October 2022

Representation Learning for Shape Decomposition, By Shape Decomposition

Gopal Sharma
University of Massachusetts Amherst

REPRESENTATION LEARNING FOR SHAPE DECOMPOSITION, BY SHAPE DECOMPOSITION

A Dissertation Presented

by

GOPAL SHARMA

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2022

Robert and Donna Manning College of
Information and Computer Sciences

REPRESENTATION LEARNING FOR SHAPE DECOMPOSITION, BY SHAPE DECOMPOSITION

A Dissertation Presented

by

GOPAL SHARMA

Approved as to style and content by:

Evangelos Kalogerakis, Co-chair

Subhransu Maji, Co-chair

Rui Wang, Member

Siddhartha Chaudhuri, Outside Member

James Allan, Chair
Robert and Donna Manning College of
Information and Computer Sciences

DEDICATION

To all who have loved me in their hearts.

ACKNOWLEDGMENTS

Just like any endeavour in human history, this thesis has seen the light of day because of many helping hands. I have been incredibly fortunate to be born in a loving and stable household, with kind and supporting family and friends. More so, from the stability of the world and our society in general, that has fostered the development of artificial intelligence, which this thesis is concerned about. The enormous progress in medical science, social science and technology has enabled me to pursue dreams, of which this manuscript is a culmination.

This thesis would not have been possible without enormous support from my advisers Prof. Evangelos Kalogerakis and Prof. Subhransu Maji. I feel honoured to be guided by them and feel privileged to be part of their lab. I will forever be grateful to my advisers for bringing me into the lab and showing confidence in me. Just like a potter who makes a pottery by supporting and shaping a clay, my advisers gave me support throughout my PhD, while shaping my thoughts and making me a better researcher. I would also like to thank my committee members Prof. Rui Wang and Prof. Siddhartha Chaudhuri for reviewing this thesis and providing invaluable suggestions to improve it.

I had expected my time in Amherst to involve just attending school with strangers, instead, I found the experience closer to spending time with a second family away from home. I left home and found a new home here. This, of course, happened because of the friends and colleagues I met and collaborators I made. Other than my advisers, my senior lab mates have contributed much to my learning. Their eagerness to help in all difficult aspects of grad life has made the experience much more pleasant. Starting with my seniors, Aruni RoyChowdhury, Pia Bideau, SouYoung Jin, Tsung-Yu Lin and Matheus Gadelha, whose support and guidance I will forever be grateful. Furthermore, my lab mates Zhan Xu,

Yang Zhou, Zezhou Cheng, Pratheba Selvaraju and Dmitry Petrov have been kind to discuss research with me and support me in various academic and non-academic endeavours.

We humans create many inanimate objects on a regular basis, however some things become part of our regular lives that their importance can't be exaggerated. Sarek, my desktop has been a faithful companion, fulfilling demands of my sub-optimal codes ceaselessly, providing computational resources seemingly out of thin air. In the context of this thesis, Sarek's reliance on electricity can only be matched by my reliance on Sarek's computational powers. Sarek's constant whirring almost made Sarek human, a constant reminder to me of a presence of another nearby soul. Perhaps result of all the artificial intelligence code run for this thesis has made Sarek a bit *conscious*.

At CICS, I got unprecedented support from staff members. My academic advisers Leeanne Leclerc and Eileen Hamel have made the traversal of the rocky terrain of the grad school feels like a smooth road.

In my early childhood, I was fortunate to be taught by my science teacher Sarla Yadav, who made me realize that no question is a stupid question and there are several open questions in science that are worth exploring. Through her teaching, I realized that science is at least as cool as cricket. I owe my interest in science to her. Before joining UMass, I was advised by Prof. G.N Pillai and Prof. Bernard Ghanem. Their advice and support have lead me to pursue higher studies.

On a personal level, I am grateful to my parents for giving me free rein of my life and thus enabled me to pursue my interests. I am grateful to have lovely and loving sisters, whose unconditional love is a blessing. And my friends who taught me how to love, show loyalty and gratitude, and find the meaning of life.

Last, though still very close to my heart are the authors that I would like to acknowledge– J.K Rowling, J.R.R Tolkien, Robin Hobb, Patrick Rothfuss, Sir Arthur Conan Doyle, Andrej Sapkowsi, Kurt Vonnegut, Fyodor Dostoevsky and Scott Lynch, for showing a different

world when this world did not make sense. Their words banished the darkness from inaccessible places of my heart and always kept it at bay.

It is impossible to acknowledge everyone and everything here, so I would just like to acknowledge the happenstance that the infinite random perturbations of fundamental particles over eternity has resulted in this beautiful age.

ABSTRACT

REPRESENTATION LEARNING FOR SHAPE DECOMPOSITION, BY SHAPE DECOMPOSITION

SEPTEMBER 2022

GOPAL SHARMA

B.Tech, INDIAN INSTITUTE OF TECHNOLOGY, ROORKEE

M.Sc., UNIVERSITY OF MASSACHUSETTS, AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Evangelos Kalogerakis and Professor Subhransu Maji

The ability to parse 3D objects into their constituent parts is essential for humans to understand and interact with the surrounding world. Imparting this skill in machines is important for various computer graphics, computer vision, and robotics tasks. Machines endowed with this skill can better interact with its surroundings, perform shape editing, texturing, recomposing, tracking, and animation. In this thesis, we ask two questions. First, how can machines decompose 3D shapes into their fundamental parts? Second, does the ability to decompose the 3D shape into these parts help learn useful 3D shape representations?

In this thesis, we focus on parsing the shape into compact representations, such as parametric surface patches and Constructive Solid Geometry (CSG) primitives, which are also widely used representations in 3D modeling in computer graphics. Inspired by the advances in neural networks for 3D shape processing, we develop neural network approaches to tackle

shape decomposition. First, we present CSGNET, a network architecture to parse shapes into CSG programs, which is trained using combination of supervised and reinforcement learning. Second, we present PARSENET, a network architecture to decompose a shape into parametric surface patches (B-Spline) and geometric primitives (plane, cone, cylinder and sphere), trained on a large set of CAD models using supervised learning.

The training of deep neural network architectures for 3D recognition and generation tasks requires a large amount of labeled datasets. We explore ways to alleviate this problem by relying on shape decomposition methods to guide the learning process. Towards that end, we first study the use of freely available metadata, albeit inconsistent, from shape repositories to learn 3D shape features. Later we show that learning to decompose a 3D shape into geometric primitives also helps in learning shape representations useful for semantic segmentation tasks. Finally, since most 3D shapes encountered in real life are textured, consisting of several fine-grained semantic parts, we propose a method to learn fine-grained representations for textured 3D shapes in a self-supervised manner by incorporating 3D geometric priors.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

The ability to parse 3D objects into its constituent parts is essential for humans to understand and interact with the surrounding world [18, 94]. Imparting this skill into machines is important for various computer graphics, computer vision and robotics tasks. Machines endowed with this skill can better interact with its surrounding, perform shape editing [110], texturing, recomposing [109], tracking [37] and animation.

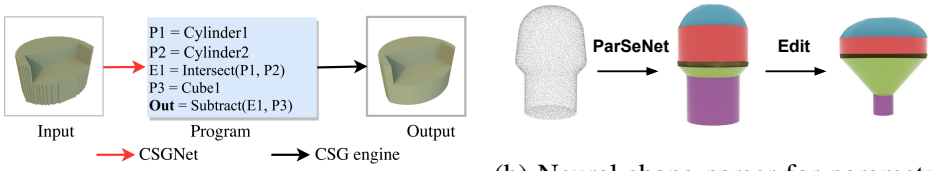
This thesis is mainly concerned with two topics: *3D shape decomposition* and *learning 3D shape representations*. Specifically, we study various ways in which 3D shapes can be decomposed into semantic parts, shape programs and geometric primitives. Further, we study ways to induce 3D shape representations that alleviate the need for a large-scale labelled dataset for shape decomposition.

With the availability of 3D sensors [263] and 3D shape repositories [27, 124], there is an abundance of 3D models. These models often come in the form of polygon meshes when they are modeled by artists, or point clouds when they are acquired from the real-world. In the case of point clouds, it is highly desirable to reconstruct them into continuous surfaces that can be further edited and manipulated by users. Similarly, polygon meshes are cumbersome to edit especially with low-level geometry operations (e.g., moving vertices, inserting polygons *etc*.). We are interested in approximating the shape (individual object) with compact representations that are used by the expert in graphics modeling packages, and can be easily used by non-expert users to manipulate the shape. For example, in the case of 2D, in vector graphics modeling packages, shapes are often created through higher-level primitives, such as parametric curves (e.g., Bezier curves) or basic shapes

(circles, polygons *etc*.), as well as operations acting on these primitives, such as boolean operations, deformations, extrusions, and so on. Furthermore in the case of 3D, observations from the computer-aided design and modeling literature suggest that designers often model shapes by constructing several non-overlapping primitives (cone, cylinder, B-spline patches *etc*.) placed seamlessly. In this thesis, we are interested in the question–*how to design deep neural network architectures for the task of shape-decomposition?* To this end, we propose deep neural networks that can map a 3D object represented using point cloud or meshes to structured representation such as programs (constructive solid geometry [131]) and parametric surface patches (B-spline) and 3D geometric primitives (cones, planes, spheres and cylinder).

Recently, several neural network architectures have been proposed to decompose shapes represented using point clouds [178], meshes [86], and voxels [183] into semantic parts. Yet, these architectures for shape decomposition are limited by the ability to collect labeled training data, which is often expensive or time consuming. We explore ways to alleviate this problem by relying on shape decomposition methods to guide the learning process within *self-supervision* regime. In this thesis, we are interested in the question–*does learning to decompose 3D shapes help in semantic segmentation tasks?* In order to answer this question, we first study the use of freely available metadata, albeit inconsistent, from shape repositories to learn 3D shape features. Later we show that learning to decompose a 3D shape into geometric primitives also helps in learning shape representations useful for semantic segmentation tasks.

Finally, since most 3D shapes encountered in real-life are textured consisting of several fine-grained semantic parts, we propose a method to learn fine-grained representations for textured 3D shapes in a self-supervised manner. We propose an approach that uses neural-network to produce view-invariant representation of 3D shapes, outperforming various state-of-the-art self-supervision methods on few-shot semantic segmentation tasks.



(a) Neural shape parser for CSG.

(b) Neural shape parser for parametric surfaces.

Figure 1.1: **Learning to decompose 3D shapes.** a) CSGNet is a neural shape parser that produces a compact program for input 3D shape. b) ParSeNet decomposes point clouds into collections of assembled parametric surface patches including B-spline patches. Predicted shape can be edited using the inferred parametrization.

To conclude, in this thesis I first propose two techniques that produce robust decomposition and better reconstruction of shapes using several evaluation metrics in comparison to both analytical and learning based baselines. Secondly, we propose three shape representation learning techniques applied directly on point clouds that produce state-of-the-art performance on shape segmentation while utilizing small amounts of labeled data and large amounts of unlabeled/weakly labeled data. Below we introduce above-mentioned topics in more detail.

1.1 Learning to decompose 3D shapes

Our goal is to describe shapes with higher-level primitives and operations which is highly desirable for designers since it is compact and makes subsequent editing easier. It may also better capture certain aspects of human shape perception such as view invariance, compositionality, and symmetry. With this goal in mind, in this thesis we explore two directions for shape parsing– a) parsing 2D and 3D shape into CSG programs (Figure 1.1a) and b) parsing 3D point cloud into parametric surface patches (Figure 1.1b).

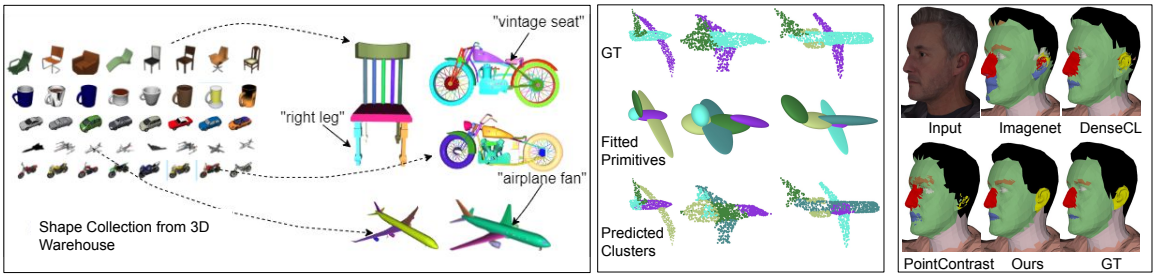
In Chapter 3, we develop an algorithm that parses an input image or a 3D object into their constituent modeling primitives and operations within the framework of Constructive Solid Geometry (CSG). CSG is a popular geometric modeling framework where shapes are generated by recursively applying boolean operations, such as union or intersection, on

simple geometric primitives, such as spheres or cylinders. Parsing a shape into its CSG program poses a number of challenges. First, the number of primitives and operations is not the same for all shapes. Second, the order of these instructions matter; small changes in the order of operations can significantly change the generated shape. Third, the number of possible programs grows exponentially with the program length, making learning and inference challenging.

