knives, and cars are generated automatically using

Figure 3-22: Results on synthetic sketches. For each category, from top to bottom: input sketch, output 3D model with sphere template (54 patches), output 3D model with category-specific template.

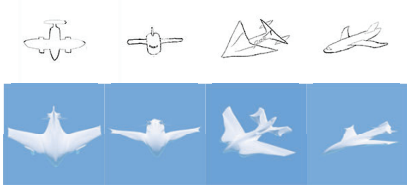
segmentations of Yi et al. [Yi+16]. For mugs, we start with an automatic template and add a hole in the handle and a void in the interior. To demonstrate a template with distinct parts, for cars, we use the segmentation during training, computing Chamfer and normal losses for wheels and body separately. For bottles and bathtubs, we simply place two and and five cuboids, respectively.

With a generic sphere template, we produce a compact piecewise-smooth surface of comparable quality to the more conventional deformable meshes. Our algorithmic construction of category-specific templates, however, enables higher-quality reconstruction of sharp features.

We demonstrate our method's ability to incorporate details from different views. We show our output when given a single view of an airplane as well as when given an additional view. The combined model incorporates elements not visible in the original view.



We also test on real sketches. Each artist was asked to sketch an airplane without having seen any sketches from our dataset. The results are similar to those on synthetic sketches, demonstrating that our dataset is reflective of choices that humans make when sketching 3D objects.



### 3D model interpolation

Our representation is well-suited for interpolating between 3D models. As each model is composed of consistently-placed patches, we linearly interpolate patch parameters (e.g., vertex positions) to generate models "between" outputs. We also interpolate in the latent space (the output of the first 1024-dimensional hidden layer). While the two interpolations are similar, each latent-space interpolant better resembles a realistic model due to learned priors. We show patch-space (right) and latent-space (left) interpolation between two cars.

### Quad meshing and NURBS decomposition

Many algorithms have been designed for converting triangle meshes into NURBS patches [KL96; EH96; Ber+99]. By fitting a template to a 3D model, our method automatically converts the model to a set of Coons patches. Moreover, our Coons patches are placed *consistently*, thus estab-

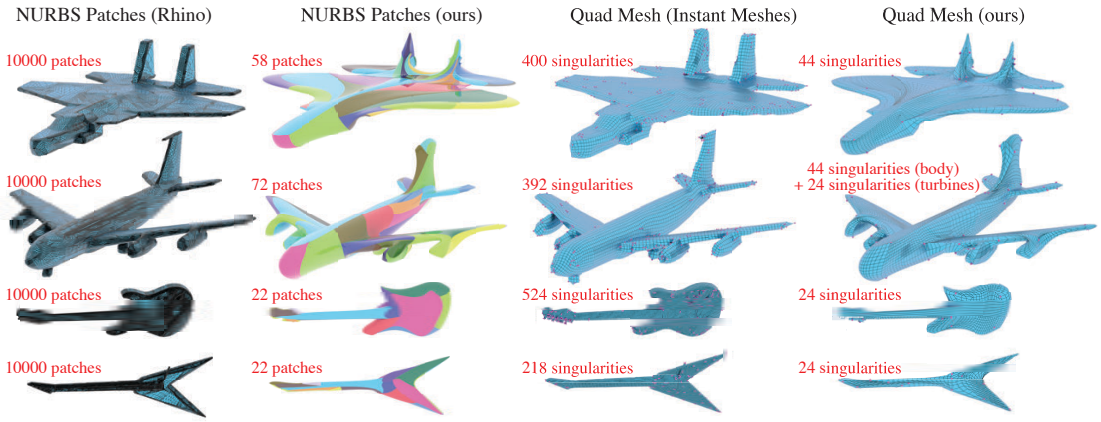



Figure 3-23: We convert two airplane and two guitar models to NURBS patches using Rhino 3D and compare to our Coons patches. Corresponding Coons patches across models of the same category are the same color. We also show quad meshes generated using Instant Meshes [Jak+15] and those from our patch decompositions. Singular points are in pink. Our representation is much more compact and hence easily editable. We produce fewer singularities and only in known places. Unlike with other methods, our patches and quad meshes are consistent across models.

lishing correspondences between models. In Fig. 3-23, we compare our Coons patches to NURBS patches automatically obtained using Rhino 3D, a commercial CAD program. Our decomposition is significantly more sparse and usable for further editing.

A common task in computer graphics is converting surfaces into quad meshes. We can easily obtain a quad mesh from our Coons patch decomposition by subdividing each patch. In order for the quad mesh to be uniform, we determine the number of subdivisions for each patch boundary curve by solving an integer linear problem (ILP). Our free variables are the numbers of subdivisions per curve, and the objective encourages the number of subdivisions to be proportional to the arc length of each curve. Constraints ensure that opposite curves for each patch are equally subdivided, and when one curve contains multiple other curves (due to edge junctions), the numbers of subdivisions are compatible. In particular, we solve the following ILP:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i \in I} (x_i - \alpha s_i)^2 \\
\text{s.t.} \quad & x_i = x_{\mathcal{O}(i)} && \forall i \in \mathscr{E} \\
& x_i = \sum_{j \in \mathcal{F}(i)} x_j && \forall i \in \mathscr{E} \\
& x_i \in \mathbb{Z}^+ && \forall i \in \mathscr{E},
\end{aligned}
$$

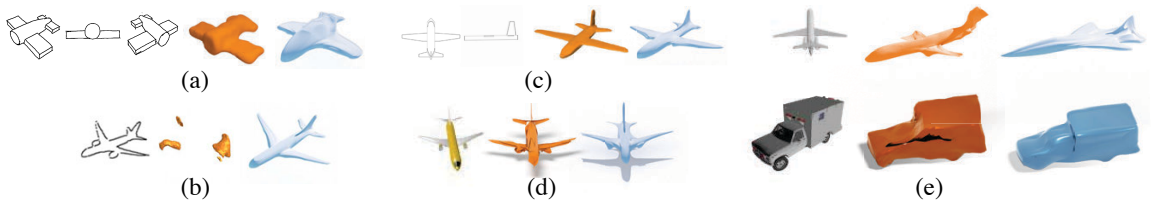



Figure 3-24: Comparisons to [Del+18] using their data (a) and our data (b), to [Lun+17] (c), to AtlasNet [Gro+18] (d), and to Pixel2Mesh [Wan+18b] (e). Our results are in blue.

where $x_i$ is the number of subdivisions for curve $i$, $s_i$ is the arc length of curve $i$, $\mathcal{C}$ contains all of the curves, $\mathcal{O}$ maps a curve to the opposite curve in its patch, $\mathcal{J}$ maps a curve to a list of curves contained within it, and $\alpha$ is a parameter that controls the coarseness of the resulting quad mesh.

We solve the ILP using MOSEK, subdivide our Coons patches to obtain a quad mesh, and then perform surface-preserving Laplacian smoothing in MeshLab for 10 iterations with maximum angle displacement of 5°.

We compare our quad meshes to those produced by Instant Meshes [Jak+15], a recent quad meshing algorithm, in Fig. 3-23. It is commonly desired to minimize the number of singular points (vertices with valence not equal to four) and to consistently position them in a quad mesh. Our algorithm achieves both of these goals—the number and location of singularities is determined by our template. While the Instant Meshes results contain between 215 and 524 singular points, our quad meshes contain only 24 (guitars), 44 (airplane without turbines), or 68 (airplane with turbines).

### Comparisons

We compare to the sketch-based reconstruction methods of Delanoy et al. [Del+18] (Fig. 3-24a) and Lun et al. [Lun+17] (Fig. 3-24c). Although we use the same species of input used to train these methods rather than attempting to re-train their models on our data, the visual quality of our predictions is comparable to theirs. Moreover, our output representation sparsely captures smooth and sharp features, independent of resolution. In contrast, Delanoy et al. [Del+18] produce a $64^3$ voxel grid—a dense, fixed-resolution representation, which cannot be edited directly and offers no topological guarantees. In Fig. 3-24b, we evaluate their system on contours from our dataset, demonstrating that our task of reconstruction with a prior on class (airplane) rather than structure (cylinders and cuboids) is misaligned with theirs: Since our data is not well-approximated by

CSG models, their method cannot extract meaningful output.

Although Lun et al. [Lun+17] ultimately produce a mesh, they perform a computationally expensive post-processing procedure, since their forward pass returns a labeled point cloud. Our method directly outputs the parameters for surface patches with no further optimization. Additionally, their final mesh contains more components (triangles) than our output (patches), making it less useful for editing.

In Fig. 3-24d, we compare to AtlasNet [Gro+18]. We retrain our model with their renderings, using the 54-face sphere template. While our 3D reconstructions capture the same degree of detail, they do not suffer from topological defects. In particular, AtlasNet's surfaces contains many patch intersections and holes. Extracting a watertight mesh would require significant post-processing. Additionally, each patch in our representation is parameterized sparsely by control points, in contrast to AtlasNet's patches, which must be sampled using a deep decoder network.

In Pixel2Mesh, Wang et al. [Wan+18b] input an image and output a triangle mesh. We train our sphere template models on their data. While their meshes have 2466 vertices (*7398 degrees of freedom*) we output 54 patches (*816 degrees of freedom*)—a more editable and interpretable representation. As shown in Fig. 3-24e, the low dimensionality of our output is not at the expense of expressiveness.

We compare to Pixel2Mesh quantitatively in Table 3.2. We select 2500 random test set views and compute Chamfer distance using 5000 sampled points. While obtain comparable Chamfer distance values, our representation is significantly more compact, editable, and less prone to non-manifold artifacts.

Table 3.2: Quantitative comparison to Pixel2Mesh. We report Chamfer distance (CD) and degrees of freedom in the representation (DOF). We obtain comparable Chamfer distance using a representation that is an order of magnitude more compact and without non-manifold artifacts.

| | CD | | DOF | |
|---|---|---|---|---|
| Category | P2M | ours | P2M | ours |
| airplane | 0.022 | 0.025 | 7398 | 816 |
| car | 0.018 | 0.022 | 7398 | 816 |

### Ablation study

We perform an ablation study of our method on an airplane model, demonstrating the effect of training without each term in our loss function as well as the difference between a category-specific template, a 54-patch sphere template, and a lower resolution 24-patch template. The results are shown in Fig. 3-25. We also show an ablation results on the knife model in Fig. 3-26.

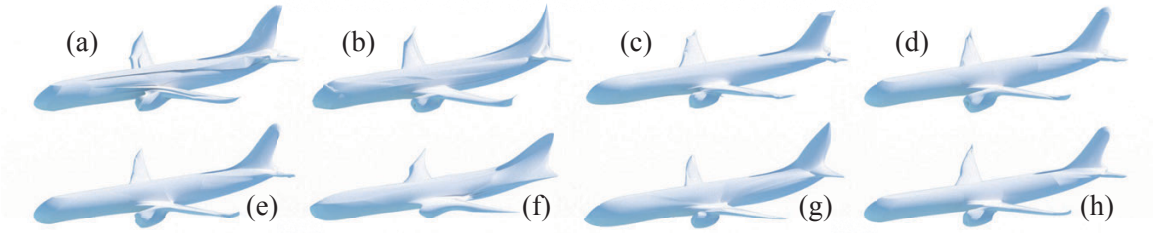



Figure 3-25: An ablation study of our model without normal alignment loss (a), without collision detection loss (b), without patch flatness loss (c), without template normal loss (d), without symmetry loss (e), as well as using 24-patch (f) and 54-patch (g) sphere templates compared to the final result (h).

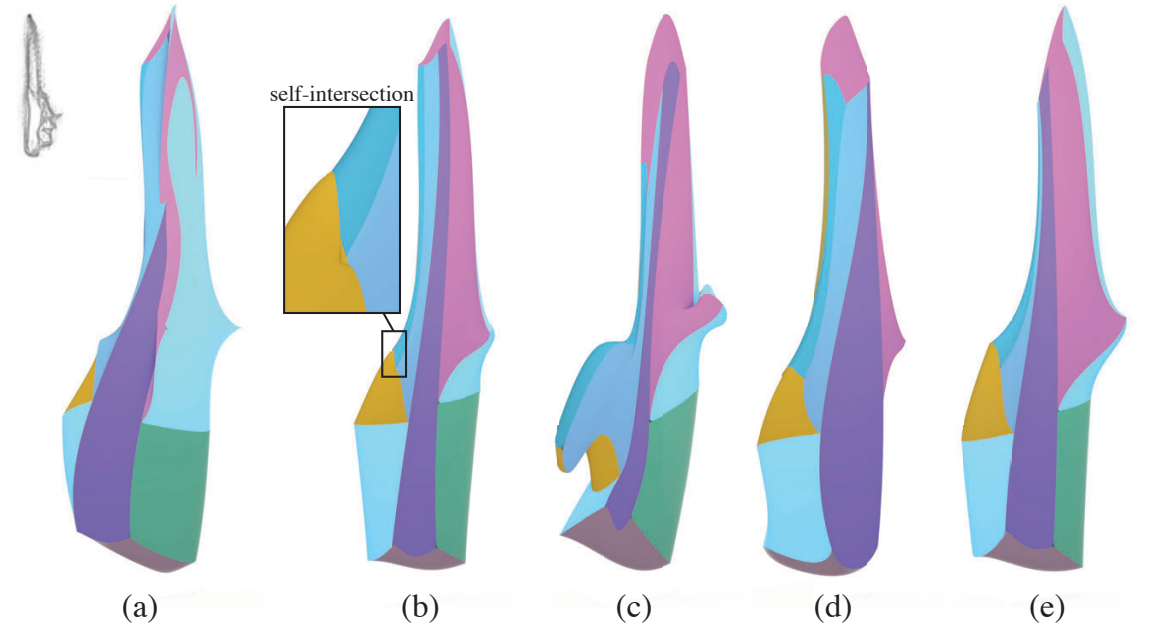



Figure 3-26: An ablation study of our model without normal alignment loss (a), without collision detection loss (b), without patch flatness loss (c), without template normal loss (d), compared to the final result (e).

The ablation study demonstrates the contribution of each component of our system method

to the final result. Training without collision detection loss results in patch intersections. Omitting the normal loss causes the 3D surface to suffer in smoothness. Patch flatness and and template normal losses encourage patch seams to align to sharp features. While both sphere templates capture the geometry, using more patches captures greater details, and using a non-generic template further improves the model.

We additionally provide a quantitative ablation study for the knives and airplanes categories in Table 3.3. We train a model without each of the normals, flatness, template normals, and collision loss terms for 50000 steps. for each model, we report the average Chamfer loss, normals loss, flatness loss, and template normals loss (as defined in Section 3.4.2) on the test set, in addition to the average number of intersecting patch pairs. The results demonstrate that each loss term contributes to our full model without sacrificing reconstruction quality.

Table 3.3: Quantitative ablation study of our loss function.

| Model | Ch. Loss | Norm. Loss | Flat. Loss | T.N. Loss | # Int. Pairs |
|---|---|---|---|---|---|
| airplanes (full) | 0.0118 | 0.725 | 0.0000732 | 0.792 | 0.0220 |
| airplanes (no norm.) | 0.0105 | 1.09 | 0.0000941 | 0.935 | 0.0357 |
| airplanes (no flat.) | 0.0119 | 0.748 | 0.000254 | 0.866 | 0.0302 |
| airplanes (no t.n.) | 0.0114 | 0.723 | 0.0000734 | 0.889 | 0.0110 |
| airplanes (no coll.) | 0.0112 | 0.727 | 0.000113 | 1.07 | 3.09 |
| knives (full) | 0.0124 | 0.635 | 0.0000950 | 0.669 | 0 |
| knives (no norm.) | 0.0124 | 1.12 | 0.0000902 | 1.46 | 3.95 |
| knives (no flat.) | 0.0122 | 0.648 | 0.00132 | 0.733 | 0.0263 |
| knives (no t.n.) | 0.0126 | 0.671 | 0.0000886 | 1.03 | 0.132 |
| knives (no coll.) | 0.0122 | 0.631 | 0.0000963 | 0.722 | 0.0263 |

## 3.6 Discussion

Representation is a key theme in deep learning—and machine learning more broadly—applied to geometry. Assorted means of communicating a shape to and from a deep network present varying

tradeoffs between efficiency, quality, and applicability. While considerable effort has been put into choosing representations for certain tasks, the tasks we consider have *fixed* representations for the input and output: They take in a shape as a function on a grid and output a sparse set of parameters. Using distance fields and derived functions as intermediate representations is natural and effective, not only performing well empirically but also providing a simple way to describe geometric loss functions.

Our learning procedures are applicable to many additional tasks. A natural next step is to incorporate our network into more complex pipelines for tasks like vectorization of complex drawings [BS19], for which the output of a learning procedure needs to be combined with classical techniques to ensure smooth, topologically valid output. A challenging direction might be to incorporate user guidance into training or evaluation, developing the algorithm as a partner in shape reconstruction rather than generating a deterministic output.

Our experiments suggest several extensions for future work. The key drawback of our first distance field–based approach is the requirement of closed-form distances for the primitives. While there are many primitives that could be incorporated this way, a fruitful direction might be to alleviate this requirement, e.g. by including flexible implicit primitives like metaballs [Bli82]. We could also incorporate more boolean operations into our pipeline, which easily supports them using algebraic operations on signed distances, in analogy to the CAD pipeline, to generate complex topologies and geometries with few primitives. The combinatorial problem of determining the best sequence of boolean operations for a given input would be particularly challenging even for clean data [Du+18]. Finally, it may be possible to incorporate our network into *generative* algorithms to create new unseen shapes.

# 4

# Learning Manifolds with Boundary

While the parametric shape representations we were able to learn in the previous chapter were easy and intuitive to control and manipulate, they were limited in expressiveness. Indeed, using simple primitives like Coons patches or cuboids to model complicated or high-frequency geometry quickly becomes infeasible . In this chapter, we propose a hybrid shape representation that exhibits some of the control of an explicit representation while also enjoying the expressiveness of neural implicit representations.

## 4.1 Introduction

Shape representation is a crucial component of geometry processing and learning algorithms. Depending on the target application, different representations have varying tradeoffs. As we previously discussed in Section 3.1, shape representations broadly fall into two classes: *Lagrangian* or *explicit* and *Eulerian* or *implicit*. In this work, we show how to use the theory of *currents* from geometric measure theory to design a flexible neural representation that combines favorable aspects from each category, representing the interiors of surfaces implicitly while maintaining an explicit boundary representation.

Lagrangian representations encode a shape by giving coordinates of points or parameterizing regions of the shape. To represent a curve in a Lagrangian way, one might give coordinates of

This chapter includes material from the following publication: [Smi+22].

successive points along the curve. Analogously, to represent a surface in 3D, one might use a mesh, which assembles the surface out of simple patches. Lagrangian representations afford great precision but require predetermined combinatorial structures, making it difficult to represent families of shapes with varying topology.

In contrast, Eulerian representations encode a shape via a function on some background domain. For example, a surface might be encoded as the *level set* of a scalar function sampled on a regular grid. Level sets of signed distance fields (SDFs) form one popular implicit representation. Implicit functions naturally capture topological variation, but traditional implicit shape representations, in which the background geometry must be discretized with a fixed grid or mesh, waste resolution on regions far away from the level set of interest. Recent *neural implicit representations* alleviate this problem [CZ19; Mes+19; Par+19]. The universal approximation and differentiability properties of neural networks make them an appealing alternative to regular grid discretizations.

Neural implicit representations come with their own limitations. Like other implicit representations based on level sets, most neural implicit representations can only encode closed surfaces, which lack boundary curves. Boundaries can provide manipulation handles for controllable deformation, and precisely-defined boundaries can be used to stitch together surfaces into a larger articulated surface.

In this chapter, we describe a new way to encode neural implicit surfaces with boundaries. The key to our representation is the theory of currents from geometric measure theory. In this theory, $k$-dimensional submanifolds are defined by their integration against differential $k$-forms, generalizing how distributions (0-currents) are defined by integration against smooth functions. Current spaces are complete normed linear spaces that make optimization over surfaces convenient, and the boundary operator also becomes linear on these spaces. Classically, currents were the key to solving Plateau's *minimal surface* problem by transforming it into *mass norm minimization*. We adopt the mass norm as the primary loss function encouraging our neural currents to converge to smooth surfaces.

We demonstrate our representation with three applications. We first demonstrate how it enables computing minimal surfaces efficiently through stochastic gradient descent. Then, by modifying the background metric used to define the mass norm, we reconstruct arbitrary surfaces

from data. Finally, we demonstrate the flexibility of our representation by encoding families of surfaces with explicit boundary control.

*Contributions.* In summary, we

- propose a new neural implicit surface representation with explicit boundary curves;
- show how to use stochastic gradient descent on the mass norm to compute minimal surfaces;
- introduce a custom background metric and additional loss terms to represent surfaces from data; and
- describe a framework for learning families of surfaces parameterized by their boundaries along with a latent code.

## 4.2 Related work

Our work takes classical ideas in minimal surface computation and brings them into the context of modern deep learning to form a new neural shape representation. Below, we summarize key prior works in these two areas.