Existing approaches for CSG parsing are predominantly search-based [58, 83, 242]. A significant portion of the related literature has focused on approaches to efficiently estimate primitives in a bottom-up manner, and to search for their combinations using heuristic optimization. While these techniques can generate complex shapes, they are prone to noise in the input and are generally slow. Our contribution is a neural network architecture called CSGNet that generates the program in a feed-forward manner for an input 2D image or a 3D object. CSGNet is efficient at the test time, as it can be viewed as an amortized search procedure. Furthermore, it can be used as an initialization for search-based approaches leading to improvements in accuracy at the cost of computation.

In Chapter 4, we aim to automate the time-consuming process of converting a 3D object represented in a point cloud format into a piecewise parametric surface representation. An important question that we answer is how surface patches should be represented. Patch representations in CAD and graphics are based on well-accepted geometric properties: (a) continuity in their tangents, normals, and curvature, making patches appear smooth, (b) editability, such that they can easily be modified based on a few intuitive degrees of freedom (DoFs), e.g., control points or axes, and (c) flexibility, so that a wide variety of surface geometries can be captured.

A variety of analytical (*i.e.* not learning-based) algorithms [52, 96, 128, 192] have been devised to approximate raw 3D data as a collection of geometric primitives and B-spline surface patches: dominant themes include Hough transforms, RANSAC [192] and clustering. We propose a learning-based approach that takes advantage of a large-



a. Learning representation from hierarchy and tags.

b. Learning representation from primitive fitting

c. Learning representation for fine-grained semantic segmentation

Figure 1.2: **Learning representations for 3D shapes.** a) We propose to utilize metadata such as polygon groupings and tags assigned to parts present in shape collections from 3D shape repositories to learning shape representations, b) SURFIT uses primitive fitting as a self-supervised task for learning 3D shape representations, c) MVDECOR produces fine-grained segmenatic segmentation for 3D shapes outperforming several SSL baselines.

scale CAD dataset consisting of single objects [124] for surface fitting task. We propose PARSENET, a parametric surface fitting network architecture which produces a compact, editable representation of a point cloud as an assembly of geometric primitives, including open or closed B-spline patches.

PARSENET is trained using ABC dataset, which provides meshes with rich annotations of surface patches. Utilizing this dataset, PARSENET models a richer class of surfaces than prior work, which only handles basic geometric primitives such as planes, cuboids and cylinders [139, 192]. PARSENET includes a novel neural network (SPLINENET) to estimate an open or closed B-spline model of a point cloud patch that provides richness and flexibility highly desired in shape design. Our contribution is an end-to-end, differentiable pipeline that decomposes the point cloud into segments and fits parametric surface patches to each segment giving state-of-the-art performance on parametric surface fitting task.

Our approach mainly works on inputs with single objects in comparison to scenes [112]. To extend this work to scenes requires development of dataset consisting of scenes annotated with primitives and a learning-based hierarchical approach to primitive fitting.

1.2 Learning representations for 3D shape understanding

3D recognition tasks such as shape recognition and shape segmentation require large amounts of annotated dataset, which incurs large manual labor and cost. This motivates the need of label-efficient learning approaches (as shown in Figure 1.2) which can use freely available data to learn shape representations. In this thesis, we explore three label-efficient approaches to learn 3D shape representation useful for down-stream tasks– 1) using inconsistent part hierarchies available in 3D shape repositories to pre-train the neural network, 2) using primitive fitting under self-supervision regime to train the neural network and 3) using multi-view correspondence learning to train neural network to produce fine-grained features for 3D shapes.

In Chapter 5, we utilize the metadata associated with shapes (individual objects) from online (3D Warehouse) repository. This metadata consists of information about geometric primitives (e.g., polygons in 3D meshes) organized in groups, often arranged in hierarchy, as well as color, material and semantic tags assigned to them. These metadata originates from the modeling decisions of designers, which are likely to be correlated with high-level semantics. These metadata have a high level of variability which is a consequence of variability in goals and expertise of designers, as shown in Figure 1.3. Our approach consists of a deep network that maps each point in a 3D shape to a fixed dimensional embedding. The network is trained in a way such that the embedding reflects the user-provided hierarchy and tags. We propose a robust tree-aware metric to supervise the point embedding network that offers better generalization to semantic segmentation tasks over a baseline scheme that is tree-agnostic (only considers the leaf-level groupings). The point embedding network trained on hierarchies also improves over models trained on shape reconstruction tasks that leverage the 3D shape geometry but not their metadata. Finally, when tags are available, we show that the embeddings can be fine-tuned leading to further improvements in performance. Our approach produces object-level representations instead of scene-level representations because the 3D Warehouse dataset contains models of individual objects only. Theoretically

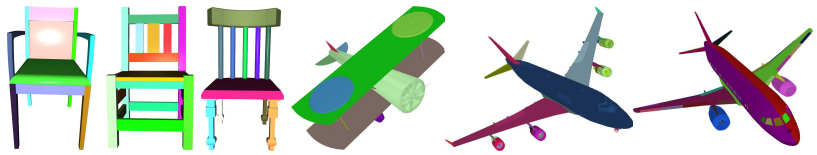


Figure 1.3: **Inconsistent shape decomposition.** Our dataset consist of shapes segmented into parts without any semantic information. Notice that shapes of same category can be segmented differently from each other. Here different color represents different leaf node in the part-hierarchy.

our approach can also work on scenes if object-level and part-level hierarchy information is provided.

In Chapter 6, we present a self-supervised approach for learning 3D shape representations by decomposing the surface of a 3D object into geometric primitives. It improves part segmentation models when learning from a few labeled examples. Our approach exploits the fact that parts of 3D objects are often aligned with simple geometric primitives, such as ellipsoids and cuboids, as shown in the Figure 1.4. Even though these primitives capture 3D objects at a rather coarse level, the induced partitions provide a strong prior for learning part segmentation networks. This purely geometric task allows us to utilize vast amounts of unlabeled data in existing 3D shape repositories to guide representation learning for part segmentation, which is especially useful in the few-shot setting. The primitive fitting module follows a novel iterative clustering and primitive parameter estimation scheme based on the obtained per-point embeddings. It is fully differentiable, thus, the whole architecture can be trained end-to-end. The self-supervised objective minimizes a reconstruction loss, computed as the Chamfer distance between the 3D surface and the collection of fitted primitives.

Annotating 3D shapes for fine-grained semantic segmentation is often done using 2D projections of the shape to avoid the need for 3D manipulation operations. This is especially true for shapes that lack structure, such as primitives, that can be easily selected in 3D. Furthermore, 2D CNNs are better at modelling high-resolution details and texture, compared to their 3D counterparts. Keeping these observations in mind, in Chapter 8 we propose MvDeCoR [205], a self-supervised technique for learning dense 3D shape representations

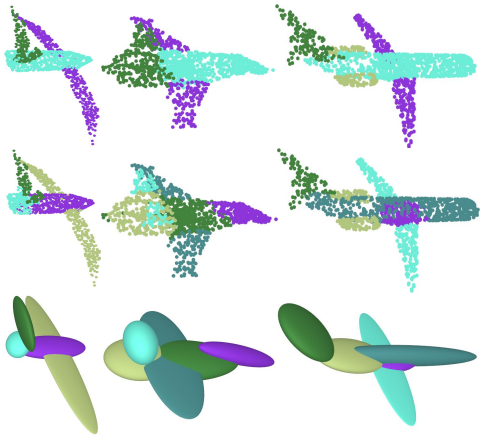


Figure 1.4: **Primitive fitting as a self-supervised task for learning 3D shape representations.** *Top row:* 3D shapes represented as point clouds, where the color indicates the parts such as wings and engines. The induced partitions and shape reconstruction obtained by fitting ellipsoids to each shape using our approach are shown in the *middle row* and *bottom row* respectively. The induced partitions have a significant overlap with semantic parts.

based on the task of learning correspondences across views of a 3D shape. At training time we render 3D shapes from multiple views with known correspondences and setup a contrastive learning task to train 2D CNNs. The learned 2D representations can be directly used for part segmentation on images, or projected onto the shape surface to produce a 3D representation for 3D tasks. The approach works well in standard few-shot fine-grained 3D part segmentation benchmarks, outperforming prior work based on 2D and 3D self-supervised learning. Though, our approach produces representation of single objects, but it can also be applied on multiple objects and scenes if the pixel-level correspondence is provided.

1.3 Summary of Publications

The list of works that are part of this thesis:

- The content of Chapter 3:
  - [200] Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. Neural shape parsers for constructive solid geometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020).

- [201] Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. CSGNet: Neural shape parser for constructive solid geometry. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018).

- The content of Chapter 4:

  - [204] Gopal Sharma, Difan Liu, Subhransu Maji, Evangelos Kalogerakis, Siddhartha Chaudhuri, and Radomír Měch. PARSENET: A parametric surface fitting network for 3d point clouds. In *European Conference on Computer Vision (ECCV)* (2020).

- The content of Chapter 5:

  - [203] Gopal Sharma, Evangelos Kalogerakis, and Subhransu Maji. Learning point embeddings from shape repositories for few-shot segmentation. In *International Conference on 3D Vision (3DV)* (2019).

- The content of Chapter 6:

  - Gopal Sharma, Bidya Dash, Matheus Gadelha, Aruni RoyChowdhury, Marios Loizou, Evangelos Kalogerakis, Liangliang Cao, Erik Learned-Miller, Rui Wang, and Subhransu Maji. SURFIT: Learning to Fit Surfaces Improves Few Shot Learning on Point Clouds. Under submission (2022).

- The content of Chapter 8:

  - Gopal Sharma, Kangxue Yin, Or Litany, Evangelos Kalogerakis, Subhransu Maji, and Sanja Fidler. MVDECOR: Multi-view dense correspondence learning for fine- grained 3d segmentation. Under submission (2022).

The list of publications I co-authored and are not part of this thesis:

- [64] Matheus Gadelha, Aruni RoyChowdhury, Gopal Sharma, Evangelos Kalogerakis, Liangliang Cao, Erik Learned-Miller, Rui Wang, and Maji Subhransu. Label-efficient

learning on point clouds using approximate convex decompositions. In *European Conference on Computer Vision (ECCV)* (2020).

- [188] Amirmohammad Rooshenas, Dongxu Zhang, Gopal Sharma, and Andrew McCallum. Search-Guided, Lightly-Supervised Training of Structured Prediction Energy Networks. *Advances in Neural Information Processing Systems* (2019).

CHAPTER 2

BACKGROUND

In this Chapter we provide background information on topics such as representation learning for 3D and shape decomposition.

2.1 Representation Learning for 3D

In the past, researchers have developed several shape descriptors for 3D object recognition problems, that find their applications in computer vision and computer graphics. These shape descriptors work directly on shapes represented using point cloud, polygonal meshes, voxel-grids or implicit surfaces. These descriptors are commonly hand-designed based on surface properties. For example, a shape can be represented with histograms or bag-of-words model using surface properties such as normals and curvatures [97], triangle areas, local shape diameters [30], heat kernel signatures on polygonal meshes [23], or extensions of SIFT and SURF features descriptors to 3D voxel grids [122], *etc*. These descriptors are largely "hand-engineered" and require extensive hyper-parameter tuning. Furthermore, these descriptors are not rich enough to capture all characteristics of the shape.

Modern approaches for representation learning rely on *deep learning*. Deep learning is a class of machine learning algorithms that enables computers to build abstract concepts from simpler ones. This is done using composition of functions, also called *layers*, which extract progressively abstract features from raw inputs. Convolutional neural network (CNN) [135] is a popular class of networks with diverse applications. CNN is a specialized kind of neural network for processing data that has a known grid-like topology and employ convolution as an operation in at least one of the layers of the network. A simple example of representation

learning is training a feed-forward neural network for classification task, *e.g.* a CNN for object recognition task. In this, an image is input to the CNN which extracts increasing abstract features of the input using several convolutional layers with non-linearity and the last layer being a linear classifier. The rest of the network learns to provide useful features for this classifier layer.

In recent years, 2D CNNs have been used for tasks related to image recognition [90, 129, 207] and generation [180] by training on large scale image datasets [45]. 2D CNNs have also found their utility in video recognition [9] and generation tasks [142]. 3D CNNs have been used to learn representations for video that incorporate temporal dimension [111].

Several deep neural networks have been proposed to learn representations for 3D shapes. Mesh CNNs are proposed for 3D data represented using polygonal meshes [85,156,176,197]. VoxelNet [42, 220] was proposed for shapes represented using voxel-grid. To improve the computation speed and memory requirements, OctNet [183] was proposed to use the Octree structure to utilize the sparsity of the voxel representation. Taking inspiration from works on point cloud learning architectures (PointNet [213] and PointNet++ [179]) to process point clouds, several works have proposed doing convolutions over the graphs constructed over point clouds [134, 239, 264]. Choy *et al.* [38] proposed 4-dimensional convolutional neural networks for spatio-temporal perception that can directly process such 3D-videos using high-dimensional convolutions. They adopted sparse tensors and propose the generalized sparse convolution which encompasses all discrete convolutions. Multi-view approaches [117, 212] have also been explored for shape classification and segmentation. To train these models for learning representation of 3D shapes, several small and large-scale dataset have been proposed–ModelNet40, ShapeNet [27], ScanNet [42] , S3DIS [14], SemanticKITTI [16], ABC [124] *etc*.

2.1.1 Self-supervised learning

Many tasks in computer vision and computer graphics have been solved with the help of supervised learning. The supervised learning relies on large amount of labeled data, which is often time consuming and expensive process. For several applications collecting large amount of data is quite hard. To overcome these limitation, several works learn representation for images and 3D shapes under self-supervision paradigm.

With the availability of large amount of images on the internet and 3D shapes from online repositories, self-supervised learning has gained popularity and produced good results on several tasks like image classification, detection and segmentation, along with tasks like 3D shape classification, shape segmentation and scene segmentation. Self-supervised learning is a form of unsupervised learning where data itself provides the supervision. The idea is to withhold a part of the data and task the neural network to predict hidden information. This task is a 'proxy' task or 'pre-text' task, where the goal is not to solve this task perfectly but to learn representations useful for other down-stream tasks like classification, segmentation *etc*.

Below we briefly discuss several self-supervision tasks.

2.1.1.0.1 Self-supervision in images. In the past, researchers have used auto-encoder to learn efficient codings of the unlabeled dataset. The autoencoder learns representation of the input image for dimensionality reduction by training the network to ignore noise in the data. Hinton *et al.* [92] used greedy layer-wise unsupervised pretraining as an initialization of deep autoencoders to learn representation for images.

With the abundance of colored images freely available from the internet, several works [132, 229] have proposed a simple colorization as a self supervision task. Given a gray-scale image, the task is to predict colored version of the image, where output colors are represented in quantized CIE Lab color space.

Ideally, one expects that the small transformations in the image does not change the semantic meaning of the image, hence the representations produced by the neural network

should also remain same under these transformations. Following this observation, several works [32, 89, 248] use contrasting learning to encourage similarity between representations of an image under different transformations. Wu *et al.* [248] setup a non-parametric classification problem at the instance level to learn image level features such that visually similar instances of objects also have similar embeddings. He *et al.* [89] build a dynamic dictionary with a queue and a moving-averaged encoder enabling large and consistent dictionary on-the-fly to generate negative instances for effective contrastive learning. BYOL [77] achieves better results without using negative samples with the help of Batch-normalization that avoid collapse of the instance embeddings.

If we geometrically transform the object in an image, its representation should also change in the same way. This observation is utilized in several works for self supervision [49, 72, 165]. Gidaris *et al.* [72] proposed a rotation prediction of the image as a pre-text task. In this, the input image is rotated randomly to $\{0, 90, 180, 270\}$ degrees, and the task for the network is to predict the correct rotation. Doersch *et al.* [49] predict the relative location of a patch w.r.t another patch, both sampled from the same image. To correctly solve this, the network has to learn the spatial context of the object.

Recently, He *et al.* [88] proposed masked-autoencoder based on transformer architecture that define reconstructing the masked input image as a self-supervision task. Their approach outperforms several supervised and self-supervised baselines.

2.1.1.0.2 Self-supervision tasks for videos. Videos follow the arrow of time, *i.e*. they are in chronological order. Several works [157, 237] have proposed using this property to learning useful representation for videos that captures low level physics like direction of gravity (things fall when dropped from a height), nature of living beings (human walk while facing front), causality (egg can be broken but vice-versa is not true) *etc*. Mishra *et al.* [157] proposed using temporal consistency as a self supervision task which improves the performance of down-stream action recognition task. Tracking is also used for self-supervision [237], where two patches from the same tracklet are labeled positive patches

and two unrelated patches are labeled negative patches under triplet loss regime to train the network. Representations learned in this manner are helpful for downstream tasks. The idea of colorization has also been extended to videos [229], where the task is to copy color from a reference frame (colored) to target frame (gray-scale). To solve this task, the network needs to keep track of correlated pixels, thus discriminating static from dynamic regions of the image.

2.1.1.0.3 Self-supervision for 3D shapes. Similarly, self supervised learning tasks are proposed for 3D representation learning for objects and scenes. For detailed review of the literature, please refer to the Chapter 6 and 8.

2.2 3D shape Decomposition

There is a large body of work dealing with shape decomposition done using both analytic and learning-based techniques. Below we briefly describe these works.

2.2.1 Low-level and mid-level features

Local surface properties such as normals and principal curvatures induce a partial local structures that helps in solving several tasks such as edge feature detection [79, 241], segmentation, classification and surface reconstruction [104, 166]. Classical approaches that extract local surface features such as normals and curvatures have relied on fitting n-jet surfaces to point clouds with the help of least squares fitting [26]. Ridge-valley lines, curves defined via first and second-order curvature derivatives, are powerful shape descriptors. The estimation of ridge-valley structures have been explored for shape analysis, face recognition [81] and shape segmentation. Ohtake *et al.* [167] proposed combining multi-level implicit surface fitting and finite difference approximations to get ridge-valley lines on meshes. Kalogerakis *et al.* [118] proposed a robust framework for extracting lines of curvatures on a noisy point cloud using robust statistical estimates of surface normal and curvature.

Several deep-learning based approaches have shown good performance in local feature estimation, such as normals and curvature for surface reconstruction [17]. EC-Net [256] learns edges on point clouds using a deep-network that operates on patches of point clouds. Loizou *et al.* [144] detects boundaries of parts in 3D shapes represented as point clouds with the help of a graph neural network. Wang *et al.* [236] proposed PIE-Net a learnable technique to identify feature edges in 3D point cloud data. These edges are further inferred as a collection of parametric curves (*i.e.*, lines, circles, and B-splines).

A survey of classical approaches to segmentation of 3D shapes is provided by Chen *et al.* [33]. We also provide reviews of recent works on segmentation of 3D shapes in the Chapters 5, 6 and 8.

2.2.2 Shape parsing

A variety of analytical (i.e. not learning-based) as well as learning-based algorithms have been devised to approximate raw 3D data as a collection of geometric primitives. Details of these approaches are discussed in Chapter 3 and Chapter 4. Here, we will briefly discuss recent development in this area.

Recently, several works have improved object and scene parsing. Paschalidou *et al.* [171] propose 3D primitive representation (called neural-parts) to approximate a shape that defines primitives using invertible neural networks. This leads to parsimonious representation of a shape using primitives that are more aligned with semantic parts of a 3D shape. Kawana *et al.* [119] proposed neural star domain that learns primitive shapes in the star domain, where each primitive is defined using a few parameters allowing intuitive shape editing. Both neural-parts and neural start-domain allows unsupervised training of the neural network using both implicit and explicit representation of shape leading to better reconstruction of shapes in comparison to just using one representation. Yu *et al.* [255] proposed CAPRI-Net that parses an input 3D shape into a compact assembly of quadric surface primitives via constructive solid geometry (CSG) operations in a completely unsupervised fashion.

Recently, several works have focused on extending the idea of primitive fitting to high resolutions point clouds. In the case of high-resolution point cloud scans, the challenge is to detect large as well as small primitives. Le *et al.* [133] proposed Cascaded Primitive Fitting Networks (CPFN) that relies on an adaptive patch sampling network to assemble detection results of global and local primitive detection networks. Huang *et al.* [106] proposed an approach for primitive instance segmentation under high resolution point clouds (both objects and scenes) by transforming the global segmentation into easier local tasks. They use adversarial network to decide whether two points belong to the same primitive. Finally during test time, a region growing method is used to segmentation the entire point cloud.

CHAPTER 3

SHAPE PARSERS FOR CONSTRUCTIVE SOLID GEOMETRY

3.1 Introduction

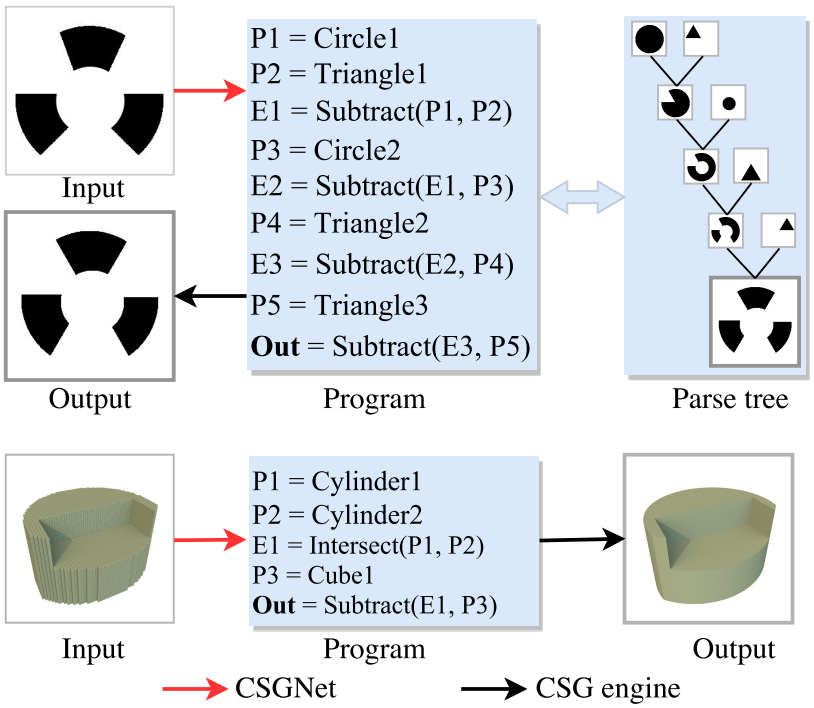


Figure 3.1: **Our shape parser produces a program that generates an input 2D or 3D shape**. On top is an input image of 2D shape, its program and the underlying parse tree where primitives are combined with boolean operations. On the bottom is an input voxelized 3D shape, the induced program, and the resulting shape from its execution.

The goal of our work is to develop an algorithm that parses shapes into their constituent modeling primitives and operations within the framework of Constructive Solid Geometry (CSG) [131]. CSG is a popular geometric modeling framework where shapes are generated by recursively applying boolean operations, such as union or intersection, on simple geometric primitives, such as spheres or cylinders. Figure 7.1 illustrates an example where

a 2D shape (top) and a 3D shape (bottom) are generated as a sequence of operations over primitives or a *visual program*. Yet, parsing a shape into its CSG program poses a number of challenges. First, the number of primitives and operations is not the same for all shapes *i.e.*, our output does not have constant dimensionality, as in the case of pixel arrays, voxel grids, or fixed point sets. Second, the order of these instructions matter — small changes in the order of operations can significantly change the generated shape. Third, the number of possible programs grows exponentially with the program length, making learning and inference challenging.

Existing approaches for CSG parsing are predominantly search-based. A significant portion of related literature has focused on approaches to efficiently estimate primitives in a bottom-up manner, and to search for their combinations using heuristic optimization. While these techniques can generate complex shapes, they are prone to noise in the input and are generally slow. Our contribution is a neural network architecture called CSGNET that generates the program in a feed-forward manner. The approach is inspired by the ability of deep networks for generative sequence modeling such as for speech and language. As a result CSGNET is efficient at test time, as it can be viewed as an *amortized search* [71] procedure. Furthermore, it be used as an initialization for search-based approaches leading to improvements in accuracy at the cost of computation.

At a high-level, CSGNET is an encoder-decoder architecture that encodes the input shape using a convolutional network and decodes it into a sequence of instructions using a recurrent network (Figure 7.2). It is trained on a large synthetic dataset of automatically generated 2D and 3D programs (Table 3.2). However, this leads to poor generalization when applied to new domains. To adapt models to new domains without program annotations, we employ policy gradient techniques from the reinforcement learning literature [245]. Combining the parser with a CSG rendering engine allows the networks to receive feedback based on the visual difference between the input and generated shape, and the parser is trained to minimize this difference (Figure 7.2). Furthermore, we investigate two network architectures: a vanilla

recurrent network (CSGNET), and a new variant called CSGNETSTACK (Figure 3.3). This new variant stores intermediate shapes produced during the execution of the CSG program, inspired by call or execution stacks [48]. This stack can also be seen as a form of explicit memory in our network encoding the intermediate program state. Our experiments demonstrate that this improves the overall accuracy of the generated programs while using less training data.