### 4.2.1 Minimal Surface Computation

Most computational approaches to minimal surface generation use a mesh or grid representation of the surface. In the twentieth century, numerical minimal surface problems were discretized by finite difference methods on a grid, assuming the surfaces were funieption graphs [Dou27; Con67]. Grid-based methods were later adapted to triangle meshes [Wil61; HSK74], allowing the generated surface to leave the space of function graphs [Wag77]. Modern mesh-based minimal surface solvers use mean curvature flow [Dzi90; Bra92; Des+99], stretched grids [Pop96], quasi-Newton iterations [PP93; SW19], Voronoi tessellations [Pan+12], or curvature flows with a conformal constraint [KSB12; CPS11]. These methods based on explicit surface representations are straightforward, but the optimization often suffers from local minima due to non-convexity and can even diverge [Wag77; PP93] if the initial mesh has the wrong topology.

A different approach to the minimal surface problem is based on *geometric measure theory* (GMT), whose theoretical foundations were developed in the 1960s [Fed96; Mor16; Fle15]. In this theory, curves and surfaces are represented implicitly by *currents* as dual to differential forms. Such representations have been used in geometry processing [Mul+07; BLM18; MC19] and medical imaging [CT14; VG05; GTY04; Dur+08; Dur+09; Dur+11]. In geometric measure theory, the minimal surface problem becomes the convex *minimal mass norm* problem (see Section 4.3.4). A discrete analog of the minimal mass problem on a graph is a linear program known as the optimal homologous chain problem [Sul90; DH11; DHK11; CLV20]. GMT-based discretization of the minimal surface problem in Euclidean space was pioneered by [PP97] and revisited by [BM19; WC21].

### 4.2.2 Deep Learning for Shape Reconstruction

Using deep learning to produce 3D geometry has gained popularity in vision and graphics. Network architectures now can output many explicit shape representations, like voxel grids [Del+18; Zha+18; Wu+18], point clouds [FSG17b; Yin+18; Yan+19], meshes [Nas+20; Wan+18b; Han+20], and parametric primitives [Sha+20; Tul+17; PUG19]. While the content produced by these approaches is generally easy to render and manipulate, it is often restricted in topology and/or resolution, limiting expressiveness.

A different approach circumvents topology and resolution issues by representing 3D shapes *implicitly*, using functions parameterized by neural networks. In DeepSDF, Park et al. [Par+19] learn a field that approximates signed distance to the target geometry, while Mescheder et al. [Mes+19] and Chen and Zhang [CZ19] classify query points as being outside or inside a shape. Others further improve the results by proposing novel regularizers, loss functions, and training or rendering approaches [Gro+20; AL20; Tak+21; Lip21]. While these works achieve impressive levels of detail in surface reconstruction, they largely suffer from two drawbacks—lack of control and inability to represent open surfaces, i.e., those with boundary.

Neural implicit learning methods typically overfit to a single target shape or learn a family of shapes parameterized by a high-dimensional latent space. While recent work has shown the possibility of adapting classical geometry processing algorithms to neural implicit geometries [Yan+21], applying targeted manipulations and deformations to learned shapes remains nontrivial. Several

papers propose *hybrid representations*, combining the expressive power of neural implicit representations with the control afforded by explicit geometries. Genova et al. [Gen+20] reconstruct shapes by learning multiple implicit representations arranged according to a learned template configuration. In DualSDF [Hao+20], manipulations can be applied to learned implicit shapes by making changes to corresponding explicit geometric primitives. BSP-Net [CTZ20] and CvxNet [Den+20] restrict the class of learned implicit surfaces to half-spaces and convex hulls, respectively. [Liu+21] defines local implicit functions on point clouds, facilitating the transition between discrete points and smooth surfaces during training. Our DeepCurrents adopt a hybrid representation, which models boundaries explicitly and allows them to be used as handles for manipulation.

## 4.3 Preliminaries

Geometric measure theory is a vast field that we will not attempt to summarize here. For a comprehensive treatment, we refer the reader to [Sim14; Fed96; Lan04]. We focus on the rudiments necessary to construct our optimization problem in dimensions two and three, eliding technical issues that arise in higher-dimensional ambient spaces.

The theory of currents is motivated by solving *Plateau's Problem*, the problem of finding the surface of minimal area enclosed by a given boundary:

$$\arg\min_{\Sigma}\{A(\Sigma) : \partial\Sigma = \Gamma\}. \tag{4.1}$$

The problem (4.1) seeks a solution in the space of smooth embedded submanifolds with boundary, which lacks convenient properties such as convexity and compactness required for reasoning about optimization. In GMT, this space is *relaxed* to a space of *currents*, generalized submanifolds characterized by integration. Plateau's problem (4.1) is systematically translated into a problem over currents. The area functional becomes the *mass norm*, and $\partial$ becomes a linear operator constructed by dualizing the exterior derivative d. We describe this translation in detail below.

### 4.3.1 Currents

Currents are to submanifolds as distributions are to sets of points. Just as distributions are characterized by integration against functions, $k$-currents are characterized by integration against differential $k$-forms in the ambient space. For our purposes, that ambient space will be an open subset $U \subseteq \mathbb{R}^d$, $d \leq 3$. For now, we will assume the metric is Euclidean; see Section 4.4.2 for the generalization to Riemannian metrics. We will also assume that $U$ is bounded and simply connected to elide various technical issues.

We denote the space of smooth $k$-forms with compact support in $U$ by $\Omega_c^k(U)$. Recall that a $k$-form $\zeta \in \Omega_c^k(U)$ smoothly assigns to each point $x \in U$ an element $\zeta_x \in \bigwedge^k T_x^* U$, the exterior power of the cotangent space at $x$. In Euclidean space, there is a canonical identification between covectors ($k = 1$) and vectors. A Riemannian metric provides a similar identification but requires more careful bookkeeping (see Section 4.4.2).

The space of $k$-currents

$$\mathscr{D}_k(U) = (\Omega_c^k(U))^* \tag{4.2}$$

is the *dual space* of (compactly-supported) $k$-forms, i.e., it consists of continuous linear functionals on $k$-forms. An element $T \in \mathscr{D}_k(U)$ is defined by its assignment of real values to $k$-forms:

$$\zeta \in \Omega_c^k(U) \mapsto T(\zeta) \in \mathbb{R}. \tag{4.3}$$

The following are two key examples of currents:

- A 0-current is simply a distribution, as

$$\mathscr{D}_0(U) = (\Omega_c^0(U))^* = (C_c^\infty(U))^* = \mathscr{D}(U). \tag{4.4}$$

- A submanifold $\Sigma \subset U$ of dimension $k$ can be viewed as a current $[\Sigma] \in \mathscr{D}_k(U)$ by integration against it:

$$[\Sigma](\zeta) := \int_\Sigma \zeta. \tag{4.5}$$

### 4.3.2 Boundary operator

In generalizing the boundary operator from submanifolds to currents, we need to ensure that $\partial[\Sigma] = [\partial\Sigma]$. Stokes' Theorem tells us that

$$[\partial\Sigma](\zeta) = \int_{\partial\Sigma} \zeta = \int_{\Sigma} \mathrm{d}\zeta = [\Sigma](\mathrm{d}\zeta), \tag{4.6}$$

motivating the definition

$$\partial T(\zeta) := T(\mathrm{d}\zeta). \tag{4.7}$$

In words, we define $\partial$ as the adjoint of d.

### 4.3.3 Mass norm

As we did for the boundary operator, we write the area functional in terms of integration against forms and then replace integration by current evaluation. This definition depends on a pointwise norm $|\cdot|$ on $k$-forms. As we are working in dimensions $d \leq 3$, it is sufficient to use the pointwise inner product norm.

If $\Sigma \subset U$ is a smooth $k$-submanifold with boundary, then its area satisfies

$$A(\Sigma) = \sup_{\zeta \in \Omega_c^k(U)} \left\{ \int_{\Sigma} \zeta : |\zeta_x| \leq 1 \ \forall x \in U \right\}. \tag{4.8}$$

So we define the *mass norm* of a current $T \in \mathscr{D}_k(U)$ as:

$$\mathbf{M}(T) := \sup_{\zeta \in \Omega_c^k(U)} \{T(\zeta) : |\zeta_x| \leq 1 \ \forall x \in U\}. \tag{4.9}$$

### 4.3.4 Minimal mass problem

Applying the transformations above to the problem (4.1), one obtains a relaxation known as the *minimal mass problem*:

$$\min_{T \in \mathscr{D}_k(U)} \{\mathbf{M}(T) : \partial T = \Gamma\}. \tag{4.10}$$

In classical GMT, the current $T$ is taken to be in the space $\mathscr{I}_k(U)$ of *integral k-currents*, which roughly means currents that look like integer linear combinations of Lipschitz surfaces. When optimizing over $\mathscr{I}_{d-1}(U)$ in ambient dimension $d \leq 7$, there is an optimal solution corresponding to a smooth submanifold (see [Fed96] Theorem 5.4.15, [Sim14] Theorem 5.8, [Lan04] Theorem 3.10). More recent theory extends this result to optimization over general currents $\mathscr{D}_k(U)$ (see [BM19] Theorem 2, [Sim14] Remark 5.2).

### 4.3.5 Representing currents by forms

For computational purposes, we follow [WC21] and optimize over $k$-currents represented by differential $(d-k)$-forms. This allows us to represent currents by neural networks.

A $(d-k)$-form can be identified with a $k$-current $[\omega] \in \mathscr{D}_k(U)$ by defining

$$[\omega](\zeta) := \int_U \omega \wedge \zeta$$

for any $\zeta \in \Omega_c^k(U)$. With this identification, we have by Stokes' Theorem:

$$\begin{aligned} \partial[\omega](\zeta) = [\omega](\mathrm{d}\zeta) &= \int_U \omega \wedge \mathrm{d}\zeta \\ &= (-1)^{d-k+1} \int_U \mathrm{d}\omega \wedge \zeta = \left[(-1)^{d-k+1}\mathrm{d}\omega\right](\zeta). \end{aligned} \tag{4.11}$$

The boundary constraint $\partial[\omega] = T$ thus becomes an exterior differential equation,

$$\mathrm{d}\omega = \delta_T, \tag{4.12}$$

where $\delta_T$ is a singular $(d-k)$-form representing $T$.

Similarly, the mass norm of a $k$-current becomes the $L^1$ norm of a $(d-k)$-form:

$$\mathbf{M}([\omega]) = \|\omega\|_1 = \int_U |\omega(x)| \mathrm{dvol} \tag{4.13}$$

where dvol is the volume form.

## 4.4 DeepCurrents

In the previous section, we described the relaxation of Plateau's minimal surface problem into a convex optimization problem over a space of currents, with the property that its optima include smooth surfaces, and we showed how to represent certain currents by differential forms. Now, we introduce our novel neural representation of currents and SGD mass minimization.

### 4.4.1 Neural representation

The linear space of solutions to (4.12) can be parameterized using the Hodge decomposition:

$$\omega = \mathrm{d}f + \alpha, \tag{4.14}$$

where $\alpha \in \Omega^{d-k}(U)$ is any particular solution of (4.12), $f \in \Omega^{d-k-1}(U)$; we ignore the harmonic term as $U$ is simply connected. For curves in $U \subset \mathbb{R}^2$ ($k = 1, d = 2$) and surfaces in $U \subset \mathbb{R}^3$ ($k = 2, d = 3$), $f$ will simply be a function on $U$, which we can represent by a neural network. It is convenient to use the (Euclidean) musical isomorphism $\#$ to encode our 1-form $\omega$ as the vector field $\omega^\#$. Intuitively, the vector field corresponding to a current points in the surface normal direction. Under this identification, $(\mathrm{d}f)^\# = \nabla f$, which can be computed by autodifferentiation.

As for $\alpha$, there is a particularly convenient choice known as the *Biot-Savart field*, which can be written in closed form when $\Gamma$ is a polygonal curve (see [WP09]):

$$\alpha^\#(x) := \int_\Gamma \frac{d\vec{\ell} \times \vec{r}}{|\vec{r}|^3} = \sum_i \frac{(\hat{t}_i \cdot (\hat{r}_i^1 - \hat{r}_i^0))(\hat{t}_i \times \vec{r}_i^0)}{|\hat{t}_i \times \vec{r}_i^0|^2}, \tag{4.15}$$

where $d\vec{\ell}$ denotes the vector arc measure on $\Gamma$, $\hat{t}_i$ is the tangent vector to the $i$th segment of $\Gamma$, $\vec{r}_i^0$ and $\vec{r}_i^1$ are, respectively, the vectors from the point $x$ to the initial and final vertices of segment $i$, and $\hat{r}_i^0$ and $\hat{r}_i^1$ are their normalized directions. In practice, we scale $\alpha$ by $10^{-3}$ to better match the normalization of our network weights; this only changes the mass minimization problem by a uniform scale. Figure 4-1a visualizes a Biot-Savart field in 2D.

Enacting the choices above, we can use neural networks to solve the minimal mass problem:

$$\arg\min_{\theta} \|\mathrm{d}f_\theta + \alpha_T\|_1 = \arg\min_{\theta} \mathbb{E}_{x\sim\mathcal{U}_U}\left[\left|\nabla_x f_\theta(x) + \alpha_T^\#(x)\right|\right], \tag{4.16}$$

where $f_\theta$ is a neural network with weights $\theta$, $\mathcal{U}_U$ is the uniform distribution over $U = [-1, 1]^d$. $\nabla$ is computed exactly via automatic differentiation, and $\alpha_T^\#$ is the boundary-dependent Biot-Savart field, computed in closed form. The expectation in (4.16) is approximated by uniform sampling over $U$, yielding a method to compute minimal surfaces via stochastic gradient descent (SGD).

Compared to previous discretizations of currents and the minimal surface problem [WC21], we

1. represent $f$ via a neural network rather than a voxel grid;
2. evaluate $\alpha$ in closed-form; and
3. evaluate the mass norm as an expectation that is amenable to SGD.

These key choices allow our minimal surfaces to achieve arbitrary resolution.

### 4.4.2 Modifying the metric

Critical to the computer vision applications that we consider, we can use a background Riemannian metric to encode general surfaces that are not minimal under the Euclidean metric. The properties of mass norm minimization almost certainly carry over—in particular, the regularity of minima (see e.g., [Mor03]).

Let $g$ be a Riemannian metric given by

$$g(X, Y) = \langle AX, Y\rangle = \langle X, AY\rangle \quad \forall X, Y \in TU, \tag{4.17}$$

where $\langle\cdot\rangle$ is the Euclidean inner product and $A$ is a smoothly varying symmetric positive definite linear map on the tangent bundle ($A_x \colon T_xU \to T_xU$). The Riemannian pointwise norm for a $k$-form $\zeta$ is given by

$$|\zeta_x|_g = |(A_x^{-1/2})^* \zeta_x|, \tag{4.18}$$

where $|\cdot|$ is the Euclidean pointwise norm, and $B^*\zeta$ denotes the pullback form $B^*\zeta(X_1, \dots, X_k) = \zeta(BX_1, \dots, BX_k)$. The $g$–mass norm is then:

$$\mathbf{M}_g([\omega]) = \|\omega\|_{1,g} = \int_U |(A^{-1/2})^*\omega| (\det A)^{d/2} \mathrm{dvol}. \tag{4.19}$$

The differential equation (4.12) and its solution (4.14) are topological and do not change.

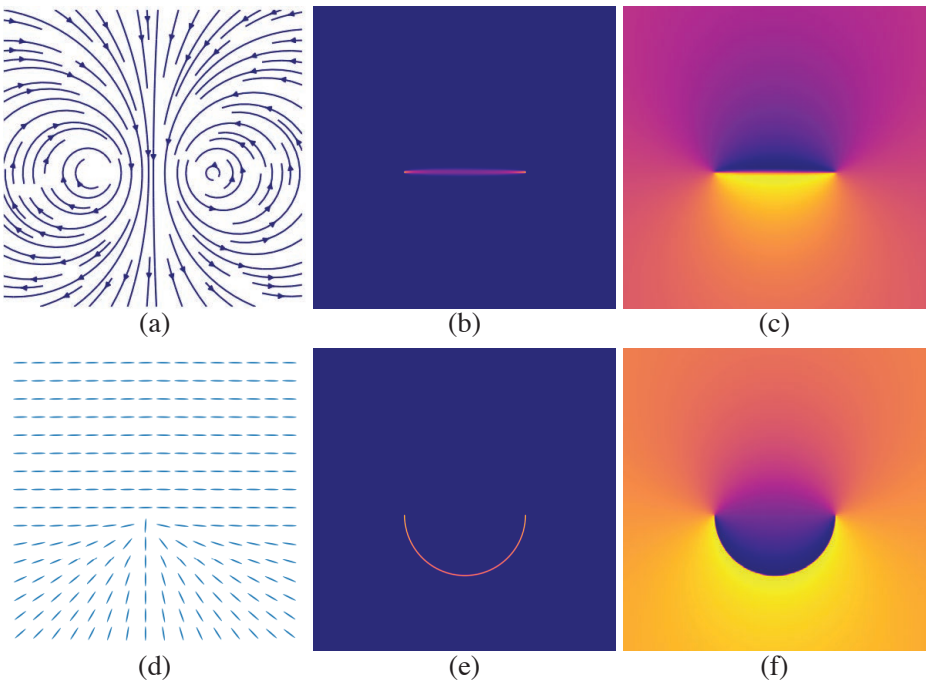



Figure 4-1: Minimizing the mass norm $\|df + \alpha\|_1$ under the Euclidean metric in two dimensions yields a line segment connecting the two boundary points (b). With our custom data-dependent background metric, we can reconstruct the semicircle as a current (e). $\alpha$ is shown as a vector field (a) and the custom metric is depicted by oriented ellipsoids (d, not to scale). Corresponding functions $f$ are shown at right (c and f).

In summary, the Riemannian problem differs from the Euclidean one by a symmetric positive definite matrix $B_x$:

$$\begin{aligned} &\arg\min_\theta \|\mathrm{d}f_\theta + \alpha_I\|_{1,g} \\ &= \arg\min_\theta \mathbb{E}_{x\sim\mathcal{U}_U} \left[ \left| B_x(\nabla_x f_\theta(x) + \alpha_I^\#(x)) \right| \right]. \end{aligned} \tag{4.20}$$

In the 2D example depicted in Fig. 4-1, minimizing the mass norm under the Euclidean metric yields a straight line segment (Fig. 4-1b). Changing the metric (Fig. 4-1d) yields a semicircle (Fig. 4-1e) instead. Corresponding density plots of $f$ are shown in Figs. 4-1c and 4-1f, respectively.

### 4.4.3 Loss functions

The main objective function optimized by our training procedures is the *current loss*, which follows from (4.20):

$$\mathscr{L}_{\text{curr}}(\cdot) = \mathbb{E}_{x\sim\mathcal{U}_U}\left[\left|B_x(\nabla_x f_\theta(x) + \alpha_T^\#(x))\right|\right]. \tag{4.21}$$

We approximate the expectation by a sample average over a sample drawn from the uniform distribution on $U$.

For minimal surface computation (Section 4.5.1), we set $B_x = I$ for all $x \in U$. For surface reconstruction (Sections 4.5.2 and 4.5.3), we define

$$B_x = w_x(I - \hat{n}_{\text{proj}_\Sigma(x)}\hat{n}_{\text{proj}_\Sigma(x)}^\top), \tag{4.22}$$

where $\Sigma$ is the ground truth surface, $\text{proj}_\Sigma(x)$ is the closest point on $\Sigma$ to $x$, and $\hat{n}_{\text{proj}_\Sigma(x)}$ is its unit normal. This positive semidefinite matrix, corresponding to a degenerate Riemannian metric, penalizes the current's deviation from agreement with the surface's orientation. A patch aligned with $\Sigma$ ($\nabla f + \alpha_T^\# \parallel \hat{n}$) costs zero.

When evaluating (4.20), for half of the samples in $U$, we set $w_x = 1$, and for the other half

$$w_x = \exp\left(-\frac{1}{2\sigma^2}\|x - \text{proj}_T(x)\|_2^2\right), \tag{4.23}$$

where $\text{proj}_T(x)$ is the closest point on the boundary to $x$ under Euclidean distance, and $\sigma = 0.1$ in practice. We find empirically that adding this boundary weighting, where samples close to the prescribed boundary have a higher contribution to the current loss, with a Gaussian falloff, slightly improves our learned surfaces, particularly near the boundary. See Section 4.5.4 for an ablation study.

For surface reconstruction, we employ an additional loss term to guide our optimization. We define *surface loss* as:

$$\mathscr{L}_{\text{surf}}(\cdot) = \mathbb{E}_{x,\varepsilon}\left[\left(\delta - f(x - \varepsilon n_x) + f(x + \varepsilon n_x)\right)^+\right], \tag{4.24}$$

where $x \sim \mathcal{U}_\Sigma$, the uniform measure on the target surface $\Sigma$, $n_x$ is the (oriented) surface normal

vector at a point $x \in \Sigma$, $\varepsilon$ and $\delta$ are small threshold. In practice, we set $\delta = 0.01$, randomly pick $\varepsilon \sim \mathcal{U}_{[0.0199,0.0201]}$, and approximate the expectation by sampling on $\Sigma$.

This hinge loss encourages the values of our learned function $f$ to differ by no less than a margin $\delta$ across the target surface. This objective may seem redundant as the metric (4.22) already encourages alignment to the target surface. In fact, the two are complementary—the surface loss encourages $f$ to jump near the target surface, while the current loss ensures that the bandwidth of the jump decreases. We find that using the surface loss term helps our models converge to better optima (see our ablation study in Section 4.5.4).