We evaluate the CSGNET and CSGNETSTACK architectures on a number of shape parsing tasks. Both offer consistently better performance than a nearest-neighbor baseline and are significantly more efficient than an optimization based approach. Reinforcement learning improves their performance when applying them to new domains without requiring ground-truth program annotations making the approach more practical (Table 3.4). We also investigate the effect of the training data size and reward choices used in the policy gradient algorithm [163] on the performance of the parser. Finally, we evaluate the performance on the task of primitive detection and compare it with a Faster R-CNN detector [182] trained on the same dataset. CSGNET offers 4.2% higher Mean Average Precision (MAP) and is $4\times$ faster compared to the Faster R-CNN detector, suggesting that joint reasoning about the presence and ordering of objects leads to better performance for object detection (Table 3.6).

This paper extends our work that first appeared in [200], adding to it an analysis on effect of reward shaping and training set size on the performance, as well as the stack-augmented network architecture. Our PyTorch [4] implementation is publicly available at: `https://hippogriff.github.io/CSGNet/`.

3.2 Related Work

CSG parsing has a long history and a number of approaches have been proposed in the literature over the past 20 years. Much of the earlier work can be categorized as "bottom-up" and focuses on the problem of converting a boundary representation (b-Rep) of the shape to a CSG program. Our work is more related to program generation approaches using neural

networks which have recently seen a revival in the context of natural language, graphics, and visual reasoning tasks. We briefly summarize prior work below.

3.2.1 Bottom-up shape parsing

An early example of a grammar-based shape parsing approach is the "pictorial structure" model [60]. It uses a tree-structured grammar to represent articulated objects and has been applied to parsing and detecting humans and other categories [22, 59, 252]. However, the parse trees are often shallow and these methods rely on accurate bottom-up proposals to guide parsing (*e.g.*, face and upper-body detection for humans). In contrast, primitive detection for CSG parsing is challenging as shapes change significantly when boolean operations are applied to them. Approaches, such as [24, 198, 199], assume an exact boundary representation of primitives which is challenging to estimate from noisy or low-resolution shapes. This combined with the fact that parse trees for CSG can be significantly deeper makes bottom-up parsing error prone. Evolutionary approaches have also been investigated for optimizing CSG trees [58, 84, 242], however, they are computationally expensive.

Thus, recent work has focused on reducing the complexity of search. Tao *et al.* [50] directly operates on input meshes, and converts the mixed domain of CSG trees (discrete operations and continuous primitive locations) to a discrete domain that is suitable for boolean satisfiability (SAT) based program synthesizers. This is different from our approach which uses a neural network to generate programs without relying on an external optimizer.

3.2.2 Inverse procedural modeling

A popular approach to generate 3D shapes and scenes is to infer context-free, often probabilistic "shape grammars" from a small set of exemplars, then sample grammar derivations to create new shapes [185,210,218,228]. This approach called Inverse Procedural Modeling (IPM) has also been used in analysis-by-synthesis image parsing frameworks [152, 221, 257].

Recent approaches employ CNNs to infer parameters of objects [130] or whole scenes [187] to aid procedural modeling. A similar trend is observed in graphics applications where CNNs are used to map input images or partial shapes to procedural model parameters [103, 164, 186]. Wu *et al.* [246] detect objects in scenes by employing a network for producing object proposals and a network that predicts whether there is an object in a proposed segment, along with various object attributes. Eslami *et al.* [54] use a recurrent neural network to attend to one object at a time in a scene, and learn to use an appropriate number of inference steps to recover object counts, identities and poses.

Our goal is fundamentally different: given a generic grammar describing 2D or 3D modeling instructions and a target image or shape, our method infers a derivation, or more specifically a modeling program, that describes it. The underlying grammar for CSG is quite generic compared to specialized shape grammars. It can model shapes in several different classes and domains (*e.g.*, furniture, logos, *etc*.).

3.2.3 Neural program induction

Our approach is inspired by recent work in using neural networks to infer programs expressed in some high-level language, *e.g.*, to answer question involving complex arithmetic, logical, or semantic parsing operations [15,46,114,116,140,162,181,259,260]. Approaches, such as [100, 113], produce programs composed of functions that perform compositional reasoning on an image using an execution engine consisting of neural modules [12]. Similarly, our method produces a program consisting of shape modeling instructions to match a target image by incorporating a shape renderer.

Other related work include the recent work by Tian *et al.* [224], which proposes a program induction architecture for 3D shape modeling. Here programs contain a variety of primitives and symmetries are incorporated with loops. While this is effective for categories such as chairs, the lack of boolean operations is limiting. A more complex approach is that of Ellis *et al.* [53], who synthesize hand-drawn shapes by combining (lines, circles,

rectangles) into Latex programs. Program synthesis is posed as a constraint satisfaction problem which is computationally expensive and can take hours to solve. In contrast, our feed-forward model that takes a fraction of a second to generate a program.

3.2.4 Primitive fitting

Deep networks have recently been applied to a wide range of primitive fitting problems for 2D and 3D shapes. Tulsiani *et al.* [225] proposed a volumetric CNN that predicts a fixed number of cuboidal primitives to describe an input 3D shape. Zou *et al.* [268] proposed an LSTM-based architecture to predict a variable number of boxes given input depth images. Li *et al.* [138] introduced a point cloud based primitive fitting network where shapes are represented as an union of primitives. Paschalidou *et al.* [172] uses superquadrics instead of traditional cuboids. Genova *et al.* [69] proposed a network that predicts local implicit functions decomposing the input shape into 3D Gaussian blobs. Huang *et al.* [105] decompose an image by detecting primitives and arranging them into layers. Gao *et al.* [67] train deep network to produce control points for splines using input images and point cloud. Recent networks such as BSP-Net [34] and CvxNet [43] are built on the concept of binary space partitioning to produce a collection of convexes that approximates the input point cloud or an image. Deprelle *et al.* [47] proposed representing shapes as the combination of learned deformable elementary 3D structures. The above approaches are trained to minimize reconstruction error like ours. On the other hand, they focus on predicting primitives, while our method also learns modeling operations (CSG) on them.

3.3 Designing a Neural Shape Parser

In this section, we first present a neural shape parser, called CSGNET, that induces programs based on a CSG grammar given only 2D/3D shapes as input. We also present another shape parser variant, called CSGNETSTACK, which incorporates a stack as a form of explicit memory and results in improved accuracy and faster training. We show that

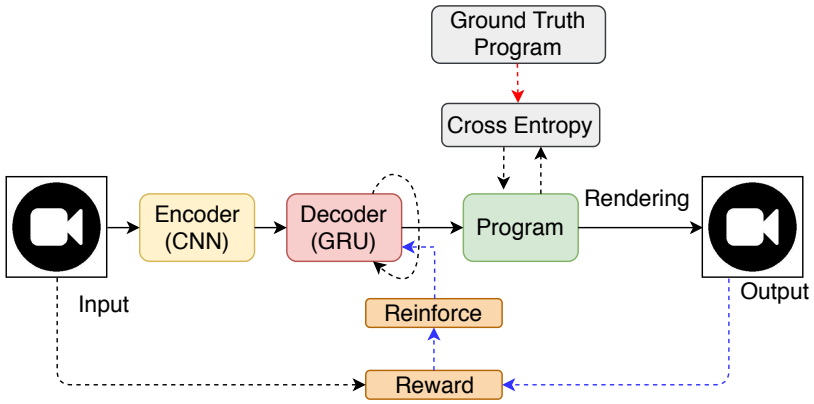


Figure 3.2: **Overview of our approach.** Our neural shape parser consists of two parts: first at every time step encoder takes as input a target shape (2D or 3D) and outputs a feature vector through CNN. Second, a decoder maps these features to a sequence of modeling instructions yielding a visual program. The rendering engine processes the program and outputs the final shape. The training signal can either come from ground truth programs when such are available, or in the form of rewards after rendering the predicted programs.

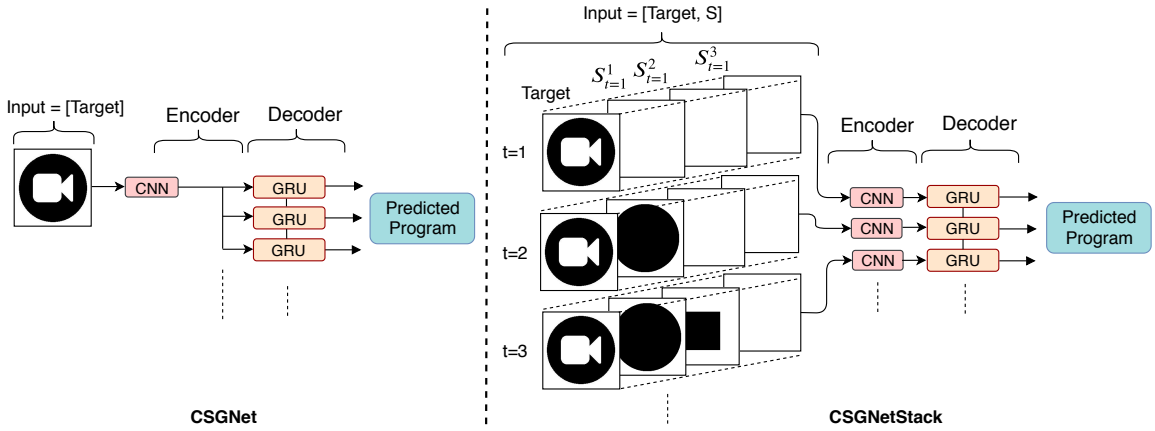


Figure 3.3: **Two proposed architectures of our neural shape parser CSGNET (left), CSGNETSTACK (right)**. CSGNet takes the target shape as input and encodes it using a CNN, whereas in CSGNETSTACK, the target shape is concatenated with stack S_t along the channel dimension and passes as input to the CNN encoder at every time step. Empty entries in the stack are shown in white.

both variants can be trained to produce CSG programs in a supervised learning setting when ground-truth programs are available. When these are not available, we show that reinforcement learning can be used based on policy gradient and reward shaping techniques. Finally, we describe ways to improve the shape parsing at test time through a post-processing stage.

CSGNET. The goal of a **shape parser** π is to produce a sequence of instructions given an input shape. The parser can be implemented as an encoder-decoder using neural network modules as shown in Figure 7.2. The **encoder** takes as input an image I and produces an encoding $\Phi(I)$ using a CNN. The **decoder** Θ takes as input $\Phi(I)$ and produces a probability distribution over programs P represented as a sequence of instructions. Decoders can be implemented using Recurrent Neural Networks (RNNs). We employ Gated Recurrent Units (GRUs) [40] that have been widely used for sequence prediction tasks such as generating natural language and speech. The overall network can be written as $\pi(I) = \Theta \circ \Phi(I)$. We call this basic architecture as CSGNET (see also Figure 3.3, left).

CSGNETSTACK. The above architecture can further be improved by incorporating feedback from the renderer back to the network. More specifically, the encoder can be augmented with an execution stack that stores the result of the renderer at every time step along with the input shape. This enables the network to adapt to both current and previous rendered results. To accomplish this, our CSG rendering engine executes the program instructions produced by the decoder with the help of stack $S = \{s_t : t = 1, 2 \ldots\}$ at each time step t. The stack is updated after every instruction is executed and contains intermediate shapes produced by previous boolean operations or simply an initially drawn shape primitive. This stack of shapes is concatenated with the target shape, all stored as binary maps, along the channel dimension. The concatenated map is processed by the network at the next time step. Instead of taking all elements of the stack, which vary in number depending on the

generated program, we only take the top-K maps of the stack. Empty entries in the stack are represented as all-zero maps (see also Figure 3.3, right). At the first time step, the stack is empty, so all K maps are zero. While the stack contains complete information about the program execution at any point in time, it can grow arbitrarily deep. Keeping the top-K elements of the stack provides a way to trade-off the computational and memory requirements with the amount of information about the program execution.

In our implementation, the parser π takes $Z = [I, S]$ as input of size $64 \times 64 \times (K + 1)$ for $2D$ networks and $64 \times 64 \times 64 \times (K + 1)$ for $3D$ networks, where I is the input shape, S is the execution stack of the renderer, and K is the size of the stack. The number of channels is $(K + 1)$ since the target shape, also represented as 64^2 (or 64^3 in 3D), is concatenated with the stack. Details of the architecture are described in Section 3.4. Similarly to the basic CSGNET architecture, the encoder takes Z as input and yields a fixed length encoding $\Phi(Z)$, which is passed as input to the decoder Θ to produce a probability distribution over programs P. The stack-based network can be written as $\pi(Z) = \Theta \circ \Phi(Z)$. We call this stack based architecture CSGNETSTACK. The difference between the two architectures is illustrated in Figure 3.3.

Grammar. The space of programs can be efficiently described according to a context-free grammar [95]. A context-free grammar is a formal grammar when its production rules can be applied regardless of the context of its non-terminal symbols. For example, in constructive solid geometry the instructions consist of drawing primitives (eg, spheres, cubes, cylinders, etc) and performing boolean operations described as a grammar with the following production rules:

$$S \rightarrow E$$

$$E \rightarrow E\ E\ T \mid P$$

$$T \rightarrow \texttt{OP}_1 | \texttt{OP}_2 | \ldots | \texttt{OP}_m$$

$$P \rightarrow \texttt{SHAPE}_1 | \texttt{SHAPE}_2 | \ldots | \texttt{SHAPE}_n$$

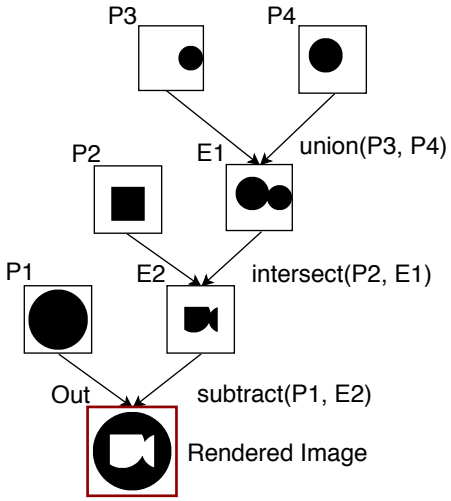
Each rule indicates possible derivations of a non-terminal symbol separated by the | symbol. Here S is the start symbol, $\texttt{OP}_i$ is chosen from a set of defined modeling operations and the $\texttt{SHAPE}_i$ is a primitive chosen from a set of basic shapes at different positions, scales, orientations, etc. Instructions can be written in a standard post-fix notation, *e.g.*, $\texttt{SHAPE}_1\texttt{SHAPE}_2\texttt{OP}_1\texttt{SHAPE}_3\texttt{OP}_2$, which can be written in in-fix notation as: $(\texttt{SHAPE}_1\ \texttt{OP}_1\ \texttt{SHAPE}_2)\ \texttt{OP}_2\ \texttt{SHAPE}_3$. Table 3.4 shows an example of a program predicted by the network and corresponding rendering process.

3.3.1 Learning

Given the input shape I and execution stack S of the renderer, the parser network π generates a program that minimizes a reconstruction error between the shape produced by executing the program and a target shape. Note that not all programs are valid. Our learning incorporates rewards promoting the generation of programs that are both valid and capture the target shape well.

3.3.1.1 Supervised learning

When target programs are available both CSGNET and CSGNETSTACK variants can be trained with standard supervised learning techniques. Training data consists of N shapes, P corresponding programs, and also in the case of CSGNETSTACK S stacks, program triplets (I^i, S^i, P^i), $i = 1, \ldots, N$. The ground-truth program P^i can be written as a sequence of instructions g_1^i, g_2^i .. $g_{T_i}^i$, where T_i is the length of the program P^i. Similarly, in the case of CSGNETSTACK, the S^i can be written as sequence of states of stack s_1^i, s_2^i .. $s_{T_i}^i$ used by the



Instruction	Execution	Stack
`circle(32,32,28)`	`push circle(32,32,28)`	`[P1]`
`square(32,40,24)`	`push square(32,40,24)`	`[P2 P1]`
`circle(48,32,12)`	`push circle(48,32,12)`	`[P3 P2 P1]`
`circle(24,32,16)`	`push circle(24,32,16)`	`[P4 P3 P2 P1]`
`union`	`A=pop; B=pop; push(B∪A)`	`[E1 P2 P1] // E1=P3∪P4`
`intersect`	`A=pop; B=pop; push(B∩A)`	`[E2 P1] // E2=P2∩E1`
`subtract`	`A=pop; B=pop; push(B−A)`	`[Out] // Out=P1−E2`

Figure 3.4: **Example program execution.** Each row in the table from the top shows the instructions, program execution, and the current state of the stack of the shift-reduce CSG parser. On the right is a graphical representation of the program. An instruction corresponding to a primitive leads to `push` operation on the stack, while an operator instruction results in popping the top two elements of the stack and pushing the result of applying this operator.

rendering engine while executing the instructions in program P^i. Note that while training in supervised setting, the stack s_t is generated by the renderer while executing ground truth instructions $g_{1:t}$, but during inference time, the stack is generated by the renderer while executing the predicted instructions. For both network variants, the RNN produces a categorical distribution π for both variants.

The parameters θ for either variant can be learned to maximize the log-likelihood of the ground truth instructions:

$$\mathcal{L}(\theta) = \sum_{i=1}^{N} \sum_{t=1}^{T_i} \log \pi_\theta(g_t^i | g_{1:t-1}^i, s_{1:t-1}^i, I^i) \tag{3.1}$$

3.3.1.2 Learning with policy gradients

Without target programs one can minimize a reconstruction error between the shape obtained by executing the program and the target. However, directly minimizing this error using gradient-based techniques is not possible since the output space is discrete and execution engines are typically not differentiable. Policy gradient techniques [245] from the reinforcement learning (RL) literature can instead be used in this case.

Concretely, the parser π_θ, that represents a policy network, can be used to sample a program y = (a_1,a_2 .. a_T) conditioned on the input shape I, and in the case of CSGNETSTACK, also on the stack S = (s_1, s_2 .. s_T). Note that while training using policy gradient and during inference time, the stack s_t is generated by the renderer while executing predicted instructions by the parser since ground-truth programs are unavailable. Then a reward R can be estimated by measuring the similarity between the generated image $\hat{I}$ obtained by executing the program and the target shape I. With this setup, we want to learn the network parameters θ that maximize the expected rewards over programs sampled under the predicted distribution $\pi_\theta(y|S, I)$ across images I sampled from a distribution $\mathcal{D}$:

$$\mathbb{E}_{I\sim\mathcal{D}}\left[J_\theta(I)\right] = \mathbb{E}_{I\sim\mathcal{D}} \sum_{t=1}^{T} \mathbb{E}_{y_t\sim\pi_\theta(y|s_{1:t-1},I)}\left[R\right]$$

The outer expectation can be replaced by a sample estimate on the training data. The gradient of the inner expectation can be obtained by rearranging the equation as [1]:

$$\nabla_\theta J_\theta(I) = \nabla_\theta \sum_y \pi_\theta(y) R = \sum_y \nabla_\theta \log \pi_\theta(y) \left[\pi_\theta(y) R\right]$$

Here we use the identity $\nabla_\theta \pi_\theta(y) = \pi_\theta(y) \nabla_\theta \log \pi_\theta(y)$. It is often intractable to compute the expectation $J_\theta(I)$ since the space of programs is very large. Hence, the expectation must be approximated. The REINFORCE algorithm computes a Monte-Carlo estimate (see also [216, 245] for derivations and explanation of the policy gradient algorithm). This is expressed as:

$$\nabla_\theta J_\theta(I) = \frac{1}{M} \sum_{m=1}^{M} \sum_{t=1}^{T} \nabla \log \pi_\theta(\hat{a}_t^m | \hat{a}_{1:t-1}^m, \hat{s}_{1:t-1}^m, I) R^m$$

by sampling M programs from the policy π_θ. Each program y^m is obtained by sampling instructions $\hat{a}_{t=1:T}^m$ from the distribution $\hat{a}_t^m \sim \pi_\theta(a_t | \hat{a}_{1:t-1}^m; \hat{s}_{1:t-1}^m, I)$ at every time step t until the stop symbol (`EOS`) is sampled. The reward R^m is calculated by executing the program y^m. Sampling-based estimates typically have high variance that can be reduced by subtracting a baseline without changing the bias as:

$$\nabla_\theta J_\theta(I) = \frac{1}{M} \sum_{m=1}^{M} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\hat{a}_t^m | \hat{a}_{1:t-1}^m, \hat{s}_{1:t-1}^m, I)(R^m - b) \tag{3.2}$$

A good choice of the baseline is the expected value of returns starting from t [217, 245]. We compute the baseline as the running average of past rewards.

Reward. The rewards should be primarily designed to encourage visual similarity of the generated program with the target. Visual similarity between two shapes is measured using

[1] conditioning on stack and input image is removed for the sake of brevity.

the Chamfer distance (CD) between points on the silhouettes of each shape. We focus on the silhouettes because these tend to be more related to the perceptual similarity of shapes [145]. The CD is between two point sets, $\mathbf{x}$ and $\mathbf{y}$, is defined as follows:

$$Ch(\mathbf{x}, \mathbf{y}) = \frac{1}{2|\mathbf{x}|} \sum_{x \in \mathbf{x}} \min_{y \in \mathbf{y}} \|x - y\|_2 + \frac{1}{2|\mathbf{y}|} \sum_{y \in \mathbf{y}} \min_{x \in \mathbf{x}} \|x - y\|_2$$

The points are scaled by the image diagonal, thus $Ch(\mathbf{x}, \mathbf{y}) \in [0, 1]\ \forall \mathbf{x}, \mathbf{y}$. The distance can be efficiently computed using distance transforms. In our implementation, we also set a maximum length T for the induced programs to avoid having too long or redundant programs (*e.g.*, repeating the same modeling instructions over and over again). We then define the reward as:

$$R = \begin{cases} f\left(Ch(\texttt{Edge}(I), \texttt{Edge}(\Re(y)\right), & y \text{ is valid} \\ 0, & y \text{ is invalid} \end{cases}$$

where f is a reward shaping function and $\Re$ is the CSG rendering engine that renders the program y into a binary image. Note that a valid program follows the grammar described in the Section 3.4.1, which can be verified by the execution engine. Since invalid programs get zero reward, the maximum length constraint on the programs helps the network to produce shorter programs with high rewards. We use maximum length $T = 13$ in all of our RL experiments. The function f shapes the CD as $f(x) = (1 - x)^\gamma$ with an exponent $\gamma > 0$. Higher values of γ makes the reward closer to zero, thereby making the network to produce programs with smaller CD. Table 3.1 (left) shows the dynamics of reward shaping function with different γ value and (right) shows that increasing γ values decreases the average CD calculated over the test set. We choose $\gamma = 20$ in our experiments, as this gives best performance on our validation set as shown in Table 3.1.

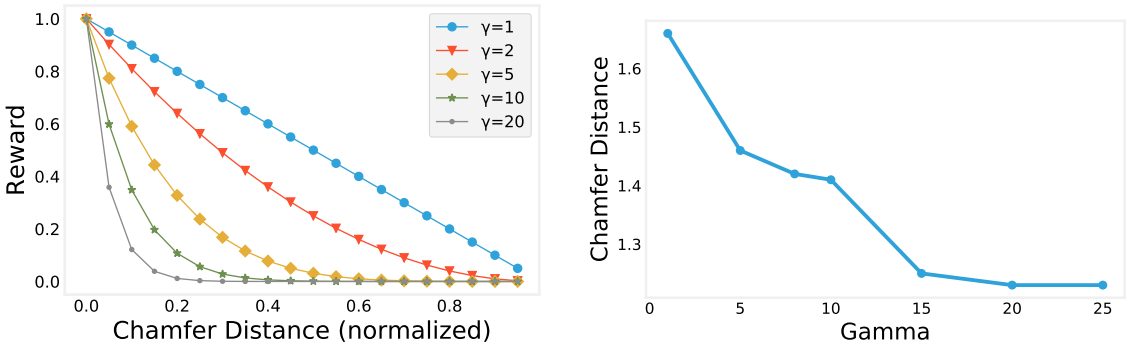


Table 3.1: **Reward shaping.** (Left) We visualize the skewness introduced by the γ in the reward function. (Right) Larger γ value produces smaller CD (in number of pixels) when our model is trained using REINFORCE.

3.3.2 Inference

Greedy decoding and beam search. Estimating the most likely program given an input is intractable using RNNs. Instead one usually employs a greedy decoder that picks the most likely instruction at each time step. An alternate is to use a beam search procedure that maintains the k-best likely sequences at each time step. In our experiments we report results with varying beam sizes.

Visually-guided refinement. Both parser variants produce a program with a discrete set of primitives. However, further refinement can be done by directly optimizing the position and size of the primitives to maximize the reward. The refinement step keeps the program structure of the program and primitive type fixed but uses a heuristic algorithm [177] to optimize the parameters using feedback from the rendering engine. In our experiments, we observed that the algorithm converges to a local minima in about 10 iterations of refinement and consistently improves the results.