### 4.4.4 Network architecture

We learn a single current $\mathrm{d}f_\theta + \alpha$, using a deep neural network to parameterize $f_\theta : \mathbb{R}^3 \to \mathbb{R}$. Given an input point $x \in \mathbb{R}^3$, we first project it onto a random Fourier feature (RFF) space, as in [Tan+20], to obtain $\hat{x} \in \mathbb{R}^{2048}$. Our RFF coefficients are 2048-dimensional and sampled from $\mathbb{N}(0, 4)$. We then decode the RFF vector to a scalar value $f_\theta(x)$ using an MLP $h_\theta$, which consists of three hidden layers, each with 256 units and softplus nonlinearities. This pipeline is illustrated in the top half of Fig. 4-2.

Additionally, we propose a boundary-conditioned autodecoder architecture for learning families of currents (see Fig. 4-2, bottom). We initialize a latent code $z_j \sim \mathbb{N}(0, 0.1)$ for each mesh, and we encode the mesh boundary geometry using a boundary encoder $\mathscr{E}^\Gamma$. For shapes that have more than one boundary, we use a separate encoder for each boundary loop to obtain a set of boundary latent codes $z^{\Gamma_i}$. We then concatenate the latent codes along with the RFFs $[z_j \mid z_j^{\Gamma_1} \mid \ldots \mid z_j^{\Gamma_B} \mid \hat{x}]$ and pass this vector through a decoder $h_\theta$ as in the overfitting setting above.

Our boundary encoder inputs boundary

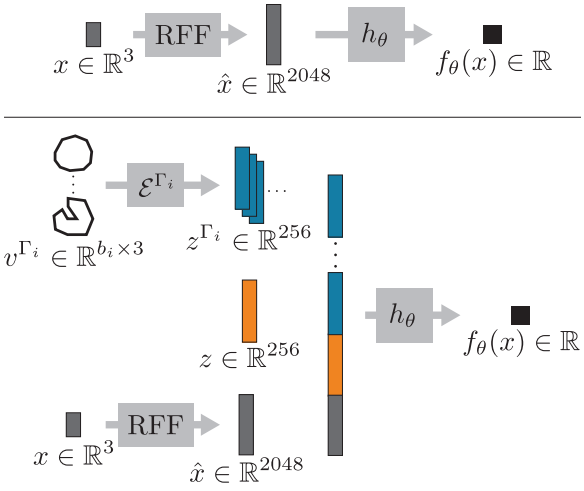



Figure 4-2: An overview of our network architectures for minimal surface optimization and single surface reconstruction (top) as well as shape space learning (bottom). An input point $x$ is first encoded using random Fourier features. These features are then optionally concatenated with latent codes corresponding to shape identity and boundary and finally decoded to a scalar output.

vertices $v^\Gamma \in \mathbb{R}^{b\times 3}$. The encoder is a network with three 1-dimensional convolutional layers with stride 1 and circular boundary conditions. The first layer uses a kernel of size 5 while the latter two use kernels of size 3. Each layer has 256 channels, and we use ReLU after each layer except the last. After the convolutions, we take the mean across all the boundary vertices to obtain the boundary latent code. Circular convolutions combined with mean pooling ensure that our encoder is invariant to cyclic permutations of the vertices, corresponding to the same boundary geometry.

## 4.5 Experimental results

We evaluate DeepCurrents experimentally by demonstrating results on minimal surface computation, overfitting for single surface reconstruction, and shape space learning and interpolation. We also show an ablation study to validate our main design choices. All of our models are trained on a single NVIDIA GeForce RTX 3090 GPU using Adam [KB14].

### 4.5.1 Minimal surfaces

We use our method to compute minimal surfaces for three boundary configurations. We train each model for $10^5$ iterations (~ 12 minutes) with a learning rate of 0.0005, sampling 4096 points from the ambient space at each step and reducing the learning rate by a factor of 0.6 every 10,000 steps. We only optimize $\mathscr{L}_{\text{curr}}$ with the Euclidean metric in these examples—we do not use $\mathscr{L}_{\text{surf}}$ or boundary weighting.

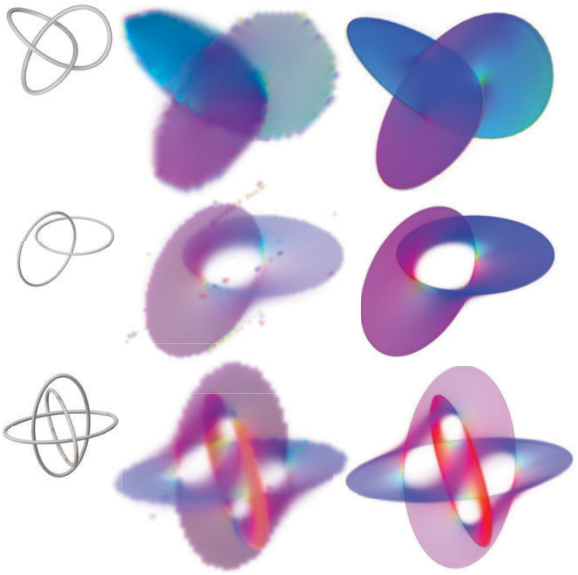


Figure 4-3: Minimal currents computed via [WC21] on a 90 × 90 × 90 grid (middle) display prominent grid artifacts, especially near the boundary. In contrast, with a similar number of parameters (725,249 weights), we achieve higher effective resolution (right). Boundaries (left) are the trefoil knot (top), Hopf link (middle), and Borromean rings (bottom).

Figure 4-3 compares our results to those from [WC21], which uses a voxel grid; the colors represent local current orientation, which corresponds to the surface normal direction. For fair comparison, we choose the grid size to

approximate our number of trainable parameters ($90^3 \approx 725{,}249$). While our learned currents adhere well to the smooth input boundaries, the currents of [WC21] show significant grid artifacts. Thus, our representation exhibits greater capacity to encode high-resolution surfaces with the same number of parameters

### 4.5.2 Surface reconstruction

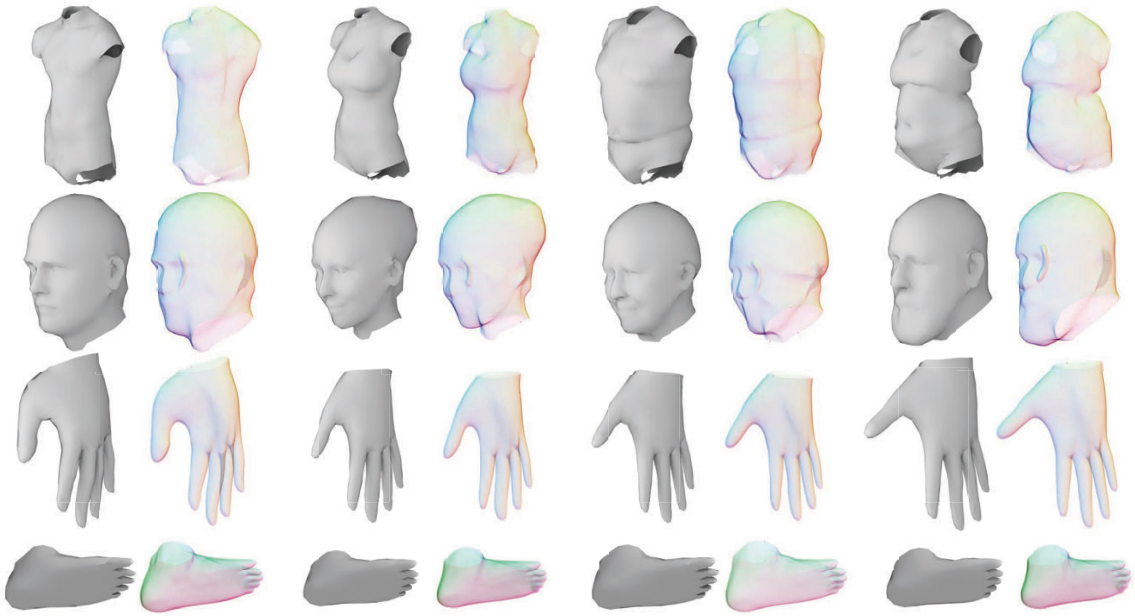


Figure 4-4: Human body surface reconstructions. We overfit DeepCurrents models to reconstruct several torso, head, hand, and foot meshes. We show a volume rendering of each learned current (right) next to the ground truth mesh (left).

We perform surface reconstruction using DeepCurrents by overfitting to several segmented parts of models from the FAUST human body dataset [Bog+14]. We preprocess the data by splitting the mesh according to the provided segmentations, rigidly aligning all the models within each segmentation class, and rescaling them to fit into $[-0.5, 0.5]^3$.

We train each model for 10,000 iterations (~ 4 minutes) with an initial learning rate of 0.001, decayed by a factor of 0.6 every 2000 iterations. We sample 4000 random points from the ambient space (to compute $\mathscr{L}_{\text{curr}}$) and 4000 points from the mesh surface (to compute $\mathscr{L}_{\text{surf}}$) at each step.

We show results on torsos, heads, hands, and feet from randomly chosen models in Fig. 4-4. Our currents faithfully reconstruct the target geometry.

Table 4.1: Quantitative comparison of unidirectional Chamfer distance to [Ven+21] on single surface reconstruction of random models from each of four shape categories.

| Model | UCD ([Ven+21]) | UCD (ours) |
|---|---|---|
| head | 0.0049 | **0.0010** |
| hand | 0.0045 | **0.0011** |
| torso | 0.0049 | **0.00092** |
| foot | 0.0055 | **0.00092** |

Additionally, we compare quantitatively to [Ven+21] in Table 4.1. We train their model for the same amount of time as ours on randomly picked models. Because their model predicts the closest point on the target surface given any input point, we use this to compute unidirectional Chamfer distance (i.e., $\mathbb{E}_{y\sim\mathcal{U}_\Sigma}[\mathrm{dist}_{\Sigma^*}(y)]$, where $\mathcal{U}_\Sigma$ is the uniform distribution on the ground truth mesh, and $\mathrm{dist}_{\Sigma^*}$ is Euclidean distance to the learned surface).

We do the same for our method by meshing our learned current: We compute the average value $s$ of $f$ over a boundary curve. Then, we extract a mesh of the level set $f^{-1}(s)$ using marching cubes. This level set is generically a closed surface containing our represented surface with boundary $\Sigma^*$ as a subset. We extract a mesh of $\Sigma^*$ by removing vertices $x$ for which the $|\nabla_x f_\theta(x) + \alpha_I^\#(x)| < \delta$, where $n_x$ is the surface normal at $x$. In practice, we use $\delta = 5 \times 10^{-3}$.

Our method consistently achieves better quality reconstructions than [Ven+21].

### 4.5.3 Latent space learning

We use our boundary-conditioned autodecoder (Section 4.4.4) to learn a disentangled representation that can interpolate in a high-dimensional learned latent space capturing shape identity while having explicit control over boundary geometry. We associate each mesh in our dataset with a random latent code (a trainable parameter), and, to disambiguate shape identity from boundary geometry, we perform random transformations. These transformations change the the boundary shape while preserving the latent code.

At each iteration, we perform random augmentations to the target meshes: we rotate each mesh by sampling a value in $[-10^\circ, 10^\circ]$ for each Euler angle, we rescale each boundary loop of the mesh by a random factor between 0.85 and 1.15 along each of its two principal directions, we propagate these transformations to the entire mesh using harmonic skinning weights, and we

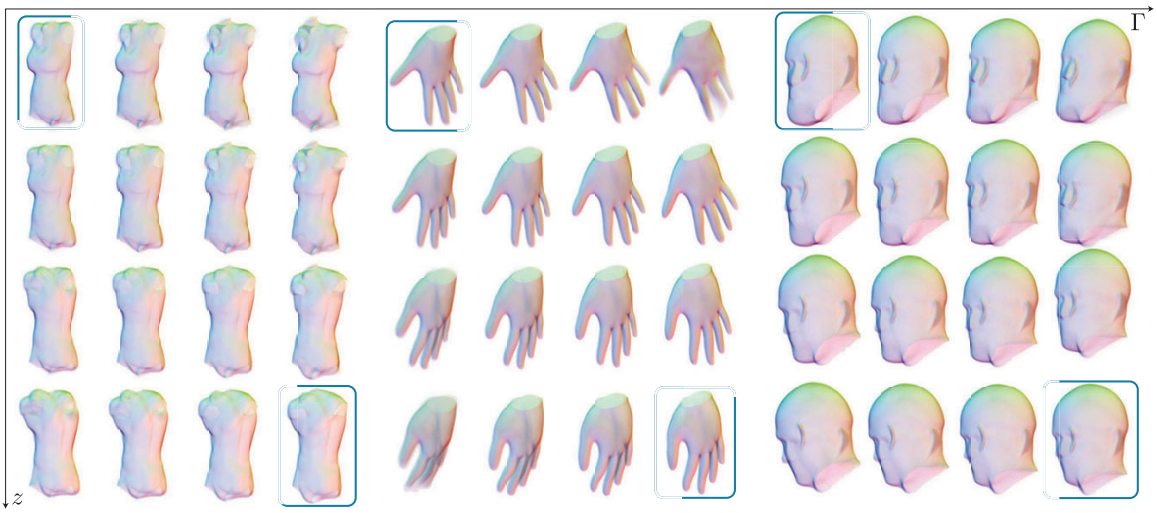



Figure 4-5: Interpolations of DeepCurrents in latent and boundary space. For each category, we pick two meshes from the training set (shown with a blue border) and interpolate linearly between the two boundaries (horizontal axis) as well as the two latent codes (vertical axis). The latent space interpolation yields a smooth transition between the two meshes while obeying the prescribed boundary interpolants.

shift the mesh by a random offset between −0.05 and 0.05 in each dimension.

We train a model for each shape category for 300,000 iterations (about 10 hours) with an initial learning rate of 0.0004, decayed by a factor of 0.5 every 60,000 iterations. At each step, we sample a random batch of 8 meshes and sample 4000 points from each mesh.

In Figure 4-5, we pick two models from each shape category and independently interpolate between their boundaries and latent identities. Our model disentangles high-level pose and style while respecting the prescribed geometry.

### 4.5.4 Ablation study

We validate some of our key design choices. In Fig. 4-6, we overfit five models to the same hand mesh. While our full model achieves a sharp reconstruction, removing boundary weighting from our current loss metric yields a fuzzier surface around the boundary. Changing the softplus activation functions in $h_\theta$ to

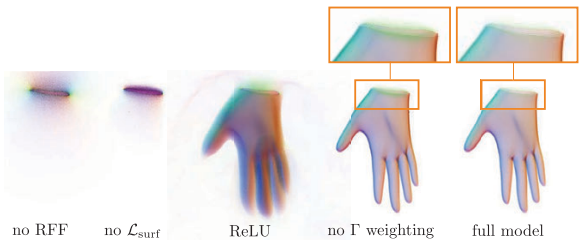



Figure 4-6: Ablation study. From left to right: reconstruction results on a hand without random Fourier features, without surface loss, without boundary weighting, and our full model.

ReLUs makes the entire learned surface significantly less sharp, which we conjecture is due to ReLU's zero second derivative when optimizing currents $\mathrm{d}f + \alpha$. Removing the surface loss term from our optimization fails to recover much of the target surface, supporting our claim that surface loss significantly helps convergence. Finally, foregoing the projection of the input points onto random Fourier features prevents the model from learning.

## 4.6 Discussion

By adopting tools from geometric measure theory, we have constructed a neural implicit representation for surfaces with boundary. Our SGD approach to mass norm minimization enables computing minimal surfaces with arbitrary resolution, in contrast to previous work that represents currents on a fixed-resolution grid. In addition, by constructing a background metric, we can engineer a mass minimization problem to encode an arbitrary surface. Combining this construction with the expressive power of neural representations, we can encode whole families of surfaces.

We see DeepCurrents as a key tool for building flexible neural surface representations. Stitching together DeepCurrents along their boundaries would produce a hybrid surface representation where the explicit boundary curves provide "handles" for user control. Such a representation would be applicable where the target surface is decomposed into parts. Unlike, say, a mesh decomposition, a DeepCurrent decomposition would not require the parts to have simple shapes or even to be simply connected.

Another direction for future work would be to investigate other loss functions and optimization problems that can be expressed in the language of currents. For example, the convex problems studied in [MC19] could be optimized using a neural representation and SGD. One could also compute minimal currents in spaces such as the rotation groups $\mathrm{SO}(d)$ or special Euclidean groups $\mathrm{SE}(d)$. Mass minimization in this context could provide a useful prior for reconstruction of shapes that come with an orientation or frame field, or it could exploit the Gauss map to encode a smoothness prior.

Another extension of our method would be to support periodic minimal surfaces, i.e., replacing the domain $[-1, 1]^3$ by the torus $\mathbb{T}^3$. This would require a modification of our explicit $\alpha$ and

the evaluation of $\mathrm{d}f$ at the boundary.

While our latent space model often produces high-quality interpolants, they are not explicitly regularized to encourage them to look like surfaces. This sometimes yields fuzzy results (see Fig. 4-5, top right among the hands). Future work could design loss terms to ensure interpolants remain minimal with respect to some metric—analogously to the eikonal regularization for SDFs in [AL20].

# 5

# Learning Sprites

The previous two chapters, have focused on leveraging deep learning to produce useful and intuitive geometric primitives without have strict supervision. Now, we apply a similar approach to textured images. We engineer an architecture that lets us discover semantically meaningful repeating elements, endowing a collection of raster images with a geometry that facilitates understanding and manipulation.

## 5.1 Introduction

Since the early days of machine learning, the accepted unit of image synthesis has been the *pixel*. But while the pixel grid is a natural representation for display hardware and convolutional generators, it does not easily permit high-level reasoning and editing.

In this chapter, we take inspiration from animation to consider an atomic unit that is richer and easier to edit than the pixel: the *sprite*. In sprite-based animation, a popular early technique for drawing cartoons and rendering video games, an artist draws a collection of patches—a *sprite sheet*—consisting of texture swatches, characters in various poses, static objects, and so on. Then, each frame is assembled by compositing a subset of the patches onto a canvas. By reusing the sprite sheet, authoring new content requires minimal effort and can even be automated procedurally.

Our goal is to invert this process, simultaneously tackling unsupervised instance segmenta-

This chapter includes material from the following publication: [Smi+21].

tion and dictionary learning. Given an image dataset, e.g., frames from a sprite-based video game, we train a model that jointly learns a 2D sprite dictionary, capturing recurring visual elements in an image collection, and explains each input frame as a combination of these potentially transparent sprites. Whereas standard CNN-based generators hide their feature representation in their intermediate layers, our model wears its representation "on its sleeve": by explicitly compositing sprites from its learnt dictionary onto a background canvas, rather than synthesizing pixels from hidden neural features, it provides a readily-interpretable visual representation.

Our contributions include the following:

- We describe a grid-based anchor system along with a learned dictionary of textured patches (with transparency) to extract a sprite-based image representation.
- We propose a method to learn the patch dictionary and the grid-based representation jointly, in a differentiable, end-to-end fashion.
- We compare to past work on learned disentangled graphics representations for video games.
- We show how our method offers promising avenues for further work towards identifying visual patterns in more complex data such as natural images and video.

## 5.2 Related work

Decomposing visual content into semantically meaningful parts for analysis, synthesis, and editing is a long-standing problem. We review the most closely related work.

*Layered decompositions.* Wang and Adelson [WA94] decompose videos into layers undergoing temporally-varying warps for compression. Similarly, Flexible Sprites [JF01] and Kannan, Jojic, and Frey [KJF05] represent videos with full-canvas semi-transparent layers to facilitate editing. Like Flexible Sprites, we adopt translation-only motion but restrict transformations to small neighborhoods around anchors, making inference tractable with many ($\geq 100$) sprites. Other methods decompose videos with moving subjects, such as humans, into independent layers, enabling matting [Lu+21] and retiming of individual actions [Lu+20]; unlike sprite-based techniques, motion and appearance are not disentangled. Sbai, Couprie, and Aubry [SCA20] use

a layered representation as inductive bias in a GAN with solid colored layers. Automatic decompositions into "soft layers" according to texture, color, or semantic features have been used in image editing [Aks+17; Aks+18]. Gandelsman, Shocher, and Irani [GSI19] use deep image priors [UVL18] to separate images into layer pairs. Huang and Murphy [HM16] introduce a recurrent architecture to output multiple layers sequentially. Reddy et al. [Red+20] discover patterns in images via differentiable compositing.

*Interpretable generators for neural synthesis.* Neural networks improve the fidelity and realism of generative models [Goo+14; Kar+20] but limit control and interpretability [Che+16; Här+20; Bau+19a; Bau+19b]. Several works explore interpretability using differentiable domain-specific functions. Hu et al. [Hu+18] and Li et al. [Li+18] constrain the generator to sets of parametric image operators. Mildenhall et al. [Mil+20b] use a ray-marching prior and rendering model to encode a radiance field for novel view synthesis. Neural textures [TZN19] replace RGB textures on 3D meshes with high-dimensional features. Rendering under new views enables view-consistent editing. Lin et al. [Lin+18] use spatial transformers in their generator to obtain geometric transformations. We synthesize frames by compositing 2D sprites undergoing rigid motions, enabling direct interpretation and control over appearance and motion.

*Object-centric representations.* Our learned sprites reveal, segment, and track object instances. Similarly, Slot Attention [Loc+20] extracts object-centric compositional video representations. However, our sprites are interpretable—motion and appearance are direct outputs—and our model scales to more objects per scene. SCALOR [Jia+19] handles up to 100 instances but does not produce a common dictionary or handle diverse sprites. While SPACE [Lin+20] decomposes images into object layers, it tends to embed sprites in the background, providing no control. Our method identifies a greater number of sprite patterns (see Section 5.4.1). Stampnet [Vis+19] discovers and localizes objects but focuses on simpler, synthetic datasets. MONet [Bur+19] decomposes images into multiple object regions using attention. Earlier attention mechanisms leverage pattern recurrence [Kos+18a; CP20] and motion cues [Esl+16] to identify individual objects. Recent works use parametric primitives as image building blocks [Smi+20; Li+20].