3.4 Experiments

We describe our experiments on different datasets exploring the generalization capabilities of our network variants (CSGNET and CSGNETSTACK). We first describe our datasets:

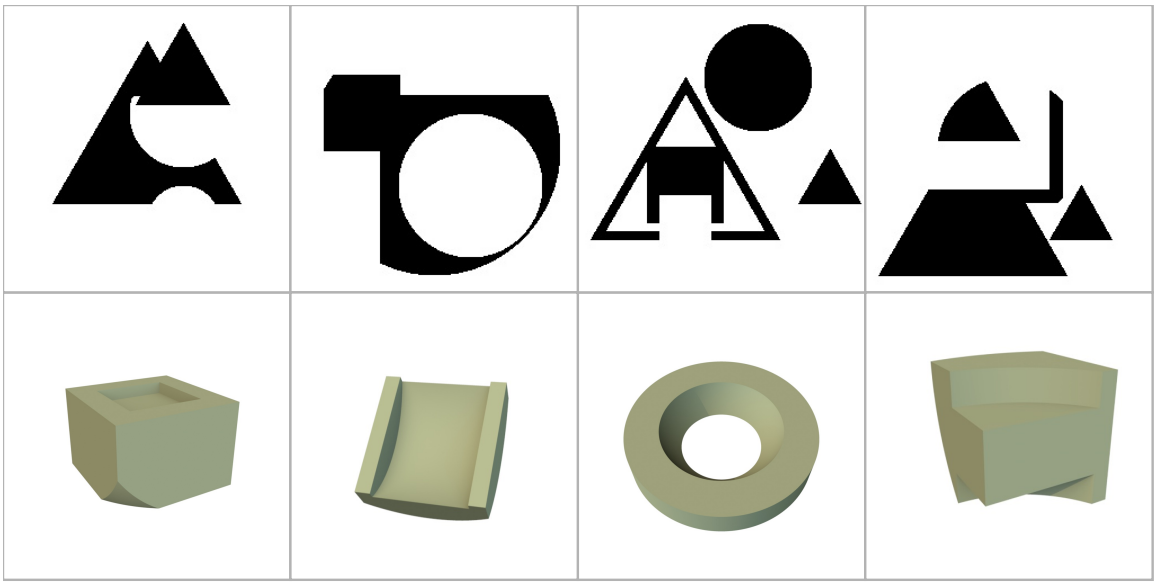


Figure 3.5: **Samples of our synthetically generated programs.** 2D samples are in the top row and 3D samples in the bottom. For clarity, the shapes are rendered in their original, high-resolution mesh format before voxelization.

(i) an automatically generated dataset of 2D and 3D shapes based on synthetic generation of CSG programs, (ii) 2D CAD shapes mined from the web where ground-truth programs are not available, and (iii) logo images mined also from the web where ground-truth programs are also not available. Below we discuss our qualitative and quantitative results on the above dataset.

3.4.1 Datasets

To train our network in the supervised learning setting, we automatically created a large set of 2D and 3D CSG-based synthetic programs according to the grammars described below.

Synthetic 2D shapes. We sampled derivations of the following CSG grammar to create our synthetic dataset in the 2D case:

Program Length	2D			3D		
	Train	Val	Test	Train	Val	Test
3	25k	5k	5k	100k	10k	20k
5	100k	10k	50k	200k	20k	40k
7	150k	20k	50k	400k	40k	80k
9	250k	20k	50k	-	-	-
11	350k	20k	100k	-	-	-
13	350k	20k	100k	-	-	-

Table 3.2: **Statistics of our 2D and 3D synthetic dataset.**

$$S \rightarrow E;$$

$$E \rightarrow EET \mid P(L, R);$$

$$T \rightarrow intersect \mid union \mid subtract;$$

$$P \rightarrow square \mid circle \mid triangle;$$

$$L \rightarrow \big[8 : 8 : 56\big]^2; \;\; R \rightarrow \big[8 : 4 : 32\big].$$

Primitives are specified by their type: *square*, *circle*, or *triangle*, locations L and circumscribing circle of radius R on a canvas of size 64×64. There are three boolean operations: $intersect$, $union$, and $subtract$. L is discretized to lie on a square grid with spacing of 8 units and R is discretized with spacing of 4 units. The *triangles* are assumed to be upright and equilateral. The synthetic dataset is created by sampling random programs containing different number of primitives from the above grammar, constraining the distribution of various primitive types and operation types to be uniform. We also ensure that no duplicate programs exist in our dataset. The primitives are rendered as binary images and the programs are executed on a canvas of 64×64 pixels. Samples from our dataset are shown in Figure 3.5. Table 3.2 provides details about the size and splits of our dataset.

Synthetic 3D shapes. We sampled derivations of the following grammar in the case of 3D CSG:

$$S \rightarrow E;\ E \rightarrow EET;$$

$$E \rightarrow sp(L,R) \mid cu(L,R) \mid cy(L,R,H)$$

$$T \rightarrow intersect \mid union \mid subtract;$$

$$L \rightarrow \big[8:8:56\big]^3$$

$$R \rightarrow \big[8:4:32\big];\ H \rightarrow \big[8:4:32\big].$$

The operations are same as in the 2D case. Three basic solids are denoted by 'sp': Sphere, 'cu': Cube, 'cy': Cylinder. L represents the center of primitive in a 3D voxel grid. R specifies radius of sphere and cylinder, or the size of cube. H is the height of cylinder. The primitives are rendered as voxels and the programs are executed on a 3D volumetric grid of size $64 \times 64 \times 64$. We used the same random sampling method as used for the synthetic 2D dataset, resulting in 3D CSG programs. 3D shape samples are shown in Figure 3.5.

2D CAD shapes. We collected $8K$ CAD shapes from the Trimble 3DWarehouse dataset [6] in three categories: chair, desk and lamps. We rendered the CAD shapes into 64×64 binary masks from their front and side views. In Section 3.4, we show that the rendered shapes can be parsed effectively through our visual program induction method. We split this dataset into $5K$ shapes for training, $1.5K$ validation and $1.5K$ for testing.

Web logos. We mined 20 binary logos from the web that can be modeled using the primitives in our output shapes. We test our approach on these logos without further training or fine-tuning our net on this data.

3.4.2 Implementation details

2D shape parsing. Our encoder is based on an image-based convnet in the case of 2D inputs. In the case of CSGNETSTACK, the input to the network is a fixed size stack along with target image concatenated along the channel dimension, resulting in an the input tensor

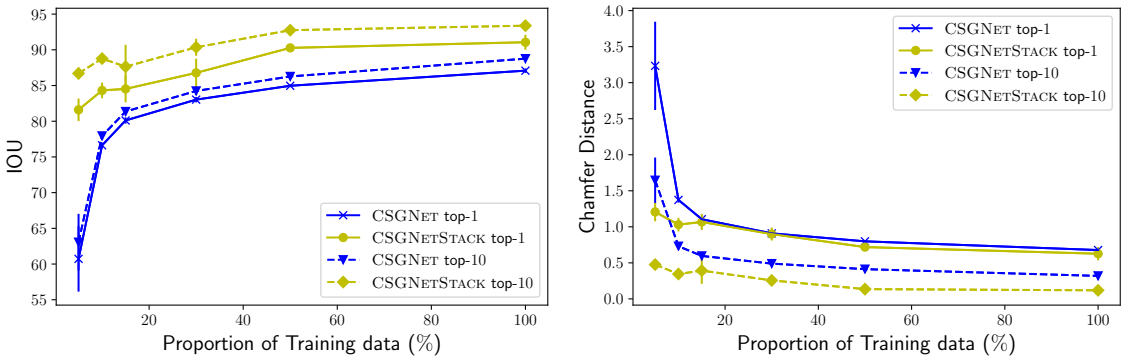


Figure 3.6: **Performance (Left: IOU, Right: chamfer distance) of models by changing training size on our synthetic dataset.** Training is done using $x\%$ of the complete dataset, where x is shown on the horizontal axis. The top-k beam sizes used during decoding at test time are shown in the legend. The performance of CSGNET (our basic non-stack neural shape parser) is shown in blue and the performance of CSGNETSTACK (our variant that uses the execution stack) is shown in lime. The above plots show the average of the metrics evaluated at 4 different training runs.

of size $64 \times 64 \times (K+1)$, where K is the number of used maps in the stack (stack size). In the architecture without stack (CSGNET), K is simply set to 0. The output of the encoder is passed as input to our GRU-based decoder at every program step. The hidden state of our GRU units is passed through two fully-connected layers, which are then converted into a probability distribution over program instructions through a classification layer. For the 2D CSG there are 400 unique instructions corresponding to 396 different primitive types, discrete locations and sizes, the 3 boolean operations and the stop symbol.

3D shape parsing. In the case of 3D shapes, the encoder is based on an volumetric, voxel-based convnet. 3D-CSGNETSTACK concatenates the stack with the target shape along the channel dimension, resulting in an input tensor of size $64 \times 64 \times 64 \times (K+1)$, where K is the number of used maps in the stack (stack size). In the architecture without stack (3D-CSGNET), K is simply set to 0. The encoder comprises of multiple layers of $3D$ convolutions yielding a fixed size encoding vector. Similarly to the $2D$ case, the GRU-based decoder takes the output of the encoder and sequentially produces the program instructions.

Method	IOU (k=1) ↑	IOU (k=10) ↑	CD (k=1) ↓	CD (k=10) ↓
NN	73.9	-	1.93	-
CSGNET	86.77	88.74	0.70	0.32
CSGNETSTACK	**91.33**	**93.45**	**0.60**	**0.12**

Table 3.3: **Comparison of a NN baseline with the supervised network without stack (CSGNET) and with stack (CSGNETSTACK) on the synthetic 2D dataset.** Results are shown using Chamfer Distance (CD) and IOU metric by varying beam sizes (k) during decoding. CD is in number of pixels.

In this case, there are 6635 unique instructions with 6631 different types of primitives with different sizes and locations, plus 3 boolean modeling operations and a stop symbol.

During training, on synthetic dataset, we sample images/3D shapes rendered from programs of variable length (up to 13 for 2D and up to 7 for 3D dataset) from training dataset from Table 3.2. More details about the architecture of our encoder and decoder (number and type of layers) are provided in the Appendix.

For supervised learning, we use the Adam optimizer [120] with learning rate 0.001 and dropout of 0.2 in non-recurrent network connections. For reinforcement learning, we use stochastic gradient descent with 0.9 momentum, 0.01 learning rate, and with the same dropout as above.

3.4.3 Results

We evaluate our network variants in two different ways: (i) as models for inferring the entire program, and (ii) as models for inferring primitives, *i.e.*, as object detectors.

3.4.3.1 Inferring programs

Evaluation on the synthetic 2D shapes. We perform supervised learning to train our stack-based network CSGNETSTACK and the non-stack-based network CSGNET on the training split of this synthetic dataset, and evaluate performance on its test split under different beam sizes. We compare with a baseline that retrieves a program in the training split using a Nearest Neighbor (NN) approach. In NN setting, the program for a test image