Applying our sprite decompositions to video games, we can learn about dynamics and gameplay, benefiting downstream agents [Jus+19; He+19] and aiding content-authoring for research

and game development, as in Procedural Content Generation [Sum+18]. GameGAN [Kim+20] synthesizes new frames from controller input. They split rendering into static and dynamic components but render full frames, without factorization into parts. Their generator is difficult to interpret: appearance and dynamics are entangled within its parameters.

*Compression.* Appearance consistency and motion compensation are central to video compression [BC15; Lu+19; Lom+19]. We model videos as compositions of sprites, factoring redundancy in the input. This draws inspiration from works like DjVu [Haf+99] and Digipaper [HFR99], which compress scanned documents by separating them into a background layer and foreground text. Image epitomes [JFK03] summarize and compress image shape and appearance into a miniature texture. Our sprite dictionary fills a similar role, providing superior editing control.

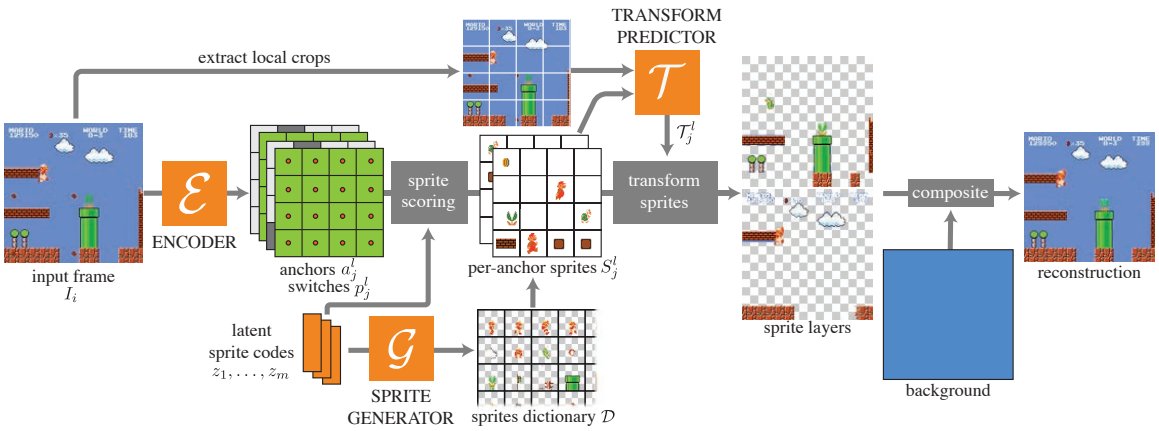



Figure 5-1: Overview. We jointly learn a sprite dictionary, represented as a set of latent codes decoded by a generator, as well as an encoder network that embeds a frame into a grid of latent codes, or *anchors*. Comparing anchor embeddings to dictionary codes lets us assign a sprite to each grid cell. Our encoder also outputs a binary switch per anchor to turn sprites on and off. After compositing, we obtain a reconstruction of the input. Our self-supervised training optimizes a reconstruction loss.

## 5.3 Method

We start with an input sequence of $n$ RGB frames $\{I_1, \ldots, I_n\}$ with resolution $w \times h$. Our goal is to decompose each frame $I_i \in \mathbb{R}^{3 \times w \times h}$ into a set of possibly overlapping sprites, organized into $\ell$ depth layers, selected from a dictionary. The dictionary is a collection of trainable latent codes $\{z_1, \ldots, z_m\}$ that are decoded into RGBA sprites using a neural network generator (Section 5.3.1).

Our training pipeline is illustrated in Fig. 5-1. We first process each input frame with a convolutional encoder to produce $\ell$ grids of feature vectors, one grid per depth layer (Section 5.3.2). The grids are lower resolution than the input frame, with a downsampling factor proportional to the sprite size. We call the center of each grid cell an *anchor*. We compare each anchor's feature vector against the dictionary's latent codes, using a softmax scoring function, to select the best matching sprite per anchor (Section 5.3.3). Using our sprite generator, we decode each anchor's matching sprite. This gives us a grid of sprites for each of the $\ell$ layers. To factorize image patterns that may not align with our anchor grid, we allow sprites to move in a small neighborhood around anchors (Section 5.3.4). We composite the layers from back to front onto the output canvas to obtain our final reconstruction (Section 5.3.5). Optionally, the background is modeled as a special learnable sprite that covers the entire canvas.

We train the dictionary latent codes, frame encoder, and sprite generator jointly on all frames, comparing our reconstruction to the input (Section 5.3.6). This self-supervised procedure yields a representation that is sparse, compact, interpretable, and well-suited for downstream editing and learning applications.

### 5.3.1 Dictionary and sprite generator

The central component of our representation is a global *dictionary* of $m$ textured patches or sprites $\mathcal{D} = \{P_1, \dots, P_m\}$, where each $P_i \in \mathbb{R}^{4 \times k \times k}$ is an RGBA patch. Our sprites have an alpha channel, which allows them to be partially transparent, with possibly irregular (i.e., non-square) boundaries. This is useful for representing animations with multiple depth layers and also allows to learn sprites smaller than their maximal resolution, if necessary, by setting alpha to zero around the boundary. The dictionary is shared among all frames; we reconstruct frames using only sprites from the dictionary.

Instead of optimizing for RGBA pixel values directly, we represent the dictionary as a set of trainable latent codes $\{z_1, \dots, z_m\}$, with $z_i \in \mathbb{R}^d$. We decode these codes into RGBA sprites using a fully-connected sprite generator $P_i = \mathcal{G}(z_i)$. This latent representation allows us to define a similarity metric over the latent space, which we use to pair anchors with dictionary sprites to best reconstruct the input frame (Section 5.3.3). At test time, we can edit the RGBA sprites directly. Unless otherwise specified, we set latent dimension to $d = 128$ and patch size to $k = 32$.

We randomly initialize the latent codes from the standard normal distribution. Our sprite generator first applies zero-mean unit-variance normalization—Layer Normalization [BKH16], without an affine transformation—to each latent code $z_i$ individually, followed by one fully-connected hidden layer with $8d$ features, Group Normalization [WH18], and ReLU activation. We obtain the final sprite using a fully-connected layer with sigmoid activation to keep RGBA values in $[0, 1]$. Latent code normalization is crucial to stabilize training and keep the latent space in a compact subspace as the optimization progresses. See Section 5.4.3 for an ablation study of this and other components.

### 5.3.2 Layered frame decomposition using sprite anchors

We seek a decomposition that best explains each input frame using dictionary sprites. We exploit translation invariance and locality in our representation; our sprites are "attached" to a regular grid of reference points, or *anchors*, inspired by [Red+16; Gir15]. Each anchor has at most one sprite; we call it *inactive* if it has none.

We give the sprites freedom of motion around their anchors to factorize structures that may not be aligned with the anchor grid. This local—Eulerian—viewpoint makes inference tractable and avoids the pitfalls of tracking the global motion of *all* the sprites across the canvas (a Lagrangian viewpoint). To enable multiple layers with sprite occlusions, we output $\ell > 1$ anchor grids for each frame ($\ell = 2$ in our experiments). Fig. 5-2 illustrates our layered anchor grids and local sprite transformations.

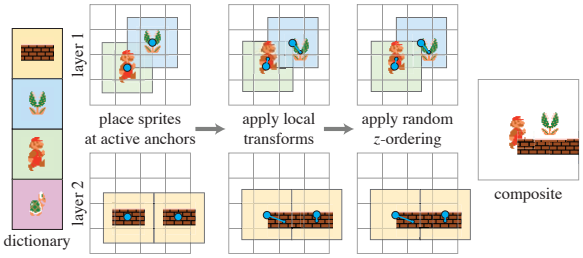



Figure 5-2: Layered sprite decomposition with local anchors. We assign at most one sprite per anchor and predict a local transformation of each placed sprite around its anchor. To allow for occlusions between sprites, we use multiple sprite layers, which we compose back to front to obtain the final image.

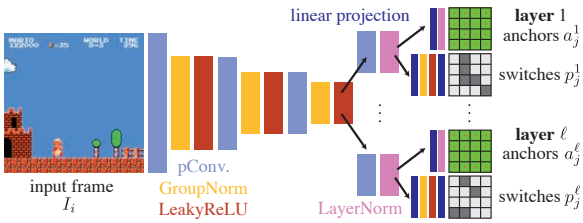



Figure 5-3: Encoder architecture.

We use a convolutional encoder $\mathscr{E}$ to map the $w \times h$ RGB frame $I_i$ to $\ell$ grids of anchors, with resolution $\frac{2w}{k} \times \frac{2h}{k}$. Each anchor $j$ in layer $l$ is represented by a feature vector $a_j^l \in \mathbb{R}^d$ characterizing local image appearance around the an-

chor and an active/inactive switch probability $p_j^l \in [0, 1]$. Our frame encoder contains $\log_2(k) - 1$ downsampling blocks, which use partial convolutions [Liu+18] with kernel size 3 and stride 2 (for downsampling), Group Normalization, and Leaky ReLU. It produces a tensor of intermediate features for each layer, which are normalized with LayerNorm. From these, we obtain the anchor switches with an MLP with one hidden layer of size $d$ followed by Group Normalization and Leaky ReLU. We get anchor features using a linear projection followed by LayerNorm. The encoder architecture is illustrated in Fig. 5-3.

### 5.3.3 Per-anchor sprite selection

Once we have the layered anchor grids for the input frame, we need to assign sprites to the active anchors. We do this by scoring every dictionary element $i$ against each anchor $j$ at layer $l$, using a softmax over dot products between dictionary codes and anchor features:

$$s_{ij}^l = \frac{\exp\left(a_j^l \cdot z_i / \sqrt{d}\right)}{\sum_{k=1}^m \exp\left(a_j^l \cdot z_k / \sqrt{d}\right)}. \tag{5.1}$$

Recall that both the anchor features and dictionary latent codes are individually normalized using Layer Normalization. Restricting both latent spaces to a compact subspace helps stabilize the optimization and avoid getting stuck in local optima. During training, each anchor's sprite is a weighted combination of the dictionary elements, masked by the anchor's active probability:

$$S_j^l = p_j^l \sum_{i=1}^m s_{ij}^l P_i. \tag{5.2}$$

This soft selection allows gradients to propagate to both dictionary and anchor features during training. Except for natural image and video datasets, at test time, we use hard selections, i.e., each anchor, we pick the sprite ($S_j^l := P_i$) with highest score $s_{ij}^l$ and binarize the switches $p_j^l \in \{0, 1\}$.

### 5.3.4 Local sprite transformations

In real animations, sprites rarely perfectly align with our regular anchor grid, so, to avoid learning several copies of the same sprites (e.g., all sub-grid translations of a given image pattern), we allow sprites to move around their anchors. In our implementation, we only allow 2D translations of

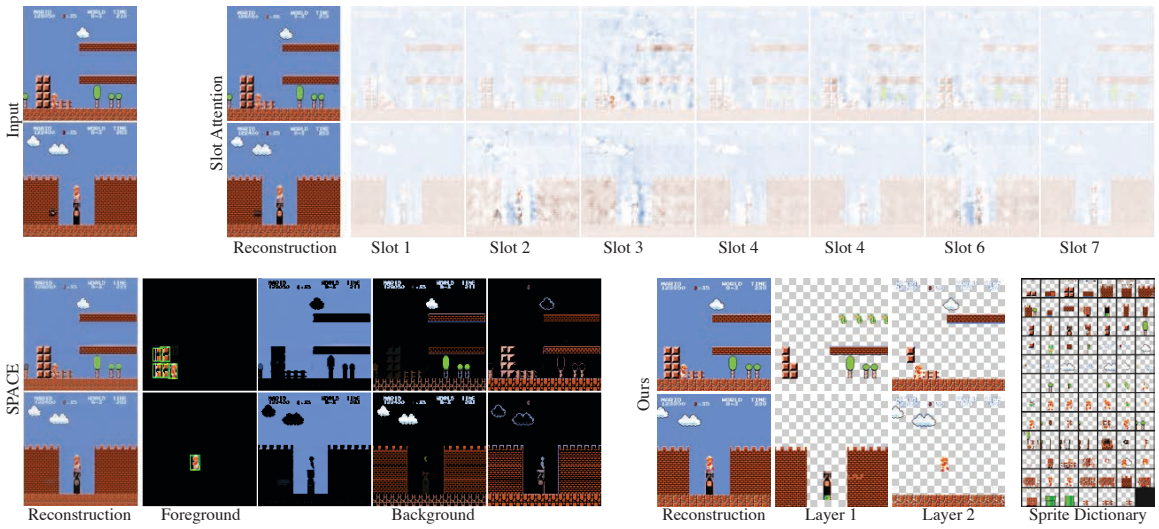



Figure 5-4: Comparison to SPACE [Lin+20] and Slot Attention [Loc+20]. While all methods obtain good reconstructions, SPACE only recognizes a few sprites, and Slot Attention does not yield a meaningful decomposition. We decompose the entire foreground and learn a dictionary.

up to $1/2$ the sprite size on each side of the anchor, i.e., $\mathcal{T}_j^l = (x_j^l, y_j^l) \in [-k/2, k/2]^2$.

We use a convolutional network to predict the translation offsets from the anchor's sprite and a crop of the input frame centered around the anchor, with identical spatial dimensions. This network follows the architecture of $\mathscr{E}$ followed by an MLP with a single hidden layer of size $d$, Group Normalization, and Leaky ReLU. Specifically, we concatenate the image crop and the anchor's sprite $S_j^l$ along the channel dimension and pass this tensor through this network to obtain the $x_j^l$ and $y_j^l$ offsets. An output layer projects to two dimensions (horziontal and vertical shift) and applies tanh to restrict the range. We apply the shifts using a spatial transformer [Jad+15].

### 5.3.5 Compositing and reconstruction

Each anchor in our layered representation is now equipped with a sprite $S_j^l$ and a transformation $\mathcal{T}_j^l$. For each layer $l$, we transform the sprites in their anchor's local coordinate system and render them onto the layer's canvas, initialized as fully transparent. Because of the local transformation, neighboring sprites within a layer may overlap. When this happens, we randomly choose an ordering, as in Fig. 5-2. This random permutation encourages our model to either avoid overlapping sprites within the same layer or make the sprite colors agree in the overlap region, since these are the only two options that yield the same rendering regardless of the random $z$-ordering. Note

that sprites on *distinct layers* are not shuffled. The shuffling prevents the network from abusing the compositing to cover patches with others from the same layer.

We optionally learn a background texture to capture elements that cannot be explained using sprites. This can be thought of as a special patch of resolution greater than that of a single frame. For each frame, we learn a (discrete) position offset in the background from which to crop. We represent these offsets as discrete pixel shifts using a softmax classification (independently for each spatial dimension). We found this encoding better behaved than using a continuous offset with a spatial transformer—the discrete encoding allows the gradient signal to propagate to all shifts rather than the weak local gradient from bilinear interpolation (see Section 5.4.3 for an ablation). We combine the background and sprite layers via standard alpha compositing [PD84]. Fig. 5-7 shows a learned background.

In some experiments, we use a simpler background model: a fixed solid color, determined by analyzing the data before training. In this variant, we sample 100 frames, cluster the pixel values into 5 clusters using $k$-means, and choose the largest cluster center as the background color.

### 5.3.6 Training procedure

Our pipeline is fully differentiable. We train the latent codes dictionary, sprite generator, frame encoder, transformation predictor, and background layer jointly, minimizing $L_2$ distance between our reconstructions and ground truth frames. We also employ two regularizers: a beta distribution prior on switches and dictionary element scores favors values close to 0 or 1, and an $L_1$ loss on switches favors a sparser solution. Our final loss function for a single input is:

$$\begin{aligned}\mathscr{L}(\cdot) =& \tfrac{1}{wh}\|O - I\|_2^2 \\ &+ \tfrac{k^2}{4\ell wh}\sum_{l=1}^{\ell}\sum_{j=1}^{\frac{2w}{k}\times\frac{2h}{k}}\left[\lambda_{\text{Beta}}\left(\tfrac{1}{m}\sum_{i=1}^{m}\text{Beta}(2,2)(s_{ij}^l) + \text{Beta}(2,2)(p_j^l)\right) + \lambda_{\text{sparse}}|p_j^l|\right],\end{aligned} \tag{5.3}$$

where $O$ is the result of compositing the background and sprite layers; we optimize $\{s_{ij}^l\}$, $\{p_j^l\}$, and $O$. We set $\lambda_{\text{sparse}} = 0.005$ and train for 200,000 steps (~ 20 hours) with $\lambda_{\text{Beta}} = 0.002$ and finetune for 10,000 steps with $\lambda_{\text{Beta}} = 0.1$. For natural images and video, we set $\lambda_{\text{Beta}} = 0$. We

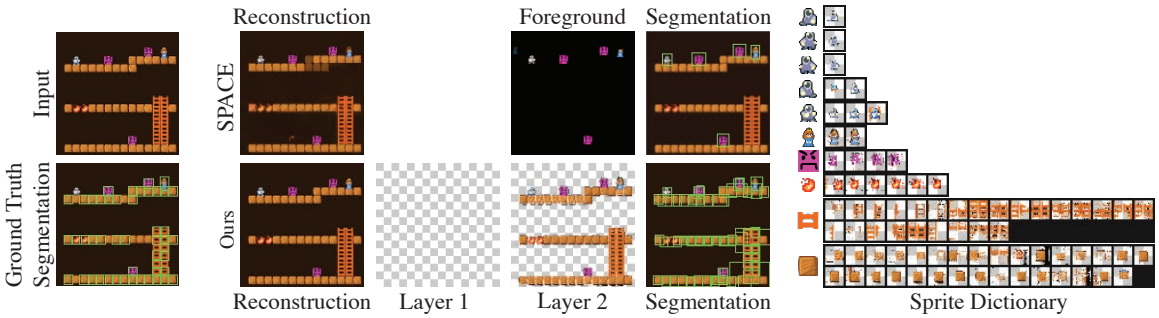



Figure 5-5: Qualitative comparison to SPACE [Lin+20] on the synthetic game dataset. We show ground truth sprite segmentations as well as those obtained from the learned SPACE foreground and from our learned sprites. While SPACE only learns several of the sprites, we reconstruct the entire foreground using our dictionary.

use the AdamW [LH19] optimizer on a GeForce GTX 1080 GPU, with batch size 4 and learning rate 0.0001, except for the background module (learning rate 0.001 when used).

## 5.4 Experimental results

We evaluate our self-supervised decomposition on several real (non-synthetic) datasets, compare to related work, and conduct an ablation study. In figures, we use a checkerboard to show transparency. Dictionary order is determined by sorting along a 1-dimensional $t$-SNE embedding of the sprite latent codes. We find this ordering tends to group semantically similar sprites, making the dictionary easier to interpret and manipulate. While our models are trained with a dictionary of 150 patches, not all patches end up being used; we only show the used patches.

### 5.4.1 Comparisons

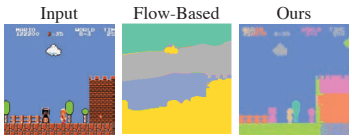
While to our knowledge no prior works target differentiable sprite-based reconstruction, we compare to two state-of-the-art methods that obtain similarly disentangled representations.

In Fig. 5-4, we compare to SPACE [Lin+20] and Slot Attention [Loc+20]. The former decomposes a scene into a foreground layer consisting of several objects as well as a background, segmented into three layers. The latter deconstructs a scene into discrete "slots." We train both methods to convergence using their default parameters. While both reconstruct the input frames faithfully, SPACE only recognizes a few sprites in its foreground layer, and Slot Attention does

not provide a semantically meaningful decomposition. In contrast, not only does our method model the entire scene using learned sprites, but also it factors out the sprites to form a consistent, sparse dictionary shared for the entire sequence.

Additionally, we evaluate on a synthetically-generated sprite-based game from [Dub+18], which is made of sprites on a solid background. We compare quantitatively to SPACE in Table 5.1 and show qualitative results in Fig. 5-5. Since we have a ground truth segmentation of each scene into sprites, we compute a matching between learned dictionary patches and sprites by associating each patch with the sprite that it most frequently overlaps. We visualize dictionary patches next to their respective sprites. We also use this labeling to compute segmentation metrics. In particular, we report mean IoU in the multiclass case (where each sprite is a distinct class) as well is in the binary case (foreground/background). Because SPACE does not learn a common dictionary, we are unable to obtain a labeling for its foreground elements and, thus, cannot evaluate its multiclass metric. For the binary metric, we obtain a significantly higher value, since SPACE defers many sprites to the background, whereas our method learns the sprites as dictionary elements.

To show that our model learns more than simple motion features, we also compare to two conventional (non-learning) baselines. In Fig. 5-6a, we compare a segmentation of a frame obtained by clustering optical flow directions using $k$-means (inspired by Liu et al. [Liu+05]) to one generated using our learned decomposition. The flow-based approach is unable to capture many of the details in the frame. In Fig. 5-6b, we show the normalized dictionary obtained using an online dictionary learning method [Mai+10]. Because this method does not have the inductive biases of our model, the resulting dictionary is not easily interpretable or editable.



(a) Segmentation

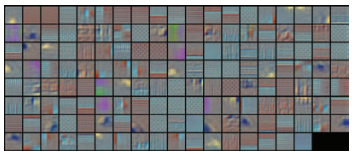

(b) Online Dictionary Learning

Figure 5-6: Comparison to conventional baselines.

Table 5.1: Comparison to SPACE [Lin+20]. We report PSNR to evaluate reconstruction quality as well as mean IoU for multiclass and binary foreground/background segmentation problems. Our method recognizes significantly more sprites than SPACE, resulting in higher mean IoU.

| Method | Reconstruction PSNR | Mean IoU (multiclass) | Mean IoU (binary) |
|---|---|---|---|
| SPACE [Lin+20] | 31.9 | - | 0.0361 |
| Ours | **38.54** | **0.6497** | **0.7352** |

### 5.4.2 Sprite-based game deconstruction

We train on Fighting Hero (one level, 5,330 frames), Nintendo Super Mario Bros. (one level, 2,220 frames), and ATARI Space Invaders (5,000 frames). We use patch size $k = 32$ for Mario and Fighting Hero and $k = 16$ for Space Invaders. For Fighting Hero, we learn a background, as described in Section 5.3.5.

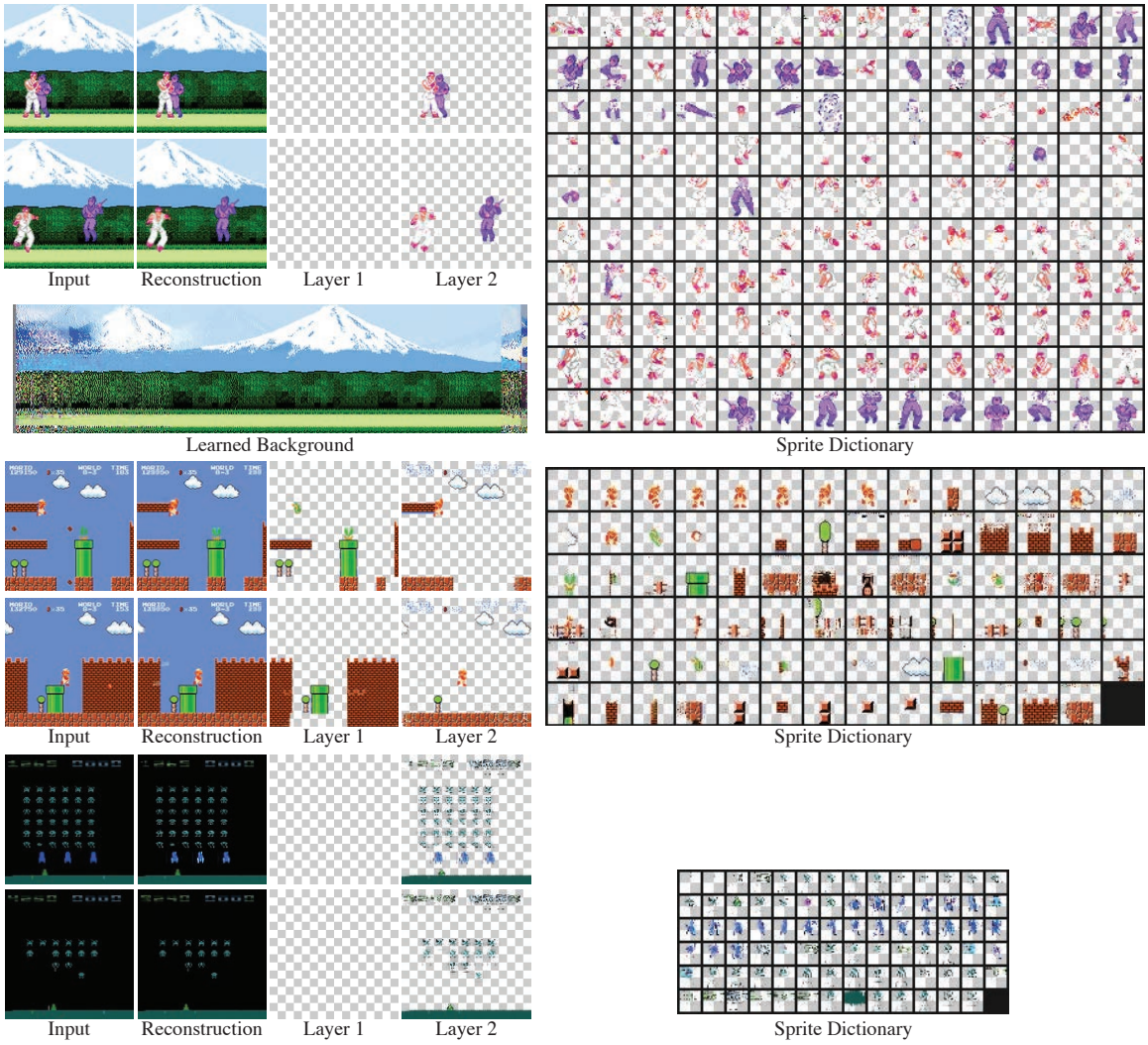



Figure 5-7: Sprite-based game decompositions. Our self-supervised technique recovers a compact dictionary of semantically meaningful sprites representing characters (or their body parts) and props. The first example shows our learned background texture; the others use a solid color as background.

Table 5.2: Ablation study on Mario data across five random seeds.

| Model | PSNR |
|---|---|
| Smaller patches | 28.85 ± 0.95 |
| Full | 28.04 ± 0.72 |
| No LayerNorm | 26.05 ± 0.45 |
| Smaller dictionary | 23.80 ± 1.38 |
| Larger patches | 23.63 ± 1.05 |
| Straight-through switches | 22.15 ± 0.25 |

The sprites, background, and example frame reconstructions are shown in Fig. 5-7. Our model successfully disentangles foreground from background and recovers a reasonable sprite sheet for each game. Having reverse-engineered the games, we can use the decomposition for applications like editing. In Fig. 5-8, we demonstrate a GUI that allows the user to move sprites around the screen.

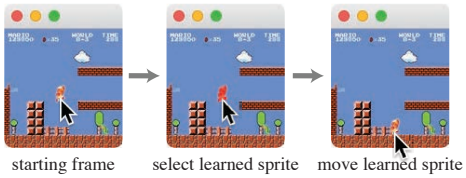


Figure 5-8: Editing GUI.

### 5.4.3 Ablation study

We show an ablation study on the Mario data. We train our full model, one with smaller 16×16 patches, another with larger 64×64 patches, a model with a smaller dictionary (25 elements), a model without LayerNorm, and one where we use a straight-through estimator [JGP17] to learn discrete switches $p_j^l$ in lieu of Beta regularization. We train each model with five random seeds and report the reconstruction PSNR means and standard deviations in Table 5.2. This experiment verifies the importance of LayerNorm in our architecture and shows that the straight-through trick is ineffective in our setting. Though the smaller patches model achieves slightly higher mean PSNR than our full model, more of the sprites are split across dictionary patches (Fig. 5-9), illustrating how the patch size choice sets an inductive bias for our decomposition.


Figure 5-9: Mario dictionary with small patches.

We also justify our choice for learning background shifts via classification (Section 5.3.5) rather than regression, i.e., using spatial transformers. Figure 5-10 shows the background


Figure 5-10: Background learned with spatial transformer.

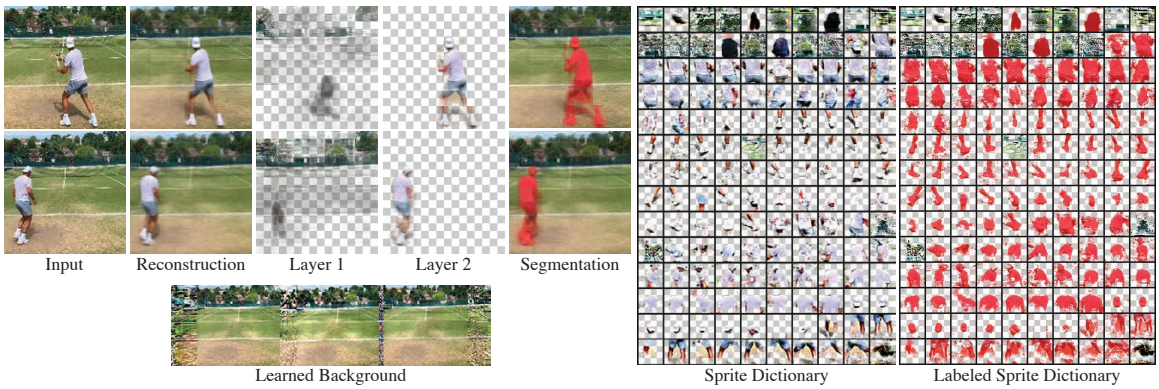



Figure 5-12: Segmentation of natural videos. Despite its simplistic motion and appearance model, our approach can be applied to real-world videos. By selecting the few sprites corresponding to the tennis player, we can quickly obtain a segmentation of the full video sequence.

learned using a spatial transformer. In contrast to our full model (Fig. 5-7), the original background is not discovered, and most of the canvas is unused. We suspect that this is due to lack of gradient signal from background pixels that do not get rendered at each training step.

### 5.4.4 Future directions and limitations

While our method is designed with sprite-based animation in mind, it can generalize to natural images and videos. An exciting direction for future work is to incorporate more expressive transformations so as to discover recurring content in generic videos. Here, we obtain preliminary results using our approach and achieve interesting decompositions even without modifications to our sprite-based model.

In Fig. 5-12, we show results on a tennis video (4,000 frames). The model learns parts of the player's body (head, limbs, shirt, etc.) as sprites and captures most of the tennis court in the learned background. By simply selecting the player sprites in the dictionary, we segment the entire video clip.

Our model can also discover recurring patterns in a single natural image. We train on random crops of a $768 \times 512$ pho-

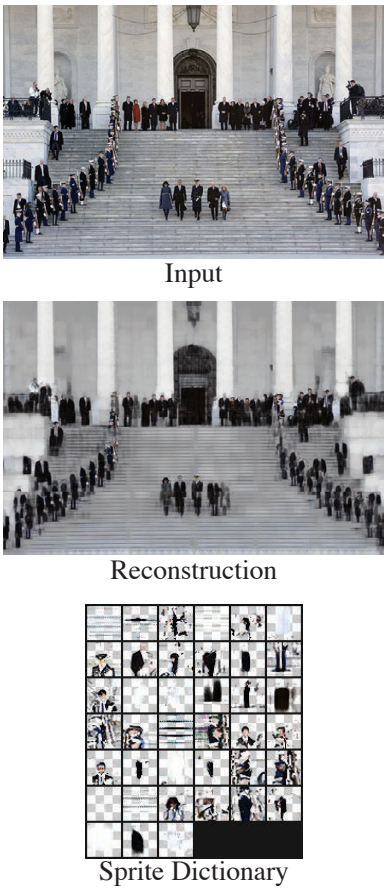
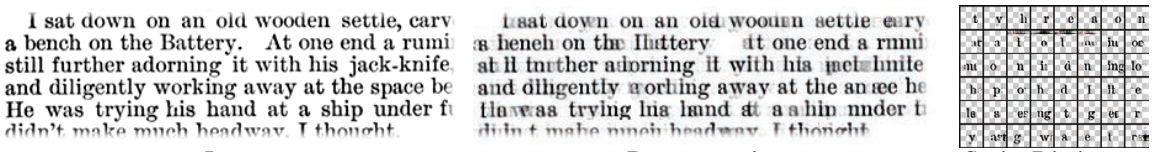



Figure 5-11: Photograph factorized into a compact dictionary.



Input Reconstruction Sprite Dictionary

Figure 5-13: Reconstruction of a scanned text excerpt, illustrating some limitations of our method. Because letters are densely packed within the text and lack motion cues, our model learns sprites comprising more than a single glyph and suffers from high reconstruction error.

tograph from the 2013 US Presidential Inauguration,[1] which contains many repeating elements such as stairs, columns, and people. With a dictionary of 39 32×32 sprites (39,936 pixels), we recover much of the detail of the original 393,216 pixels.

We also apply our approach to automatic font discovery. We train on random 128×128 crops of six scanned pages of *Moby Dick*, each of approximately 500×800 resolution. Figure 5-13 shows an input text excerpt, our reconstruction, and the learned dictionary.

This dataset differs significantly from our other testing datasets. Each input frame consists of many densely packed sprites (~100 glyphs in each 128×128 crop), and many individual glyphs consist of smaller repeating elements. We hypothesize that because of these issues, combined with a lack of motion cues between frames, we do not achieve a perfect reconstruction, learning certain sprites with multiple glyphs and others with just partial glyphs. Incorporating priors tailored to regularly structured and dense data like text is a direction for future research.

## 5.5 Discussion

We present a self-supervised method to jointly learn a patch dictionary and a frame encoder from a video, where the encoder explains frames as compositions of dictionary elements, anchored on a regular grid. By generating layers of alpha-masked sprites and predicting per-sprite local transformation, we recover fine-scale motion and achieve high-quality reconstructions with semantically meaningful, well-separated sprites. Applied to content with significant recurrence, our approach recovers structurally significant patterns.

Understanding recurring patterns and their relationships is central to machine learning. Learning to act intelligently in video games or in the physical world requires breaking experiences down

[1] AP Photo/Cliff Owen

into elements between which knowledge can be transferred effectively. Our sprite-based decomposition provides an intuitive basis for this purpose.

In this chapter, we focused on a simplified video domain. In the future, we would like to expand the range of deformations applied to the learned dictionary elements, such as appearance or shape changes. Combining this machinery with some of the techniques we developed for learning parametric geometry in Chapter 3, could further motivate unsupervised approaches to the discovery and analysis of textured objects. For example, we could consider analyzing trajectories and dynamics of sprites by learning spline curves in time, or we could learn textured 3D models, relying on a patch-based parameterization. Our work opens significant avenues for future research to explore recurrences and object relationships in more complex domains.

# 6

# Conclusion

The methods and experiments presented in this thesis demonstrate how making a geometry a first-class citizen in applications of deep learning to shape data can result in more intuitive and useful algorithms. Rather than trying to shoehorn shape representations into a format that works with standard architectures and treating the deep network as a black box with fixed input and output, we instead consider the specific properties unique to the data that we have access to or that we require for downstream tasks. For example, we extract features based on the underlying surface approximated by a triangle mesh instead of naively relying on the combinatorial structure, and we consider the parameterization used to define CAD primitives when designing corresponding loss functions.

Fortunately, we often do not need to reinvent the wheel to design sensible learning methodologies. There is a rich literature in mathematics and computer science of machinery for processing, analyzing, and producing the types of geometry common in many applications. By familiarizing ourselves with these classical approaches and theories, we can design the proper inductive biases in our machine learning models.

This paradigm for designing learning algorithms inspires many exciting directions for future work. We can consider other common tasks in geometry processing, like geometric flows or parameterizations, that could benefit from the data-rich setting of machine learning. We could also look at creating sensible learning approaches that operate on other non-uniform shape modalities. For instance, we could learn from collection of shape data in the form of volumetric tetrahedral or hexahedral meshes, often used in simulation, or from digital sculpting models, pop-

ular with modern virtual or augmented reality input modalities. Beyond considering methods tailored specifically to individual applications or representations, we could also look towards unified multi-modal geometric machine learning models. Recent methods have shown impressive results by learning simultaneously from various data formats, like images and text [Rad+21] or various types of sensor data [Che+22]. Perhaps by learning on different shape modalities at the same time, using a system that is aware of the specific mathematical features of each individual representation, we can learn a whole that is greater than each of its parts.

# Bibliography


[Aks+17] Yağiz Aksoy, Tunç Ozan Aydin, Aljoša Smolić, and Marc Pollefeys. "Unmixing-Based Soft Color Segmentation for Image Manipulation". In: *ACM Transactions on Graphics* 36.2 (2017), pp. 1–19 (cit. on p. 98).

[Aks+18] Yağiz Aksoy, Tae-Hyun Oh, Sylvain Paris, Marc Pollefeys, and Wojciech Matusik. "Semantic Soft Segmentation". In: *ACM Transactions on Graphics* 37.4 (2018), pp. 1–13 (cit. on p. 98).

[AL20] Matan Atzmon and Yaron Lipman. "SAL: Sign Agnostic Learning of Shapes From Raw Data". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (cit. on pp. 12, 80, 95).

[ASC11] Mathieu Aubry, Ulrich Schlickewei, and Daniel Cremers. "The Wave Kernel Signature: A Quantum Mechanical Approach to Shape Analysis". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2011, pp. 1626–1633 (cit. on pp. 18, 35).

[AT16] James Atwood and Don Towsley. "Diffusion-Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. 2016, pp. 1993–2001 (cit. on p. 20).

[Au+11] Oscar Kin-Chung Au, Youyi Zheng, Menglin Chen, Pengfei Xu, and Chiew-Lan Tai. "Mesh Segmentation with Concavity-Aware Fields". In: *IEEE Transactions on Visualization and Computer Graphics* 18.7 (2011), pp. 1125–1134 (cit. on p. 18).

[Aza+18] Samaneh Azadi, Matthew Fisher, Vladimir Kim, Zhaowen Wang, Eli Shechtman, and Trevor Darrell. "Multi-Content GAN for Few-Shot Font Style Transfer". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 11. 2018, p. 13 (cit. on pp. 46, 65, 66).

[Bal+18] Elena Balashova, Amit Bermano, Vladimir G. Kim, Stephen DiVerdi, Aaron Hertzmann, and Thomas Funkhouser. "Learning A Stroke-Based Representation for Fonts". In: *Computer Graphics Forum*. 2018 (cit. on p. 46).

[Bau+19a] David Bau, Jun-Yan Zhu, Hendrik Strobelt, Bolei Zhou, Joshua B Tenenbaum, William T Freeman, and Antonio Torralba. "GAN Dissection: Visualizing and Understanding Generative Adversarial Networks". In: *International Conference on Learning Representations (ICLR)*. 2019 (cit. on p. 98).

[Bau+19b] David Bau, Jun-Yan Zhu, Jonas Wulff, William Peebles, Hendrik Strobelt, Bolei Zhou, and Antonio Torralba. "Seeing What a GAN Cannot Generate". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 4502–4511 (cit. on p. 98).

[BB10] Michael Bronstein and Alexander Bronstein. "Shape Recognition with Spectral Distances". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.5 (2010), pp. 1065–1071 (cit. on p. 18).

[BC15] Srinivas Bachu and K Manjunatha Chari. "A review on motion estimation in video compression". In: *International Conference on Signal Processing and Communication Engineering Systems*. IEEE, 2015, pp. 250–256 (cit. on p. 99).

[Ben+10] Mirela Ben-Chen, Adrian Butscher, Justin Solomon, and Leonidas Guibas. "On Discrete Killing Vector Fields and Patterns on Surfaces". In: *Computer Graphics Forum*. Vol. 29. 2010, pp. 1701–1711 (cit. on p. 19).

[Ber+99] Fausto Bernardini, Chandrajit L Bajaj, Jindong Chen, and Daniel R Schikore. "Automatic Reconstruction of 3D CAD Models from Digital Scans". In: *International Journal of Computational Geometry & Applications* 9.4 (1999), pp. 327–369 (cit. on p. 70).

[Bes+15] Mikhail Bessmeltsev, Will Chang, Nicholas Vining, Alla Sheffer, and Karan Singh. "Modeling Character Canvases from Cartoon Drawings". In: *ACM Transactions on Graphics* 34.5 (2015), 162:1–162:16 (cit. on p. 47).

[BK10] Michael Bronstein and Iasonas Kokkinos. "Scale-Invariant Heat Kernel Signatures for Non-Rigid Shape Recognition". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2010, pp. 1704–1711 (cit. on p. 19).

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. "Layer Normalization". In: *arXiv:1607.06450* (2016) (cit. on p. 101).

[Bli82] James F Blinn. "A Generalization of Algebraic Surface Drawing". In: *ACM Transactions on Graphics* 1.3 (1982), pp. 235–256 (cit. on p. 76).

[BLM18] Blanche Buet, Gian Paolo Leonardi, and Simon Masnou. "Discretization and Approximation of Surfaces Using Varifolds". In: *Geometric Flows* 3.1 (2018), pp. 28–56 (cit. on p. 80).

[BM19] Haïm Brezis and Petru Mironescu. "The Plateau problem from the perspective of optimal transport". In: *Comptes Rendus Mathematique* 357.7 (2019), pp. 597–612 (cit. on pp. 80, 84).

[Bog+14] Federica Bogo, Javier Romero, Matthew Loper, and Michael J Black. "FAUST: Dataset and evaluation for 3D mesh registration". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014, pp. 3794–3801 (cit. on p. 91).

[Bor84] Gunilla Borgefors. "Distance transformations in arbitrary dimensions". In: *Computer Vision, Graphics, and Image Processing* 27.3 (1984), pp. 321–345 (cit. on p. 48).

[Bos+15a] Davide Boscaini, Davide Eynard, Drosos Kourounis, and Michael Bronstein. "Shape-from-Operator: Recovering Shapes from Intrinsic Operators". In: *Computer Graphics Forum*. Vol. 34. 2015, pp. 265–274 (cit. on p. 19).

[Bos+15b] Davide Boscaini, Jonathan Masci, Simone Melzi, Michael Bronstein, Umberto Castellani, and Pierre Vandergheynst. "Learning class-specific descriptors for deformable shapes using localized spectral convolutional networks". In: *Computer Graphics Forum*. Vol. 34. 2015, pp. 13–23 (cit. on p. 21).