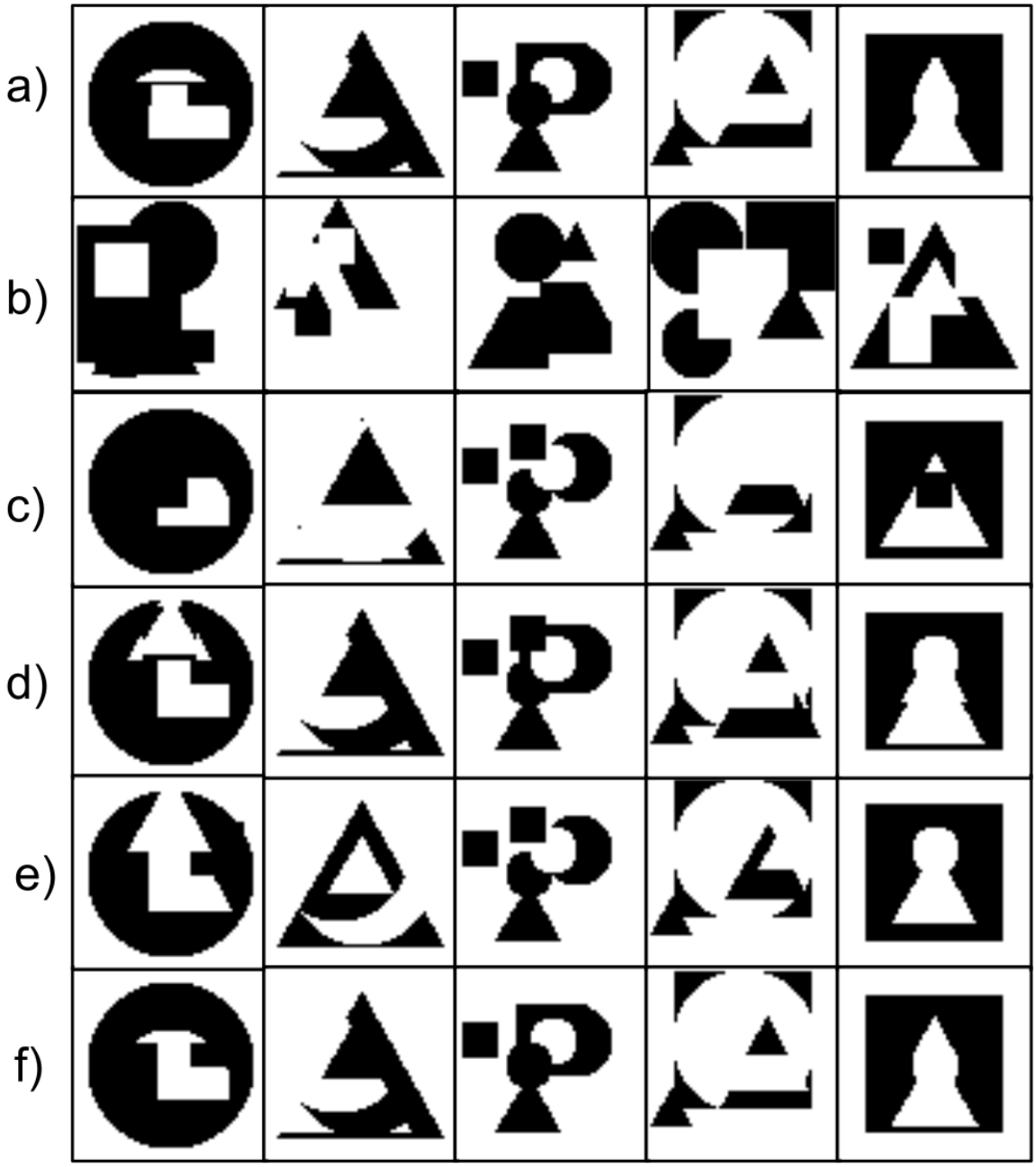


Figure 3.7: **Comparison of performance on synthetic 2D dataset.** a) Input image, b) NN-retrieved image, c) top-1 prediction of CSGNET, d) top-1 prediction of CSGNETSTACK, e) top-10 prediction of CSGNET and f) top-10 prediction of CSGNETSTACK.

Method	Train	Test	CD (@refinement iterations) ↓					
			i=0	i=1	i=2	i=4	i=10	i=∞
NN	-	-	1.92	1.22	1.13	1.08	1.07	1.07
CSGNET	Supervised	k=1	2.45	1.2	1.03	0.97	0.96	0.96
CSGNET	Supervised	k=10	1.68	0.79	0.67	0.63	0.62	0.62
CSGNETSTACK	Supervised	k=1	3.98	2.66	2.41	2.29	2.25	2.25
CSGNETSTACK	Supervised	k=10	1.38	0.56	0.45	0.40	0.39	0.39
CSGNET	RL	k=1	1.40	0.71	0.63	0.60	0.60	0.60
CSGNET	RL	k=10	1.19	0.53	0.47	0.41	0.41	0.41
CSGNETSTACK	RL	k=1	1.27	0.67	0.60	0.58	0.57	0.57
CSGNETSTACK	RL	k=10	**1.02**	**0.48**	**0.43**	**0.35**	**0.34**	**0.34**

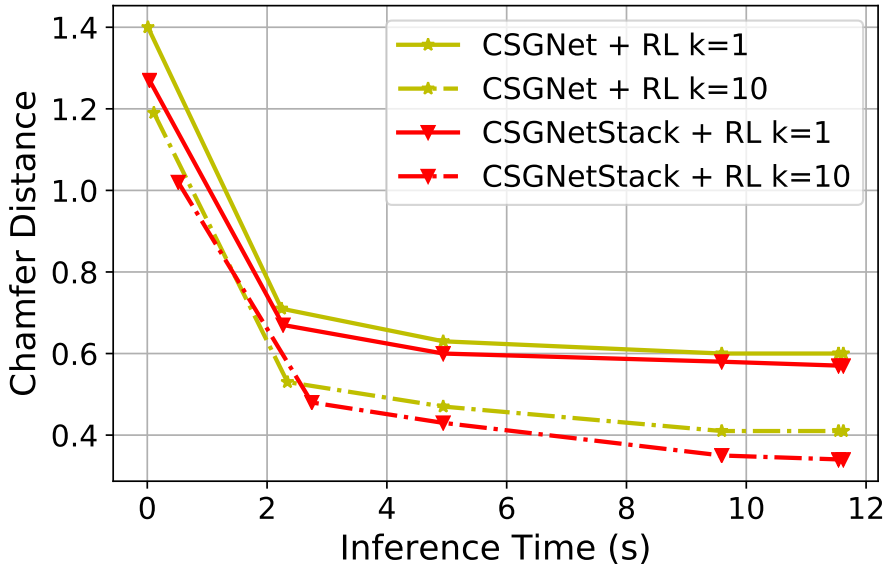


Table 3.4: **Comparison of various approaches on the CAD shape dataset**. CSGNET: neural shape parser without stack, CSGNETSTACK: parser with stack, NN: nearest neighbor. Left: Results are shown with different beam sizes (k) during decoding. Fine-tuning using RL improves the performance of both network, with CSGNETSTACK perfoming the best. Increasing the number of iterations (i) of visually guided refinement during testing also improves results significantly. $i = \infty$ corresponds to running visually guided refinement till convergence. Bottom: Inference time for different methods. Increasing number of iterations of visually guided refinement improves the performance, with least CD in a given inference time is produced by Stack based architecture. CD metric is in number of pixels.

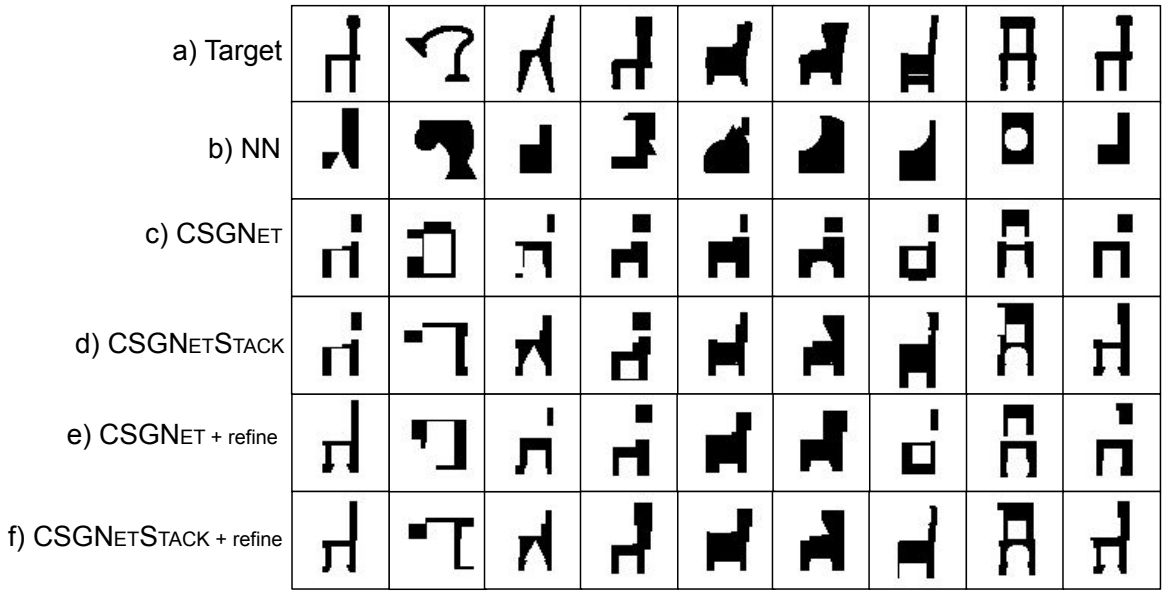


Figure 3.8: **Comparison of performance on the 2D CAD dataset**. a) Target image, b) NN retrieved image, c) best result from beam search on top of CSGNET fine-tuned with RL, d) best result from beam search on top of CSGNETSTACK fine-tuned with RL, and refining results using the visually guided search on the best beam result of CSGNET (e) and CSGNETSTACK (f).

is retrieved by taking the program of the train image that is most similar to the test image using the IOU metric.

Table 3.3 compares CSGNETSTACK, CSGNET, and a NN baseline using the Chamfer distance and IOU between the test target and predicted shapes using the complete synthetic dataset. Our parser is able to outperform the NN method. One would expect that NN would perform well here because the size of the training set is large. However, our results indicate that our compositional parser is better at capturing shape variability, which is still significant in this dataset. Results are also shown with increasing beam sizes (k) during decoding, which consistently improves performance. Figure 3.7 also shows the programs retrieved through NN and our generated program for a number of characteristic examples in our test split of our synthetic dataset.

We also examine the learning capability of CSGNETSTACK with significantly less synthetic training dataset in comparison to CSGNET in the Figure 3.6. With just 5% of the

total dataset, CSGNETSTACK performs 80% IOU (1.3 CD) in comparison to 70% IOU (1.7 CD) using CSGNET. The CSGNETSTACK continues to perform better compared to CSGNET in the case of more training data. This shows that incorporating the extra knowledge in the form of an execution stack based on the proposed architecture makes it easier to learn to parse shapes.

Evaluation on 2D CAD shapes. For this dataset, we report results on its test split under two conditions: (i) when training our network only on synthetic data, and (ii) when training our network on synthetic data and also fine-tuning it on the training split of rendered CAD dataset using policy gradients.

Table 3.4 shows quantitative results on this dataset. We first compare with the NN baseline. For any shape in this dataset, where ground truth program is not available, NN retrieves a shape from synthetic dataset and we use the ground truth program of the retrieved synthetic shape for comparison.

We then list the performance of CSGNETSTACK and CSGNET trained in a supervised manner only on our synthetic dataset. Further training with Reinforcement Learning (RL) on the training split of the 2D CAD dataset improves the results significantly and outperforms the NN approach by a considerable margin. This also shows the advantage of using RL, which trains the shape parser without ground-truth programs. The stack based network CSGNETSTACK performs better than CSGNET showing better generalization on the new dataset. We note that directly training the network using RL alone does not yield good results which suggests that the two-stage learning (supervised learning and RL) is important. Finally, optimizing the best beam search program with visually guided refinement yielded results with the smallest Chamfer Distance. Figure 3.8 shows a comparison of the rendered programs for various examples in the test split of the 2D CAD dataset for variants of our network. Visually guided refinement on top of beam search of our two stage-learned network qualitatively produces results that best match the input image.

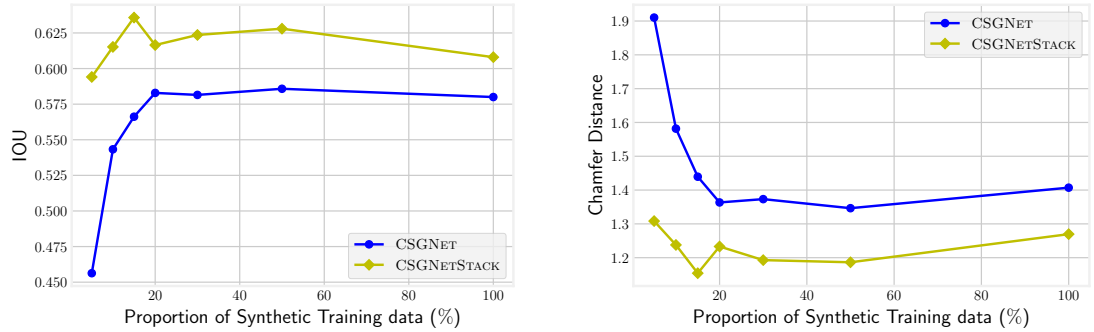


Figure 3.9: **Performance (Left: IOU, Right: chamfer distance) of CSGNET and CSGNETSTACK on the test split of the 2D CAD dataset wrt the size of the synthetic dataset used to pre-train the two architectures.** Pre-training is done using $x\%$ of the complete synthetic dataset (x is shown on the horizontal axis) and fine-tuning is done on the complete CAD dataset. CSGNETSTACK performs better while using less proportion of the synthetic dataset for pretraining. Increasing the size of pretraining dataset beyond 15% leads to decrease in performance, which hints at slight overfitting on the synthetic dataset domain.

Method	NN	3D-CSGNET			3D-CSGNETSTACK		
		k=1	k=5	k=10	k=1	k=5	k=10
IOU (%) ↑	73.2	80.1	85.3	89.2	81.5	86.9	90.5
CD ↓	2.53	1.86	1.19	0.93	1.71	1.01	0.81

Table 3.5: **Comparison of the supervised network (3D-CSGNETSTACK and 3D-CSGNET) with NN baseline on the 3D dataset**. Results are shown using IOU(%) and Chamfer distance (CD) metrics, and varying beam sizes (k) during decoding. CD has been multiplied by 100.