[Bos+16] Davide Boscaini, Jonathan Masci, Emanuele Rodolà, and Michael Bronstein. "Learning shape correspondence with anisotropic convolutional neural networks". In: *Advances in Neural Information Processing Systems*. 2016 (cit. on p. 20).

[Bra92] Kenneth A Brakke. "The Surface Evolver". In: *Experimental Mathematics* 1.2 (1992), pp. 141–165 (cit. on p. 79).

[Bro+11] Alexander Bronstein, Michael Bronstein, Leonidas Guibas, and Maks Ovsjanikov. "Shape Google: Geometric words and expressions for invariant shape retrieval". In: *ACM Transactions on Graphics* 30.1 (2011), pp. 1–20 (cit. on p. 18).

[Bro+17] Michael Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. "Geometric Deep Learning: Going beyond Euclidean data". In: *IEEE Signal Processing Magazine* 34.4 (2017), pp. 18–42 (cit. on p. 18).

[Bru+14] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. "Spectral Networks and Locally Connected Networks on Graphs". In: *International Conference on Learning Representations (ICLR)*. 2014 (cit. on pp. 20, 22).

[BS19] Mikhail Bessmeltsev and Justin Solomon. "Vectorization of Line Drawings via PolyVector Fields". In: *ACM Transactions on Graphics* 38.1 (2019) (cit. on pp. 68, 76).

[Bur+19] Christopher P Burgess, Loic Matthey, Nicholas Watters, Rishabh Kabra, Irina Higgins, Matt Botvinick, and Alexander Lerchner. "MONet: Unsupervised Scene Decomposition and Representation". In: *arXiv:1901.11390* (2019) (cit. on p. 98).

[BWL20] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. "YOLOv4: Optimal speed and accuracy of object detection". In: *arXiv:2004.10934* (2020) (cit. on p. 10).

[Cha+15] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. *ShapeNet: An Information-Rich 3D Model Repository*. Tech. rep. arXiv:1512.03012. Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015 (cit. on pp. 44, 66, 68).

[Che+05] Joseph Jacob Cherlin, Faramarz Samavati, Mario Costa Sousa, and Joaquim A. Jorge. "Sketch-based Modeling with Few Strokes". In: *Proceedings of the 21st Spring Conference on Computer Graphics*. 2005, pp. 137–145 (cit. on p. 47).

[Che+13] Tao Chen, Zhe Zhu, Ariel Shamir, Shi-Min Hu, and Daniel Cohen-Or. "3-Sweep". In: *ACM Transactions on Graphics* 32.6 (2013), pp. 1–10 (cit. on p. 47).

[Che+16] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. "InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets". In: *Advances In Neural Information Processing Systems*. 2016 (cit. on p. 98).

[Che+22] Xuanyao Chen, Tianyuan Zhang, Yue Wang, Yilun Wang, and Hang Zhao. "FUTR3D: A Unified Sensor Fusion Framework for 3D Detection". In: *arXiv::2203.10642* (2022) (cit. on p. 113).

[Cho+16] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. "3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, 2016, pp. 628–644 (cit. on p. 45).

[CK14] Neill DF Campbell and Jan Kautz. "Learning a Manifold of Fonts". In: *ACM Transactions on Graphics* 33.4 (2014), p. 91 (cit. on p. 46).

[Cla+17] Sebastian Claici, Mikhail Bessmeltsev, Scott Schaefer, and Justin Solomon. "Isometry-Aware Preconditioning for Mesh Parameterization". In: *Computer Graphics Forum*. Vol. 36. 2017, pp. 37–47 (cit. on pp. 19, 27).

[CLV20] David Cohen-Steiner, André Lieutier, and Julien Vuillamy. "Lexicographic Optimal Homologous Chains and Applications to Point Cloud Triangulations". In: *Proceedings of the 36th International Symposium on Computational Geometry*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020 (cit. on p. 80).

[CMP20] Julian Chibane, Aymen Mir, and Gerard Pons-Moll. "Neural Unsigned Distance Fields for Implicit Function Learning". In: *Advances in Neural Information Processing Systems*. 2020 (cit. on p. 13).

[Col+08] Forrester Cole, Aleksey Golovinskiy, Alex Limpaecher, Heather Stoddart Barros, Adam Finkelstein, Thomas Funkhouser, and Szymon Rusinkiewicz. "Where do people draw lines?" In: *ACM Transactions on Graphics* 27.3 (2008), pp. 1–11 (cit. on p. 68).

[Con67] Paul Concus. "Numerical Solution of the Minimal Surface Equation". In: *Mathematics of Computation* 21.99 (1967), pp. 340–350 (cit. on p. 79).

[Coo67] S. A. Coons. *Surfaces for Computer-Aided Design of Space Forms*. Tech. rep. Massachusetts Institute of Technology, 1967 (cit. on p. 55).

[Cor+17] Etienne Corman, Justin Solomon, Mirela Ben-Chen, Leonidas Guibas, and Maks Ovsjanikov. "Functional Characterization of Intrinsic and Extrinsic Geometry". In: *ACM Transactions on Graphics* 36.2 (2017), pp. 1–17 (cit. on p. 19).

[Cos+19] Luca Cosmo, Mikhail Panine, Arianna Rampini, Maks Ovsjanikov, Michael Bronstein, and Emanuele Rodolà. "Isospectralization, or how to hear shape, style, and correspondence". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 7529–7538 (cit. on p. 19).

[CP20] Eric Crawford and Joelle Pineau. "Exploiting Spatial Invariance for Scalable Unsupervised Object Tracking". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 2020, pp. 3684–3692 (cit. on p. 98).

[CPK18] Yoni Choukroun, Gautam Pai, and Ron Kimmel. "Sparse approximation of 3D meshes using the spectral geometry of the Hamiltonian operator". In: *Journal of Mathematical Imaging and Vision* 60.6 (2018), pp. 941–952 (cit. on p. 18).

[CPS11] Keenan Crane, Ulrich Pinkall, and Peter Schröder. "Spin Transformations of Discrete Surfaces". In: *ACM Transactions on Graphics* 30.4 (2011), pp. 1–10 (cit. on p. 79).

[CT14] Nicolas Charon and Alain Trouvé. "Functional Currents: a new mathematical tool to model and analyse functional shapes". In: *Journal of Mathematical Imaging and Vision* 48.3 (2014), pp. 413–431 (cit. on p. 80).

[CTZ20] Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. "BSP-Net: Generating Compact Meshes via Binary Space Partitioning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (cit. on pp. 13, 81).

[CW13] Ken Chan and Justin Wan. "Reconstruction of Missing Cells by a Killing Energy Minimizing Nonrigid Image Registration". In: *International Conference of the IEEE Engineering in Medicine and Biology Society*. 2013, pp. 3000–3003 (cit. on p. 19).

[CZ19] Zhiqin Chen and Hao Zhang. "Learning Implicit Fields for Generative Shape Modeling". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 5939–5948 (cit. on pp. 12, 78, 80).

[Dai+21] Zihang Dai, Hanxiao Liu, Quoc V Le, and Mingxing Tan. "CoAtNet: Marrying Convolution and Attention for All Data Sizes". In: *Advances In Neural Information Processing Systems*. 2021 (cit. on p. 10).

[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*. Vol. 29. 2016, pp. 3844–3852 (cit. on p. 20).

[Del+18] Johanna Delanoy, Mathieu Aubry, Phillip Isola, Alexei A Efros, and Adrien Bousseau. "3D Sketching using Multi-View Deep Volumetric Prediction". In: *Conference on Computer Graphics and Interactive Techniques* 1.1 (2018), pp. 1–22 (cit. on pp. 12, 44, 47, 72, 80).

[Den+20] Boyang Deng, Kyle Genova, Soroosh Yazdani, Sofien Bouaziz, Geoffrey Hinton, and Andrea Tagliasacchi. "CvxNet: Learnable Convex Decomposition". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (cit. on pp. 13, 81).

[Des+05] Mathieu Desbrun, Anil Hirani, Melvin Leok, and Jerrold E Marsden. "Discrete Exterior Calculus". In: *arXiv:math/0508341* (2005) (cit. on p. 24).

[Des+99] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H Barr. "Implicit Fairing of Irregular Meshes using Diffusion and Curvature Flow". In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. 1999, pp. 317–324 (cit. on p. 79).

[DH11] Nathan M Dunfield and Anil N Hirani. "The Least Spanning Area of a Knot and the Optimal Bounding Chain Problem". In: *Proceedings of the 27th Annual Symposium on Computational Geometry*. 2011, pp. 135–144 (cit. on p. 80).

[DHK11] Tamal K Dey, Anil N Hirani, and Bala Krishnamoorthy. "Optimal Homologous Cycles, Total Unimodularity, and Linear Programming". In: *SIAM Journal on Computing* 40.4 (2011), pp. 1026–1044 (cit. on p. 80).

[DL16] Chao Ding and Ligang Liu. "A Survey of Sketch Based Modeling Systems". In: *Frontiers of Computer Science* 10.6 (2016), pp. 985–999 (cit. on p. 47).

[Dou27] Jesse Douglas. "A Method of Numerical Solution of the Problem of Plateau". In: *Annals of Mathematics* (1927), pp. 180–188 (cit. on p. 79).

[DQN17] Angela Dai, Charles Ruizhongtai Qi, and Matthias Nießner. "Shape Completion using 3D-Encoder-Predictor CNNs and Shape Synthesis". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 3. 2017 (cit. on p. 46).

[DSO20] Nicolas Donati, Abhishek Sharma, and Maks Ovsjanikov. "Deep Geometric Functional Maps: Robust Feature Learning for Shape Correspondence". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 8592–8601 (cit. on p. 22).

[Du+18] Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. "InverseCSG: Automatic Conversion of 3D Models to CSG Trees". In: *ACM Transactions on Graphics* 37.6 (2018) (cit. on p. 76).

[Dub+18] Rachit Dubey, Pulkit Agrawal, Deepak Pathak, Thomas L. Griffiths, and Alexei A. Efros. "Investigating Human Priors for Playing Video Games". In: *Proceedings of the 35th International Conference on Machine Learning*. 2018 (cit. on p. 106).

[Dur+08] Stanley Durrleman, Xavier Pennec, Alain Trouvé, Paul Thompson, and Nicholas Ayache. "Inferring brain variability from diffeomorphic deformations of currents: an integrative approach". In: *Medical Image Analysis* 12.5 (2008), pp. 626–637 (cit. on p. 80).

[Dur+09] Stanley Durrleman, Xavier Pennec, Alain Trouvé, and Nicholas Ayache. "Statistical models of sets of curves and surfaces based on currents". In: *Medical Image Analysis* 13.5 (2009), pp. 793–808. ISSN: 1361-8415 (cit. on p. 80).

[Dur+11] Stanley Durrleman, Pierre Fillard, Xavier Pennec, Alain Trouvé, and Nicholas Ayache. "Registration, Atlas Estimation and Variability Analysis of White Matter Fiber Bundles Modeled as Currents". In: *NeuroImage* 55.3 (2011), pp. 1073–1090. ISSN: 1053-8119 (cit. on p. 80).

[Dut+20] Ionut Cosmin Duta, Li Liu, Fan Zhu, and Ling Shao. "Pyramidal Convolution: Rethinking Convolutional Neural Networks for Visual Recognition". In: *arXiv:2006.11538* (2020) (cit. on p. 10).

[Dzi90] Gerhard Dziuk. "An algorithm for evolutionary surfaces". In: *Numerische Mathematik* 58.1 (1990), pp. 603–611 (cit. on p. 79).

[EH96] Matthias Eck and Hugues Hoppe. "Automatic Reconstruction of B-Spline Surfaces of Arbitrary Topological Type". In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. 1996, pp. 325–334 (cit. on p. 70).

[EM03] David S Ebert and F Kenton Musgrave. *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann, 2003 (cit. on p. 51).

[Ent+15] Even Entem, Loic Barthe, Marie-Paule Cani, Frederic Cordier, and Michiel van de Panne. "Modeling 3D Animals from a Side-view Sketch". In: *Computer Graphics* 46 (C 2015), pp. 221–230 (cit. on p. 47).

[Esl+16] SM Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Koray Kavukcuoglu, and Geoffrey E Hinton. "Attend, Infer, Repeat: Fast Scene Understanding with Generative Models". In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. 2016 (cit. on p. 98).

[ET20] Moshe Eliasof and Eran Treister. "DiffGCN: Graph Convolutional Networks via Differential Operators and Algebraic Multigrid Pooling". In: *Advances in Neural Information Processing Systems*. 2020 (cit. on p. 22).

[Fed96] Herbert Federer. *Geometric Measure Theory*. Springer, 1996. ISBN: 978-3-540-60656-7 (cit. on pp. 80, 81, 84).

[Fen+19] Yutong Feng, Yifan Feng, Haoxuan You, Xibin Zhao, and Yue Gao. "MeshNet: Mesh Neural Network for 3D Shape Representation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 8279–8286 (cit. on p. 22).

[Fey+18] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. "SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 869–877 (cit. on p. 21).

[Fle15] Wendell H Fleming. *Geometric Measure Theory at Brown in the 1960s*. 2015 (cit. on p. 80).

[FRR15] Jorge Fuentes-Pacheco, José Ruiz-Ascencio, and Juan Manuel Rendón-Mancha. "Visual Simultaneous Localization and Mapping: A Survey". In: *Artificial Intelligence Review* 43.1 (2015), pp. 55–81 (cit. on p. 45).

[FSG17a] Haoqiang Fan, Hao Su, and Leonidas Guibas. "DeepPointSet: A Point Set Generation Network for 3D Object Reconstruction from a Single Image". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (cit. on pp. 12, 45, 48).

[FSG17b] Haoqiang Fan, Hao Su, and Leonidas J Guibas. "A Point Set Generation Network for 3D Object Reconstruction from a Single Image." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 2. 2017, p. 6 (cit. on pp. 44, 80).

[Gan+18] Vignesh Ganapathi-Subramanian, Olga Diamanti, Soeren Pirk, Chengcheng Tang, Matthias Niessner, and Leonidas Guibas. "Parsing Geometry Using Structure-Aware Shape Templates". In: *International Conference on 3D Vision (3DV)*. IEEE, 2018, pp. 672–681 (cit. on p. 46).

[Gao+19] Jun Gao, Chengcheng Tang, Vignesh Ganapathi-Subramanian, Jiahui Huang, Hao Su, and Leonidas J Guibas. "DeepSpline: Data-Driven Reconstruction of Parametric Curves and Surfaces". In: *arXiv:1901.03781* (2019) (cit. on pp. 12, 46).

[Gao+20] Jun Gao, Wenzheng Chen, Tommy Xiang, Alec Jacobson, Morgan McGuire, and Sanja Fidler. "Learning Deformable Tetrahedral Meshes for 3D Reconstruction". In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020 (cit. on p. 20).

[GDT16] Fernando de Goes, Mathieu Desbrun, and Yiying Tong. "Vector field processing on triangle meshes". In: *ACM SIGGRAPH 2016 Courses*. Association for Computing Machinery, 2016, pp. 1–49 (cit. on p. 19).

[Gen+19] Kyle Genova, Forrester Cole, Daniel Vlasic, Aaron Sarna, William T. Freeman, and Thomas Funkhouser. "Learning Shape Templates with Structured Implicit Functions". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019 (cit. on p. 46).

[Gen+20] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas Funkhouser. "Local Deep Implicit Functions for 3D Shape". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 4857–4866 (cit. on pp. 13, 81).

[GGH02] Xianfeng Gu, Steven Gortler, and Hugues Hoppe. "Geometry images". In: *ACM Transactions on Graphics* 21.3 (2002), pp. 355–361 (cit. on p. 21).

[Gir15] Ross Girshick. "Fast R-CNN". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2015, pp. 1440–1448 (cit. on p. 101).

[GIZ09] Yotam Gingold, Takeo Igarashi, and Denis Zorin. "Structured Annotations for 2D-to-3D Modeling". In: *ACM Transactions on Graphics* 28.5 (2009), p. 1 (cit. on p. 47).

[Goo+14] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative Adversarial Nets". In: *Advances In Neural Information Processing Systems*. 2014 (cit. on p. 98).

[Gro+18] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan Russell, and Mathieu Aubry. "AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018 (cit. on pp. 12, 44, 46, 48, 61–63, 72, 73).

[Gro+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*. 2020, pp. 3569–3579 (cit. on pp. 12, 80).

[Gry+19] Yulia Gryaditskaya, Mark Sypesteyn, Jan Willem Hoftijzer, Sylvia Pont, Frédo Durand, and Adrien Bousseau. "OpenSketch: A Richly-Annotated Dataset of Product Design Sketches". In: *ACM Transactions on Graphics* 38 (2019) (cit. on p. 68).

[GSI19] Yosef Gandelsman, Assaf Shocher, and Michal Irani. ""Double-DIP": Unsupervised Image Decomposition via Coupled Deep-Image-Priors". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 11026–11035 (cit. on p. 98).

[GTY04] Joan Glaunes, Alain Trouvé, and Laurent Younes. "Diffeomorphic matching of distributions: A new approach for unlabelled point-sets and sub-manifolds matching". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 2. IEEE, 2004 (cit. on p. 80).

[Haf+99] Patrick Haffner, Léon Bottou, Paul G Howard, and Yann LeCun. "DjVu: Analyzing and Compressing Scanned Documents for Internet Distribution". In: *Proceedings of the Fifth International Conference on Document Analysis and Recognition. ICDAR'99 (Cat. No. PR00318)*. IEEE, 1999, pp. 625–628 (cit. on p. 99).

[Hai+19] Niv Haim, Nimrod Segol, Heli Ben-Hamu, Haggai Maron, and Yaron Lipman. "Surface Networks via General Covers". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 632–641 (cit. on p. 21).

[Han+19] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. "MeshCNN: A Network with an Edge". In: *ACM Transactions on Graphics* 38.4 (2019), pp. 1–12 (cit. on pp. 12, 21, 37–39, 41).

[Han+20] Rana Hanocka, Gal Metzer, Raja Giryes, and Daniel Cohen-Or. "Point2Mesh: A Self-Prior for Deformable Meshes". In: *ACM Transactions on Graphics* 39.4 (2020). ISSN: 0730-0301 (cit. on pp. 20, 80).

[Hao+20] Zekun Hao, Hadar Averbuch-Elor, Noah Snavely, and Serge Belongie. "DualSDF: Semantic Shape Manipulation using a Two-Level Representation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 7631–7641 (cit. on pp. 13, 81).

[Här+20] Erik Härkönen, Aaron Hertzmann, Jaakko Lehtinen, and Sylvain Paris. "GANSpace: Discovering Interpretable GAN Controls". In: *Advances In Neural Information Processing Systems*. 2020 (cit. on p. 98).

[HBL15] Mikael Henaff, Joan Bruna, and Yann LeCun. "Deep Convolutional Networks on Graph-Structured Data". In: *arXiv:1506.05163* (2015) (cit. on p. 19).

[He+16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778 (cit. on pp. 51, 60).

[He+19] Zhen He, Jian Li, Daxue Liu, Hangen He, and David Barber. "Tracking by Animation: Unsupervised Learning of Multi-Object Attentive Trackers". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 1318–1327 (cit. on p. 98).

[He+20] 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 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*. 2020, pp. 2214–2224 (cit. on p. 21).

[Her+20] Amir Hertz, Rana Hanocka, Raja Giryes, and Daniel Cohen-Or. "Deep Geometric Texture Synthesis". In: *ACM Transactions on Graphics* 39.4 (2020) (cit. on p. 20).

[HFR99] Daniel Huttenlocher, Pedro Felzenszwalb, and William Rucklidge. "Digipaper: A Versatile Color Document Image Representation". In: *Proceedings 1999 International Conference on Image Processing (Cat. 99CH36348)*. Vol. 1. IEEE, 1999, pp. 219–223 (cit. on p. 99).

[Hil+12] Klaus Hildebrandt, Christian Schulz, Christoph von Tycowicz, and Konrad Polthier. "Modal Shape Analysis beyond Laplacian". In: *Computer Aided Geometric Design* 29.5 (2012), pp. 204–218 (cit. on p. 18).

[HM16] Jonathan Huang and Kevin Murphy. "Efficient inference in occlusion-aware generative models of images". In: *International Conference on Learning Representations (ICLR) Workshops*. 2016 (cit. on p. 98).

[HSG18] Jingwei Huang, Hao Su, and Leonidas Guibas. "Robust Watertight Manifold Surface Generation Method for ShapeNet Models". In: *arXiv:1802.01698* (2018) (cit. on p. 68).

[HSK74] Masahiro Hinata, Masaaki Shimasaki, and Takeshi Kiyono. "Numerical solution of Plateau's problem by a finite element method". In: *Mathematics of Computation* 28.125 (1974), pp. 45–60 (cit. on p. 79).

[HTM19] Christian Häne, Shubham Tulsiani, and Jitendra Malik. "Hierarchical Surface Prediction". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019) (cit. on p. 12).

[Hu+18] Yuanming Hu, Hao He, Chenxi Xu, Baoyuan Wang, and Stephen Lin. "Exposure: A White-Box Photo Post-Processing Framework". In: *ACM Transactions on Graphics* 37.2 (2018), pp. 1–17 (cit. on p. 98).

[Hua+09] Qi-Xing Huang, Martin Wicke, Bart Adams, and Leonidas Guibas. "Shape Decomposition using Modal Analysis". In: *Computer Graphics Forum*. Vol. 28. 2009, pp. 407–416 (cit. on p. 18).

[Hua+17] H. Huang, E. Kalogerakis, E. Yumer, and R. Mech. "Shape Synthesis from Sketches via Procedural Models and Convolutional Networks". In: *IEEE Transactions on Visualization and Computer Graphics* 23.8 (2017), pp. 2003–2013 (cit. on p. 47).

[Hua+19] Jingwei Huang, Haotian Zhang, Li Yi, Thomas Funkhouser, Matthias Nießner, and Leonidas Guibas. "TextureNet: Consistent Local Parametrizations for Learning from High-Resolution Signals on Meshes". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 4440–4449 (cit. on p. 21).

[IMT99] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. "Teddy: A Sketching Interface for 3D Freeform Design". In: *Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '99. ACM Press/Addison-Wesley Publishing Co., 1999, pp. 409–416. ISBN: 0-201-48560-5 (cit. on p. 47).

[Iso+17] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. "Image-to-Image Translation with Conditional Adversarial Networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (cit. on pp. 10, 43).

[Jad+15] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. "Spatial Transformer Networks". In: *Advances In Neural Information Processing Systems*. 2015 (cit. on p. 103).

[Jak+15] Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. "Instant Field-Aligned Meshes". In: *ACM Transactions on Graphics* 34.6 (2015) (cit. on pp. 71, 72).

[JF01] Nebojsa Jojic and Brendan J Frey. "Learning Flexible Sprites in Video Layers". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2001 (cit. on p. 97).

[JFK03] Nebojsa Jojic, Brendan J. Frey, and Anitha Kannan. "Epitomic analysis of appearance and shape". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2003, 34–41 vol.1 (cit. on p. 99).

[JGP17] Eric Jang, Shixiang Gu, and Ben Poole. "Categorical Reparameterization with Gumbel-Softmax". In: *International Conference on Learning Representations (ICLR)*. 2017 (cit. on p. 108).

[Jia+19] Jindong Jiang, Sepehr Janghorbani, Gerard De Melo, and Sungjin Ahn. "SCALOR: Generative World Models with Scalable Object Representations". In: *International Conference on Learning Representations (ICLR)*. 2019 (cit. on p. 98).

[Jus+19] Niels Justesen, Philip Bontrager, Julian Togelius, and Sebastian Risi. "Deep Learning for Video Game Playing". In: *IEEE Transactions on Games* 12.1 (2019), pp. 1–20 (cit. on p. 98).

[JZD98] Anil K. Jain, Yu Zhong, and Marie-Pierre Dubuisson-Jolly. "Deformable Template Models: A Review". In: *Signal Process.* 71.2 (1998), pp. 109–129 (cit. on p. 44).

[Kal+17] Evangelos Kalogerakis, Melinos Averkiou, Subhransu Maji, and Siddhartha Chaudhuri. "3D Shape Segmentation with Projective Convolutional Networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 3779–3788 (cit. on p. 22).

[Kar+20] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. "Analyzing and Improving the Image Quality of StyleGAN". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 8110–8119 (cit. on p. 98).

[KB14] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations (ICLR)*. 2014 (cit. on pp. 52, 60, 90).

[Kim+13] Vladimir G Kim, Wilmot Li, Niloy J Mitra, Siddhartha Chaudhuri, Stephen DiVerdi, and Thomas Funkhouser. "Learning Part-based Templates from Large Collections of 3D Shapes". In: *ACM Transactions on Graphics* 32.4 (2013), p. 70 (cit. on p. 47).

[Kim+20] Seung Wook Kim, Yuhao Zhou, Jonah Philion, Antonio Torralba, and Sanja Fidler. "Learning to Simulate Dynamic Environments with GameGAN". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (cit. on p. 99).

[KJF05] Anitha Kannan, Nebojsa Jojic, and Brendan J Frey. "Generative Model for Layers of Appearance and Deformation". In: *AISTATS*. Vol. 2005. Citeseer, 2005, p. 1 (cit. on p. 97).

[KL96] Venkat Krishnamurthy and Marc Levoy. "Fitting Smooth Surfaces to Dense Polygon Meshes". In: *Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. Association for Computing Machinery, 1996, pp. 313–324. ISBN: 0-89791-746-4 (cit. on p. 70).

[Kos+18a] Adam R Kosiorek, Hyunjik Kim, Ingmar Posner, and Yee Whye Teh. "Sequential Attend, Infer, Repeat: Generative Modelling of Moving Objects". In: *Advances In Neural Information Processing Systems*. 2018 (cit. on p. 98).

[Kos+18b] Ilya Kostrikov, Zhongshi Jiang, Daniele Panozzo, Denis Zorin, and Joan Bruna. "Surface networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 2540–2548 (cit. on p. 22).

[Kov+11] Artiom Kovnatsky, Michael Bronstein, Alexander Bronstein, and Ron Kimmel. "Photometric Heat Kernel Signatures". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2011, pp. 616–627 (cit. on p. 19).

[KSB12] Michael Kazhdan, Jake Solomon, and Mirela Ben-Chen. "Can Mean-Curvature Flow Be Made Non-Singular?" In: *Computer Graphics Forum*. Vol. 31. Wiley Online Library, 2012, pp. 1745–1754 (cit. on p. 79).

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances In Neural Information Processing Systems*. 2012, pp. 1097–1105 (cit. on p. 43).

[KUH18] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. "Neural 3D Mesh Renderer". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018 (cit. on p. 12).

[KW17] Thomas Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *International Conference on Learning Representations (ICLR)*. 2017 (cit. on pp. 20, 21).

[Lan04] Urs Lang. *Introduction to Geometric Measure Theory*. 2004 (cit. on pp. 81, 84).

[LB13] Roee Litman and Alexander Bronstein. "Learning Spectral Descriptors for Deformable Shape Correspondence". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.1 (2013), pp. 171–180 (cit. on pp. 19, 35).

[LDG18] Yiyi Liao, Simon Donné, and Andreas Geiger. "Deep Marching Cubes: Learning Explicit Surface Representations". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 2916–2925 (cit. on p. 46).

[Lev+18] Ron Levie, Federico Monti, Xavier Bresson, and Michael Bronstein. "CayleyNets: Graph Convolutional Neural Networks with Complex Rational Spectral Filters". In: *IEEE Transactions on Signal Processing* 67 (2018), pp. 97–109 (cit. on p. 20).

[LF04] Xia Liu and Kikuo Fujimura. "Hand gesture recognition using depth data". In: *Proceedings of the Sixth IEEE International Conference on Automatic Face and Gesture Recognition*. IEEE, 2004, p. 529 (cit. on pp. 44, 48).

[LH19] Ilya Loshchilov and Frank Hutter. "Decoupled Weight Decay Regularization". In: *International Conference on Learning Representations (ICLR)*. 2019 (cit. on pp. 36, 105).

[Li+17] Changjian Li, Hao Pan, Xin Tong, Alla Sheffer, and Wenping Wang. "BendSketch: Modeling Freeform Surfaces Through 2D Sketching". In: *ACM Transactions on Graphics* 36.4 (2017) (cit. on p. 47).

[Li+18] Tzu-Mao Li, Michaël Gharbi, Andrew Adams, Frédo Durand, and Jonathan Ragan-Kelley. "Differentiable Programming for Image Processing and Deep Learning in Halide". In: *ACM Transactions on Graphics* 37.4 (2018), pp. 1–13 (cit. on p. 98).

[Li+20] Tzu-Mao Li, Michal Lukáč, Michaël Gharbi, and Jonathan Ragan-Kelley. "Differentiable Vector Graphics Rasterization for Editing and Learning". In: *ACM Transactions on Graphics* 39.6 (2020), pp. 1–15 (cit. on p. 98).

[Lia+11] Z. Lian, A. Godil, B. Bustos, M. Daoudi, J. Hermans, S. Kawamura, Y. Kurita, G. Lavoué, H. V. Nguyen, R. Ohbuchi, Y. Ohkita, Y. Ohishi, F. Porikli, M. Reuter, I. Sipiran, D. Smeets, P. Suetens, H. Tabia, and D. Vandermeulen. "SHREC '11 Track: Shape Retrieval on Non-rigid 3D Watertight Meshes". In: *Eurographics Workshop on 3D Object Retrieval*. Ed. by H. Laga, T. Schreck, A. Ferreira, A. Godil, I. Pratikakis, and R. Veltkamp. The Eurographics Association, 2011. ISBN: 978-3-905674-31-6 (cit. on p. 38).

[Lim+18] 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 (ECCV)*. 2018 (cit. on p. 22).

[Lin+18] Chen-Hsuan Lin, Ersin Yumer, Oliver Wang, Eli Shechtman, and Simon Lucey. "ST-GAN: Spatial Transformer Generative Adversarial Networks for Image Compositing". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 9455–9464 (cit. on p. 98).

[Lin+20] Zhixuan Lin, Yi-Fu Wu, Skand Vishwanath Peri, Weihao Sun, Gautam Singh, Fei Deng, Jindong Jiang, and Sungjin Ahn. "SPACE: Unsupervised Object-Oriented Scene Representation via Spatial Attention and Decomposition". In: *International Conference on Learning Representations (ICLR)*. 2020 (cit. on pp. 98, 103, 105, 106).

[Lip21] Yaron Lipman. "Phase Transitions, Distance Functions, and Implicit Neural Representations". In: *Proceedings of the 38th International Conference on Machine Learning*. 2021 (cit. on pp. 12, 80).

[Lit+17] Or Litany, Tal Remez, Emanuele Rodolà, Alex Bronstein, and Michael Bronstein. "Deep Functional Maps: Structured Prediction for Dense Shape Correspondence". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2017, pp. 5659–5667 (cit. on p. 22).

[Liu+05] Ce Liu, Antonio Torralba, William T Freeman, Frédo Durand, and Edward H Adelson. "Motion Magnification". In: *ACM Transactions on Graphics* 24.3 (2005), pp. 519–526 (cit. on p. 106).

[Liu+18] Guilin Liu, Kevin J. Shih, Ting-Chun Wang, Fitsum A. Reda, Karan Sapra, Zhiding Yu, Andrew Tao, and Bryan Catanzaro. "Partial Convolution based Padding". In: *arXiv:1811.11718* (2018) (cit. on p. 102).

[Liu+20] Hsueh-Ti Derek Liu, Vladimir Kim, Siddhartha Chaudhuri, Noam Aigerman, and Alec Jacobson. "Neural Subdivision". In: *ACM Transactions on Graphics* 39.4 (2020) (cit. on p. 20).

[Liu+21] Shi-Lin Liu, Hao-Xiang Guo, Hao Pan, Pengshuai Wang, Xin Tong, and Yang Liu. "Deep Implicit Moving Least-Squares Functions for 3D Reconstruction". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021 (cit. on p. 81).

[LJC17] Hsueh-Ti Derek Liu, Alec Jacobson, and Keenan Crane. "A Dirac Operator for Extrinsic Shape Analysis". In: *Computer Graphics Forum*. Vol. 36. 2017, pp. 139–149 (cit. on pp. 18, 27).

[Loc+20] Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. "Object-Centric Learning with Slot Attention". In: *arXiv:2006.15055* (2020) (cit. on pp. 98, 103, 105).

[Lom+19] Salvator Lombardo, Jun Han, Christopher Schroers, and Stephan Mandt. "Deep Generative Video Compression". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019 (cit. on p. 99).

[LRS18] Chenxi Liu, Enrique Rosales, and Alla Sheffer. "StrokeAggregator: Consolidating Raw Sketches into Artist-Intended Curve Drawings". In: *ACM Transactions on Graphics* 37.4 (2018) (cit. on p. 68).

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3431–3440 (cit. on p. 43).

[Lu+19] Guo Lu, Wanli Ouyang, Dong Xu, Xiaoyun Zhang, Chunlei Cai, and Zhiyong Gao. "DVC: An End-to-end Deep Video Compression Framework". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019 (cit. on p. 99).

[Lu+20] Erika Lu, Forrester Cole, Tali Dekel, Andrew Zisserman, David Salesin, William T. Freeman, and Michael Rubinstein. "Layered Neural Rendering for Retiming People in Video". In: *ACM Transactions on Graphics* (2020) (cit. on p. 97).

[Lu+21] Erika Lu, Forrester Cole, Tali Dekel, Andrew Zisserman, William T Freeman, and Michael Rubinstein. "Omnimatte: Associating Objects and Their Effects in Video". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021 (cit. on p. 97).

[Lun+17] Zhaoliang Lun, Matheus Gadelha, Evangelos Kalogerakis, Subhransu Maji, and Rui Wang. "3D Shape Reconstruction from Sketches via Multi-view Convolutional Networks". In: *International Conference on 3D Vision (3DV)*. 2017 (cit. on pp. 12, 44, 47, 72, 73).

[LZ07] Rong Liu and Hao Zhang. "Mesh Segmentation via Spectral Embedding and Contour Analysis". In: *Computer Graphics Forum*. Vol. 26. 2007, pp. 385–394 (cit. on p. 18).

[Mai+10] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. "Online learning for matrix factorization and sparse coding." In: *Journal of Machine Learning Research* 11.1 (2010) (cit. on p. 106).

[Man+18] Priyanka Mandikal, K L Navaneet, Mayank Agarwal, and R Venkatesh Babu. "3D-LMNet: Latent Embedding Matching for Accurate and Diverse 3D Point Cloud Reconstruction from a Single Image". In: *BMVC*. 2018 (cit. on p. 12).

[Mar+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". In: *ACM Transactions on Graphics* 36.4 (2017) (cit. on pp. 21, 37).

[Mas+15] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. "Geodesic convolutional neural networks on Riemannian manifolds". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2015, pp. 37–45 (cit. on p. 20).

[MC19] Thomas Mollenhoff and Daniel Cremers. "Lifting vectorial variational problems: a natural formulation based on geometric measure theory and discrete exterior calculus". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 11117–11126 (cit. on pp. 80, 94).

[Mes+19] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. "Occupancy Networks: Learning 3D Reconstruction in Function Space". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019 (cit. on pp. 12, 78, 80).

[MHN13] A.L. Maas, A.Y. Hannun, and A.Y. Ng. "Rectifier Nonlinearities Improve Neural Network Acoustic Models". In: *Proceedings of the 30th International Conference on Machine Learning*. 2013 (cit. on p. 36).

[Mil+20a] Francesco Milano, Antonio Loquercio, Antoni Rosinol, Davide Scaramuzza, and Luca Carlone. "Primal-Dual Mesh Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. 2020 (cit. on pp. 12, 21, 37–41).

[Mil+20b] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. "Nerf: Representing scenes as neural radiance fields for view synthesis". In: *European Conference on Computer Vision*. Springer, 2020, pp. 405–421 (cit. on p. 98).

[MKK21] Thomas W. Mitchel, Vladimir G. Kim, and Michael Kazhdan. "Field Convolutions for Surface CNNs". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 10001–10011 (cit. on p. 21).

[Mo+19] Kaichun Mo, Paul Guerrero, Li Yi, Hao Su, Peter Wonka, Niloy Mitra, and Leonidas Guibas. "StructureNet: Hierarchical Graph Networks for 3D Shape Generation". In: *ACM Transactions on Graphics* 38.6 (2019) (cit. on p. 46).

[Mon+17] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael Bronstein. "Geometric deep learning on graphs and manifolds using mixture model CNNs". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 (cit. on p. 20).

[Mor03] Frank Morgan. "Regularity of isoperimetric hypersurfaces in Riemannian manifolds". In: *Transactions of the American Mathematical Society* 355.12 (2003), pp. 5041–5052 (cit. on p. 86).

[Mor16] Frank Morgan. *Geometric measure theory: a beginner's guide*. Academic Press, 2016 (cit. on p. 80).

[Mul+07] Patrick Mullen, Alexander McKenzie, Yiying Tong, and Mathieu Desbrun. "A variational approach to Eulerian geometry processing". In: *ACM Transactions on Graphics* 26.3 (2007) (cit. on p. 80).

[Nas+20] Charlie Nash, Yaroslav Ganin, SM Ali Eslami, and Peter Battaglia. "PolyGen: An Autoregressive Generative Model of 3D Meshes". In: *Proceedings of the 37th International Conference on Machine Learning*. PMLR, 2020, pp. 7220–7229 (cit. on pp. 12, 80).

[Nis+16] Gen Nishida, Ignacio Garcia-Dorado, Daniel G. Aliaga, Bedrich Benes, and Adrien Bousseau. "Interactive Sketching of Urban Procedural Models". In: *ACM Transactions on Graphics* 35.4 (2016) (cit. on p. 47).

[NLX18] Chengjie Niu, Jun Li, and Kai Xu. "Im2Struct: Recovering 3D shape structure from a single RGB image". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 4521–4529 (cit. on p. 46).

[ODo+14] Peter O'Donovan, Jānis Lībeks, Aseem Agarwala, and Aaron Hertzmann. "Exploratory font selection using crowdsourced attributes". In: *ACM Transactions on Graphics* 33.4 (2014), p. 92 (cit. on p. 46).

[Ols+09] Luke Olsen, Faramarz F. Samavati, Mario Costa Sousa, and Joaquim A. Jorge. "Sketch-based modeling: A survey". In: *Computers & Graphics* 33.1 (2009), pp. 85–103 (cit. on p. 47).

[OSG08] Maks Ovsjanikov, Jian Sun, and Leonidas Guibas. "Global intrinsic symmetries of shapes". In: *Computer Graphics Forum*. Vol. 27. 2008, pp. 1341–1348 (cit. on p. 18).

[Ovs+10] Maks Ovsjanikov, Quentin Mérigot, Facundo Mémoli, and Leonidas Guibas. "One point isometric matching with the heat kernel". In: *Computer Graphics Forum*. Vol. 29. 2010, pp. 1555–1564 (cit. on p. 18).

[Ovs+11] Maks Ovsjanikov, Wilmot Li, Leonidas Guibas, and Niloy J Mitra. "Exploration of continuous variability in collections of 3D shapes". In: *ACM Transactions on Graphics* 30.4 (2011), pp. 1–10 (cit. on p. 47).

[Ovs+12] Maks Ovsjanikov, Mirela Ben-Chen, Justin Solomon, Adrian Butscher, and Leonidas Guibas. "Functional maps: A flexible representation of maps between shapes". In: *ACM Transactions on Graphics* 31.4 (2012), pp. 1–11 (cit. on p. 18).

[Pan+12] Hao Pan, Yi-King Choi, Yang Liu, Wenchao Hu, Qiang Du, Konrad Polthier, Caiming Zhang, and Wenping Wang. "Robust Modeling of Constant Mean Curvature Surfaces". In: *ACM Transactions on Graphics* 31.4 (2012), pp. 1–11 (cit. on p. 79).

[Par+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 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019 (cit. on pp. 12, 46, 78, 80).

[PD84] Thomas Porter and Tom Duff. "Compositing. Digital Images". In: *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*. 1984, pp. 253–259 (cit. on p. 104).

[PFC15] Huy Quoc Phan, Hongbo Fu, and Antoni B Chan. "Flexyfont: Learning transferring rules for flexible typeface synthesis". In: *Computer Graphics Forum*. Vol. 34. Wiley Online Library, 2015, pp. 245–256 (cit. on p. 46).

[PO18] Adrien Poulenard and Maks Ovsjanikov. "Multi-directional geodesic neural networks via equivariant convolution". In: *ACM Transactions on Graphics* 37.6 (2018), pp. 1–14 (cit. on p. 21).

[Pop96] EV Popov. "On some variational formulations for minimum surface". In: *Transactions of the Canadian Society for Mechanical Engineering* 20.4 (1996), pp. 391–400 (cit. on p. 79).

[PP93] Ulrich Pinkall and Konrad Polthier. "Computing Discrete Minimal Surfaces and Their Conjugates". In: *Experimental Mathematics* 2.1 (1993), pp. 15–36 (cit. on p. 79).

[PP97] Harold R Parks and Jon T Pitts. "Computing Least Area Hypersurfaces Spanning Arbitrary Boundaries". In: *SIAM Journal on Scientific Computing* 18.3 (1997), pp. 886–917 (cit. on p. 80).

[PUG19] Despoina Paschalidou, Ali Osman Ulusoy, and Andreas Geiger. "Superquadrics Revisited: Learning 3D Shape Parsing beyond Cuboids". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019 (cit. on pp. 46, 54, 80).

[Qia+20] Yi-Ling Qiao, Lin Gao, Paul Rosin, Yu-Kun Lai, Xilin Chen, et al. "Learning on 3D meshes with Laplacian encoding and pooling". In: *IEEE Transactions on Visualization and Computer Graphics* (2020) (cit. on p. 22).

[QMK06] Zheng Qin, Michael D McCool, and Craig S Kaplan. "Real-Time Texture-Mapped Vector Glyphs". In: *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*. ACM, 2006, pp. 125–132 (cit. on p. 52).

[Rad+21] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. "Learning transferable visual models from natural language supervision". In: *Proceedings of the 38th International Conference on Machine Learning*. 2021, pp. 8748–8763 (cit. on p. 113).

[Rav+10] Dan Raviv, Michael Bronstein, Alexander Bronstein, and Ron Kimmel. "Volumetric heat kernel signatures". In: *Proceedings of the ACM Workshop on 3D Object Retrieval*. 2010, pp. 39–44 (cit. on pp. 18, 19).

[Rav+11] Dan Raviv, Michael Bronstein, Alexander Bronstein, Ron Kimmel, and Nir Sochen. "Affine-invariant diffusion geometry for the analysis of deformable 3D shapes". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2011, pp. 2361–2367 (cit. on p. 19).

[Red+16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788 (cit. on p. 101).

[Red+20] Pradyumna Reddy, Paul Guerrero, Matt Fisher, Wilmot Li, and Niloy J. Mitra. "Discovering Pattern Structure Using Differentiable Compositing". In: *ACM Transactions on Graphics* 39.6 (2020), 262:1–262:15 (cit. on p. 98).

[Rez+16] Danilo Jimenez Rezende, SM Ali Eslami, Shakir Mohamed, Peter Battaglia, Max Jaderberg, and Nicolas Heess. "Unsupervised learning of 3D structure from images". In: *Advances in neural information processing systems*. 2016, pp. 4996–5004 (cit. on p. 12).

[Rus07] Raif Rustamov. "Laplace-Beltrami eigenfunctions for deformation invariant shape representation". In: *Symposium on Geometry Processing*. 2007, pp. 225–233 (cit. on p. 18).

[RWP06] Martin Reuter, Franz-Erich Wolter, and Niklas Peinecke. "Laplace–Beltrami spectra as 'shape-DNA' of surfaces and solids". In: *Computer-Aided Design* 38.4 (2006), pp. 342–366 (cit. on p. 18).

[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 (ECCV)*. 2016 (cit. on p. 21).

[SBS21] Dmitriy Smirnov, Mikhail Bessmeltsev, and Justin Solomon. "Learning Manifold Patch-Based Representations of Man-Made Shapes". In: *International Conference on Learning Representations (ICLR)*. 2021 (cit. on p. 42).

[SCA20] Othman Sbai, Camille Couprie, and Mathieu Aubry. "Unsupervised Image Decomposition in Vector Layers". In: *IEEE International Conference on Image Processing*. IEEE, 2020 (cit. on p. 97).

[Sch+17] Adriana Schulz, Ariel Shamir, Ilya Baran, David IW Levin, Pitchaya Sitthi-Amorn, and Wojciech Matusik. "Retrieval on Parametric Shape Collections". In: *ACM Transactions on Graphics* 36.1 (2017), p. 11 (cit. on p. 47).

[Sch+20] Jonas Schult, Francis Engelmann, Theodora Kontogianni, and Bastian Leibe. "DualConvMesh-Net: Joint Geodesic and Euclidean Convolutions on 3D Meshes". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 8612–8622 (cit. on p. 21).

[SDL18] Stefan Schonsheck, Bin Dong, and Rongjie Lai. "Parallel Transport Convolution: A New Tool for Convolutional Neural Networks on Manifolds". In: *arXiv:1805.07857* (2018) (cit. on p. 21).

[Sei+06] Steven M Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. "A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2006 (cit. on p. 45).

[SG18] David Stutz and Andreas Geiger. "Learning 3D Shape Completion under Weak Supervision". In: *International Journal of Computer Vision* (2018), pp. 1–20 (cit. on p. 46).

[Sha+20] Gopal Sharma, Difan Liu, Subhransu Maji, Evangelos Kalogerakis, Siddhartha Chaudhuri, and Radomír Měch. "ParSeNet: A parametric surface fitting network for 3D point clouds". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, 2020, pp. 261–276 (cit. on p. 80).

[Sha+22] Nicholas Sharp, Souhaib Attaiki, Keenan Crane, and Maks Ovsjanikov. "DiffusionNet: Discretization Agnostic Learning on Surfaces". In: *ACM Transactions on Graphics* (2022) (cit. on pp. 12, 22).

[She+12] Chao-Hui Shen, Hongbo Fu, Kang Chen, and Shi-Min Hu. "Structure recovery by part assembly". In: *ACM Transactions on Graphics* 31.6 (2012), p. 180 (cit. on p. 47).

[Shi+15] Baoguang Shi, Song Bai, Zhichao Zhou, and Xiang Bai. "Deep Panoramic Representation for 3-D Shape Recognition". In: *IEEE Signal Processing Letters* 22.12 (2015), pp. 2339–2343 (cit. on p. 22).

[SI10] Rapee Suveeranont and Takeo Igarashi. "Example-based automatic font generation". In: *International Symposium on Smart Graphics*. Springer, 2010, pp. 127–138 (cit. on p. 46).

[SII18] Edgar Simo-Serra, Satoshi Iizuka, and Hiroshi Ishikawa. "Mastering Sketching: Adversarial Augmentation for Structured Prediction". In: *ACM Transactions on Graphics* 37.1 (2018) (cit. on p. 68).

[Sim14] Leon Simon. "Introduction to geometric measure theory". In: *Tsinghua Lectures* (2014) (cit. on pp. 81, 84).

[Sla+17] Miroslava Slavcheva, Maximilian Baust, Daniel Cremers, and Slobodan Ilic. "KillingFusion: Non-rigid 3D reconstruction without correspondences". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 1386–1395 (cit. on p. 19).

[Smi+20] Dmitriy Smirnov, Matthew Fisher, Vladimir G. Kim, Richard Zhang, and Justin Solomon. "Deep Parametric Shape Predictions using Distance Fields". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (cit. on pp. 42, 56, 98).

[Smi+21] Dmitriy Smirnov, Michaël Gharbi, Matthew Fisher, Vitor Guizilini, Alexei A. Efros, and Justin Solomon. "MarioNette: Self-Supervised Sprite Learning". In: *Advances in Neural Information Processing Systems*. 2021 (cit. on p. 96).

[Smi+22] Dmitriy Smirnov*, David Palmer*, Stephanie Wang, Albert Chern, and Justin Solomon. "DeepCurrents: Learning Implicit Representations of Shapes with Boundaries". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022 (cit. on p. 77).

[SO20a] Abhishek Sharma and Maks Ovsjanikov. "Weakly supervised deep functional map for shape matching". In: *Advances in Neural Information Processing Systems*. 2020 (cit. on p. 22).

[SO20b] Nicholas Sharp and Maks Ovsjanikov. "PointTriNet: Learned Triangulation of 3D Point Sets". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2020 (cit. on p. 20).

[SOG09] Jian Sun, Maks Ovsjanikov, and Leonidas Guibas. "A Concise and Provably Informative Multi-Scale Signature Based on Heat Diffusion". In: *Computer Graphics Forum*. Vol. 28. 2009, pp. 1383–1392 (cit. on pp. 18, 35).

[Sol+11a] Justin Solomon, Mirela Ben-Chen, Adrian Butscher, and Leonidas Guibas. "As-Killing-As-Possible Vector Fields for Planar Deformation". In: *Computer Graphics Forum*. Vol. 30. 2011, pp. 1543–1552 (cit. on pp. 19, 27).

[Sol+11b] Justin Solomon, Mirela Ben-Chen, Adrian Butscher, and Leonidas Guibas. "Discovery of Intrinsic Primitives on Triangle Meshes". In: *Computer Graphics Forum*. Vol. 30. 2011, pp. 365–374 (cit. on p. 19).

[Son+20] An Ping Song, Xin Yi Di, Xiao Kang Xu, and Zi Heng Song. "MeshGraphNet: An effective 3D polygon mesh recognition With topology reconstruction". In: *IEEE Access* 8 (2020), pp. 205181–205189 (cit. on p. 21).

[Spa+12] M. Spagnuolo, M. Bronstein, A. Bronstein, and A. Ferreira. "Affine-Invariant Photometric Heat Kernel Signatures". In: *Eurographics* (2012), pp. 39–46 (cit. on p. 19).

[SS21] Dmitriy Smirnov and Justin Solomon. "HodgeNet: Learning Spectral Geometry on Triangle Meshes". In: *ACM Transactions on Graphics (TOG)* 40.4 (2021), 166:1–166:11 (cit. on p. 15).

[SSC19] Nicholas Sharp, Yousuf Soliman, and Keenan Crane. "The vector heat method". In: *ACM Transactions on Graphics* 38.3 (2019), pp. 1–19 (cit. on p. 19).

[STP17] Konstantinos Sfikas, Theoharis Theoharis, and Ioannis Pratikakis. "Exploiting the PANORAMA representation for convolutional neural network classification and retrieval". In: *Proceedings of the Workshop on 3D Object Retrieval*. Vol. 6. 2017, p. 7 (cit. on p. 22).

[Su+15] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. "Multi-view convolutional neural networks for 3D shape recognition". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2015, pp. 945–953 (cit. on pp. 22, 45).

[Sul90] John M Sullivan. "A Crystalline Approximation Theorem for Hypersurfaces". In: *Ann Arbor, Thesis (Ph.D.), Princeton University* (1990) (cit. on p. 80).

[Sum+18] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. "Procedural Content Generation via Machine Learning (PCGML)". In: *IEEE Transactions on Games* 10.3 (2018), pp. 257–270 (cit. on p. 99).

[Sun+19] Chunyu Sun, Qianfang Zou, Xin Tong, and Yang Liu. "Learning Adaptive Hierarchical Cuboid Abstractions of 3D Shape Collections". In: *ACM Transactions on Graphics* 38.6 (2019) (cit. on pp. 46, 56).

[Sun+20] Zhiyu Sun, Ethan Rooke, Jerome Charton, Yusen He, Jia Lu, and Stephen Baek. "ZerNet: Convolutional neural networks on arbitrary surfaces via Zernike local tangent space estimation". In: *Computer Graphics Forum*. 2020 (cit. on pp. 20, 21).

[SW12] Amit Singer and H.-T. Wu. "Vector diffusion maps and the connection Laplacian". In: *Communications on Pure and Applied Mathematics* 65.8 (2012), pp. 1067–1144 (cit. on pp. 19, 35).

[SW19] Henrik Schumacher and Max Wardetzky. "Variational convergence of discrete minimal surfaces". In: *Numerische Mathematik* 141.1 (2019), pp. 173–213 (cit. on p. 79).

[Tak+21] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. "Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021 (cit. on pp. 12, 80).

[Tan+20] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. "Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains". In: *Advances in Neural Information Processing Systems*. 2020 (cit. on p. 89).

[Tat+18] Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou. "Tangent convolutions for dense prediction in 3D". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 3887–3896 (cit. on p. 20).

[TEM18] Shubham Tulsiani, Alexei A Efros, and Jitendra Malik. "Multi-view consistency as supervisory signal for learning shape and pose prediction". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 2897–2905 (cit. on p. 12).

[Tul+17] Shubham Tulsiani, Hao Su, Leonidas J Guibas, Alexei A Efros, and Jitendra Malik. "Learning shape abstractions by assembling volumetric primitives". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 2. 2017 (cit. on pp. 43, 44, 48, 56, 66, 67, 80).

[TZN19] Justus Thies, Michael Zollhöfer, and Matthias Nießner. "Deferred Neural Rendering: Image Synthesis using Neural Textures". In: *ACM Transactions on Graphics* 38.4 (2019), pp. 1–12 (cit. on p. 98).

[UIM12] Nobuyuki Umetani, Takeo Igarashi, and Niloy J Mitra. "Guided exploration of physically valid shapes for furniture design." In: *ACM Transactions on Graphics* 31.4 (2012), pp. 86–1 (cit. on p. 47).

[USB16] Paul Upchurch, Noah Snavely, and Kavita Bala. "From A to Z: Supervised Transfer of Style and Content Using Deep Neural Network Generators". In: *arXiv:1603.02003* (2016) (cit. on p. 46).

[UVL18] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. "Deep Image Prior". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 9446–9454 (cit. on p. 98).

[VBV18] Nitika Verma, Edmond Boyer, and Jakob Verbeek. "FeastNet: Feature-steered graph convolutions for 3D shape analysis". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 2598–2606 (cit. on p. 20).

[Ven+21] Rahul Venkatesh, Tejan Karmali, Sarthak Sharma, Aurobrata Ghosh, R. Venkatesh Babu, Laszlo A. Jeni, and Maneesh Singh. "Deep Implicit Surface Point Prediction Networks". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 12653–12662 (cit. on pp. 13, 92).

[VG05] Marc Vaillant and Joan Glaunes. "Surface matching via currents". In: *Biennial International Conference on Information Processing in Medical Imaging*. Springer, 2005, pp. 381–392 (cit. on p. 80).

[Vis+19] Joost Visser, Alessandro Corbetta, Vlado Menkovski, and Federico Toschi. "StampNet: unsupervised multi-class object discovery". In: *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2019, pp. 2951–2955 (cit. on p. 98).

[Vla+08] Daniel Vlasic, Ilya Baran, Wojciech Matusik, and Jovan Popović. "Articulated Mesh Animation from Multi-view Silhouettes". In: *ACM Transactions on Graphics* 27.3 (2008) (cit. on pp. 15, 38, 40).

[WA94] J. Y. A. Wang and E. H. Adelson. "Representing Moving Images with Layers". In: *IEEE Transactions on Image Processing* 3.5 (1994), pp. 625–638 (cit. on p. 97).

[Wag77] H-J Wagner. "A contribution to the numerical approximation of minimal surfaces". In: *Computing* 19.1 (1977), pp. 35–58 (cit. on p. 79).

[Wan+12] Yunhai Wang, Shmulik Asafi, Oliver Van Kaick, Hao Zhang, Daniel Cohen-Or, and Baoquan Chen. "Active Co-Analysis of a Set of Shapes". In: *ACM Transactions on Graphics* 31.6 (2012), pp. 1–10 (cit. on p. 37).

[Wan+14] Hao Wang, Tong Lu, Oscar Kin-Chung Au, and Chiew-Lan Tai. "Spectral 3D mesh segmentation with a novel single segmentation field". In: *Graphical Models* 76.5 (2014), pp. 440–456 (cit. on p. 18).

[Wan+18a] Lingjing Wang, Jifei Wang, Cheng Qian, and Yi Fang. "Unsupervised learning of 3D model reconstruction from hand-drawn sketches". In: *ACM MM*. ACM MM. Association for Computing Machinery, Inc, 2018, pp. 1820–1828 (cit. on p. 47).

[Wan+18b] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. "Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018 (cit. on pp. 12, 72, 73, 80).

[Wan+18c] Pengyu Wang, Yuan Gan, Panpan Shui, Fenggen Yu, Yan Zhang, Songle Chen, and Zhengxing Sun. "3D Shape Segmentation via Shape Fully Convolutional Networks". In: *Computers & Graphics* 70 (2018), pp. 128–139 (cit. on pp. 21, 44).

[Wan+18d] Yu Wang, Mirela Ben-Chen, Iosif Polterovich, and Justin Solomon. "Steklov spectral geometry for extrinsic shape analysis". In: *ACM Transactions on Graphics* 38.1 (2018), pp. 1–21 (cit. on pp. 18, 19).

[Wan+19] Yu Wang, Vladimir Kim, Michael Bronstein, and Justin Solomon. "Learning geometric operators on meshes". In: *International Conference on Learning Representations (ICLR) Workshops*. 2019 (cit. on pp. 22, 24).

[WC21] Stephanie Wang and Albert Chern. "Computing Minimal Surfaces with Differential Forms". In: *ACM Transactions on Graphics* 40.4 (2021), pp. 1–14 (cit. on pp. 80, 84, 86, 90, 91).

[WEH20] Ruben Wiersma, Elmar Eisemann, and Klaus Hildebrandt. "CNNs on Surfaces using Rotation-Equivariant Features". In: *ACM Transactions on Graphics* 39.4 (2020) (cit. on pp. 12, 21).

[Wei+16] Lingyu Wei, Qixing Huang, Duygu Ceylan, Etienne Vouga, and Hao Li. "Dense Human Body Correspondences Using Convolutional Networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 1544–1553 (cit. on p. 22).

[WH18] Yuxin Wu and Kaiming He. "Group normalization". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 3–19 (cit. on p. 101).

[WH21] Jianfeng Wang and Xiaolin Hu. "Convolutional Neural Networks with Gated Recurrent Connections". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021) (cit. on p. 10).

[Wil+19] Francis Williams, Teseo Schneider, Claudio Silva, Denis Zorin, Joan Bruna, and Daniele Panozzo. "Deep Geometric Prior for Surface Reconstruction". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 10130–10139 (cit. on p. 12).

[Wil61] Walter L Wilson. “On discrete Dirichlet and Plateau problems”. In: *Numerische Mathematik* 3.1 (1961), pp. 359–373 (cit. on p. 79).

[WP09] Steffen Weißmann and Ulrich Pinkall. “Real-time Interactive Simulation of Smoke Using Discrete Integrable Vortex Filaments”. In: *Workshop in Virtual Reality Interactions and Physical Simulation "VRIPHYS" (2009)*. Ed. by Hartmut Prautzsch, Alfred Schmitt, Jan Bender, and Matthias Teschner. The Eurographics Association, 2009. ISBN: 978-3-905673-73-9 (cit. on p. 85).

[WS19] Yu Wang and Justin Solomon. “Intrinsic and extrinsic operators for shape analysis”. In: *Handbook of Numerical Analysis*. Vol. 20. Elsevier, 2019, pp. 41–115 (cit. on p. 18).

[Wu+16] Jiajun Wu, Tianfan Xue, Joseph J Lim, Yuandong Tian, Joshua B Tenenbaum, Antonio Torralba, and William T Freeman. “Single Image 3D Interpreter Network”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2016 (cit. on p. 12).

[Wu+17] Jiajun Wu, Yifan Wang, Tianfan Xue, Xingyuan Sun, William T Freeman, and Joshua B Tenenbaum. “MarrNet: 3D Shape Reconstruction via 2.5D Sketches”. In: *Advances In Neural Information Processing Systems*. 2017 (cit. on p. 12).

[Wu+18] Jiajun Wu, Chengkai Zhang, Xiuming Zhang, Zhoutong Zhang, William T Freeman, and Joshua B Tenenbaum. “Learning 3D shape priors for shape completion and reconstruction”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Vol. 3. 2018 (cit. on pp. 12, 80).

[XDZ17] Haotian Xu, Ming Dong, and Zichun Zhong. “Directionally Convolutional Networks for 3D Shape Segmentation”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2017, pp. 2698–2707 (cit. on p. 22).

[Yan+16] Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. “Perspective Transformer Nets: Learning Single-View 3D Object Reconstruction without 3D Supervision”. In: *Advances In Neural Information Processing Systems*. 2016, pp. 1696–1704 (cit. on p. 12).

[Yan+18] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. “FoldingNet: Point Cloud Auto-Encoder via Deep Grid Deformation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018 (cit. on p. 12).

[Yan+19] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. “PointFlow: 3D Point Cloud Generation with Continuous Normalizing Flows”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 4541–4550 (cit. on p. 80).

[Yan+20] Yuqi Yang, Shilin Liu, Hao Pan, Yang Liu, and Xin Tong. “PFCNN: Convolutional Neural Networks on 3D Surfaces Using Parallel Frames”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 13578–13587 (cit. on p. 21).

[Yan+21] Guandao Yang, Serge Belongie, Bharath Hariharan, and Vladlen Koltun. "Geometry Processing with Neural Fields". In: *Advances in Neural Information Processing Systems*. 2021 (cit. on pp. 13, 80).

[Ye+18] Zi Ye, Olga Diamanti, Chengcheng Tang, Leonidas Guibas, and Tim Hoffmann. "A unified discrete framework for intrinsic and extrinsic Dirac operators for geometry processing". In: *Computer Graphics Forum*. Vol. 37. 2018, pp. 93–106 (cit. on p. 18).

[Yi+16] Li Yi, Vladimir G. Kim, Duygu Ceylan, I.-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. "A Scalable Active Framework for Region Annotation in 3D Shape Collections". In: *ACM Transactions on Graphics* 35.6 (2016), pp. 1–12 (cit. on p. 70).

[Yi+17] Li Yi, Hao Su, Xingwen Guo, and Leonidas Guibas. "SpecSyncCNN: Synchronized spectral CNN for 3D shape segmentation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2282–2290 (cit. on p. 22).

[Yin+18] Kangxue Yin, Hui Huang, Daniel Cohen-Or, and Hao Zhang. "P2P-NET: Bidirectional Point Displacement Net for Shape Transform". In: *ACM Transactions on Graphics* 37.4 (2018), 152:1–152:13 (cit. on pp. 12, 80).

[Yu+20] Changqian Yu, Jingbo Wang, Changxin Gao, Gang Yu, Chunhua Shen, and Nong Sang. "Context Prior for Scene Segmentation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (cit. on p. 10).

[Zen+12] Wei Zeng, Ren Guo, Feng Luo, and Xianfeng Gu. "Discrete Heat Kernel Determines Discrete Riemannian Metric". In: *Graphical Models* 74.4 (2012), pp. 121–129 (cit. on p. 19).

[Zha+18] Xiuming Zhang, Zhoutong Zhang, Chengkai Zhang, Joshua B Tenenbaum, William T Freeman, and Jiajun Wu. "Learning to Reconstruct Shapes From Unseen Classes". In: *Advances in Neural Information Processing Systems*. 2018 (cit. on pp. 12, 80).

[Zha+20] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Zhi Zhang, Haibin Lin, Yue Sun, Tong He, Jonas Muller, R. Manmatha, Mu Li, and Alexander Smola. "ResNeSt: Split-Attention Networks". In: *arXiv:2004.08955* (2020) (cit. on p. 10).

[Zhu+17] Jun-Yan Zhu, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A Efros, Oliver Wang, and Eli Shechtman. "Toward Multimodal Image-to-Image Translation". In: *Advances in Neural Information Processing Systems*. 2017, pp. 465–476 (cit. on p. 53).