We also show an ablation study indicating how much pretraining on the synthetic dataset is required to perform well on the CAD dataset in Figure 3.9. With just 5% of the synthetic dataset based pretraining, CSGNETSTACK gives 60% IOU (and 1.3 CD) in comparison to 46% IOU (and 1.9 CD), which shows the faster learning capability of our stack based architecture. Increasing the synthetic training size used in pretraining shows slight decrease in performance for the CSGNETSTACK network after 15%, which hints at the overfitting of the network on the synthetic dataset domain.

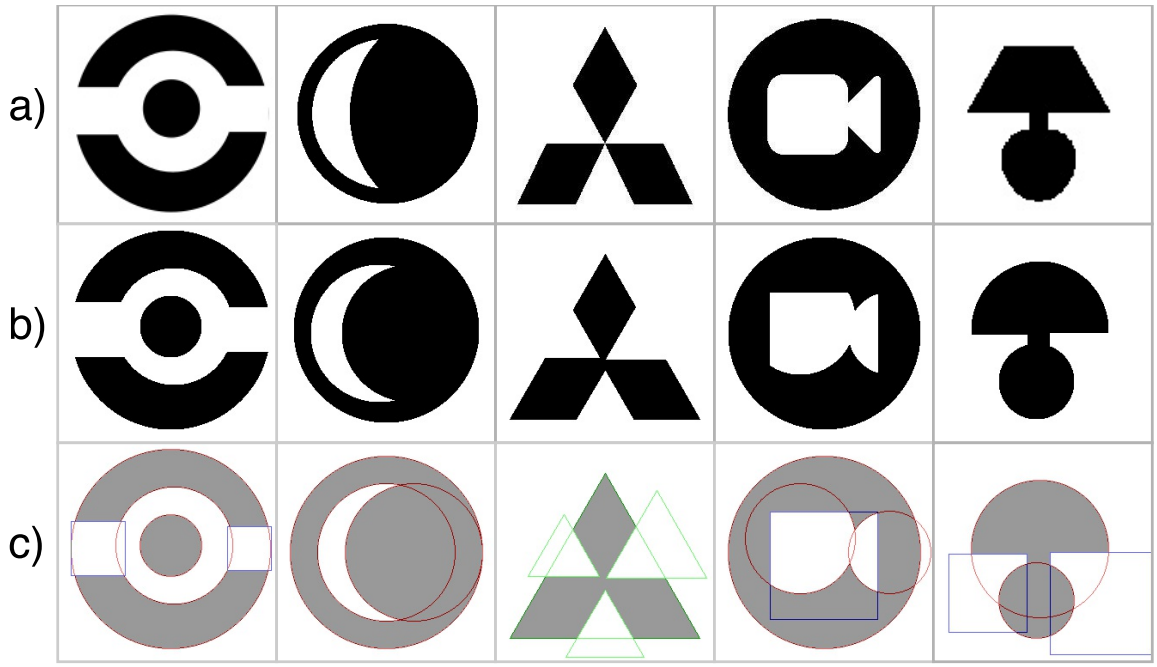


Figure 3.10: **Results for our logo dataset**. a) Target logos, b) output shapes from CSGNET and c) inferred primitives from output program. Circle primitives are shown with red outlines, triangles with green and squares with blue.

Evaluation on Logos. We experiment with the logo dataset described in Section 3.4.1 (none of these logos participate in training). Outputs of the induced programs parsing the input logos are shown in Figure 3.10. In general, our method is able to parse logos into primitives well, yet performance can degrade when long programs are required to generate them, or when they contain shapes that are very different from our used primitives.

Evaluation on Synthetic 3D CSG. Finally, we show that our approach can be extended to 3D shapes. In the 3D CSG setting we use 3D-CSG dataset as described in the Section 3.4.1. We train a stack based 3D-CSGNETSTACK network that takes $64 \times 64 \times 64 \times (K+1)$ voxel representation of input shape concatenated with voxel representation of stack. The input to our 3D-CSGNET are voxelized shapes in a $64 \times 64 \times 64$ grid. Our output is a 3D CSG program, which can be rendered as a high-resolution polygon mesh (we emphasize that our output is not voxels, but CSG primitives and operations that can be computed and

rendered accurately). Figure 3.11 show pairs of input voxel grids and our output shapes from the test split of the 3D dataset. The quantitative results are shown in the Table 3.5, where we compare our 3D-CSGNETSTACK and 3D-CSGNET networks at different beam search decodings with the NN method, using both the IOU and Chamfer distance metrics. Chamfer distance is computed by sampling 5k points on ground truth and predicted surface. The stack-based network also improves the performance over the non-stack variant. The results indicate that our method is promising in inducing correct programs for 3D shapes, which also has the advantage of accurately reconstructing the voxelized surfaces into high-resolution surfaces.

3.4.3.2 Primitive detection

Successful program induction for a shape requires not only predicting correct primitives but also correct sequences of operations to combine these primitives. Here we evaluate the shape parser as a primitive detector (*i.e.*, we evaluate the output primitives of our program, not the operations themselves). This allows us to directly compare our approach with bottom-up object detection techniques.

In particular we compare against Faster R-CNNs [182], a state-of-the-art object detector. The Faster R-CNN is based on the VGG-M network [29] and is trained using bounding-box and primitive annotations based on our 2D synthetic training dataset. At test time the detector produces a set of bounding boxes with associated class scores. The models are trained and evaluated on 640×640 pixel images. We also experimented with bottom-up approaches for primitive detection based on Hough transform [51] and other rule-based approaches. However, our experiments indicated that the Faster R-CNN was considerably better.

For a fair comparison, we obtain primitive detections from CSGNET trained on the 2D synthetic dataset only (same as the Faster R-CNN). To obtain detection scores, we sample k programs with beam-search decoding. The primitive score is the fraction of times it

Method	Circle	Square	Triangle	Mean	Speed (im/s)
Faster R-CNN	87.4	71.0	81.8	80.1	5
CSGNET, $k = 10$	86.7	79.3	83.1	83.0	80
CSGNET, $k = 40$	**88.1**	**80.7**	**84.1**	**84.3**	20

Table 3.6: **MAP of detectors on the synthetic 2D shape dataset.** We also report detection speed measured as images/second on a NVIDIA 1070 GPU.

appears across all beam programs. This is a Monte Carlo estimate of our detection score. The accuracy can be measured through standard evaluation protocols for object detection (similar to those in the PASCAL VOC benchmark). We report the Mean Average Precision (MAP) for each primitive type using an overlap threshold between the predicted and the true bounding box of 0.5 intersection-over-union. Table 3.6 compares the parser network to the Faster R-CNN approach.

Our parser clearly outperforms the Faster R-CNN detector on the squares and triangles category. With larger beam search, we also produce slighly better results for circle detection. Interestingly, our parser is considerably faster than Faster R-CNN tested on the same GPU.

3.5 Limitations and Conclusion

We believe that our work represents a step towards neural generation of modeling programs given target visual content, which we believe is ambitious and hard. We demonstrated that the model generalizes across domains, including logos, 2D silhouettes, and 3D CAD shapes. It also is an effective primitive detector in the context of 2D shape primitive detection.

One might argue that the 2D images and 3D shapes considered in this work are relatively simple in structure or geometry. However, we would like to point out that even in this ostensibly simple application scenario (i) our method demonstrates competitive or even better results than state-of-the-art object detectors, and most importantly (ii) the problem of generating programs using neural networks was far from trivial to solve: based on our

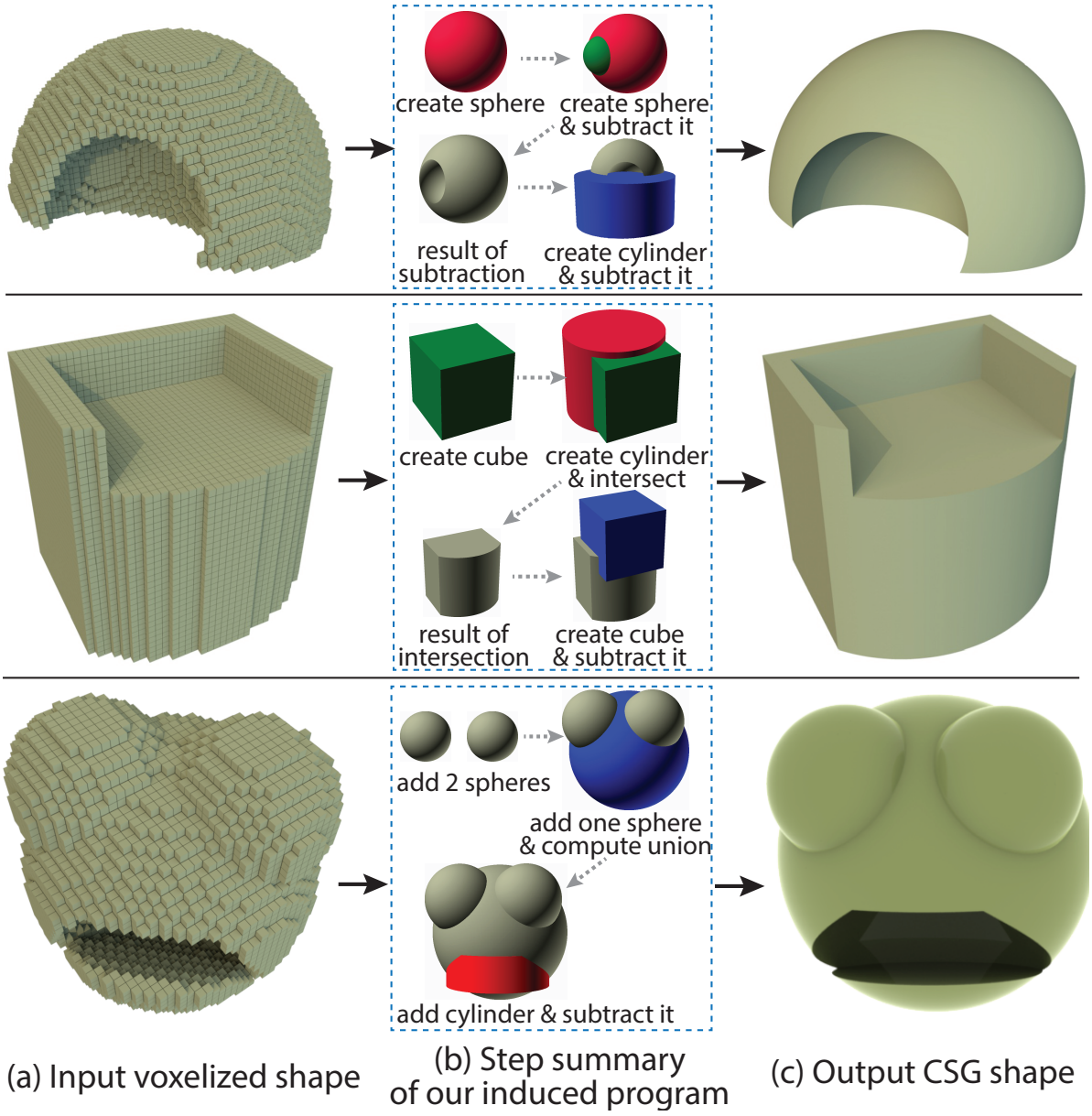


Figure 3.11: **Qualitative performance of 3D-CSGNET.** a) Input voxelized shape, b) Summarization of the steps of the program induced by 3D-CSGNET in the form of intermediate shapes, c) Final output created by executing induced program.

experiments, a combination of memory-enabled networks, supervised and RL strategies, along with beam and local exploration of the state space all seemed necessary to produce good results.

As future work, we would like to generalize our approach to longer programs with much larger spaces of parameters in the modeling operations and more sophisticated reward functions balancing perceptual similarity to the input image and program length. Our method is currently limited in its capability to generate 3D shapes, since the supported resolution is low due to the voxel representation we use in our encoder. Sparser shape representations [38] could help extending our network to handle more challenging 3D cases and datasets, such as ShapeNet [27] and ABC [124]. Another limitation is that our current control of the CSG program size is crude; it is based only on an upper bound of program size and a zero reward for invalid programs, which often occur with larger number of program instructions. Investigating more sophisticated complexity penalties could help promoting right-sized programs. Other promising direction is alternate strategies for combining bottom-up proposals and top-down approaches for parsing shapes, in particular, approaches based on constraint satisfaction and optimization.

Acknowledgments. The project is supported in part by grants from the National Science Foundation (NSF) CHS-1422441, CHS-1617333, IIS-1617917 and IIS-1908669. We also acknowledge the MassTech collaborative grant for funding the UMass GPU cluster.

CHAPTER 4

SURFACE FITTING FOR 3D POINT CLOUD

4.1 Introduction

3D point clouds can be rapidly acquired using 3D sensors or photogrammetric techniques. However, they are rarely used in this form in design and graphics applications. Observations from the computer-aided design and modeling literature [57, 61, 175, 194] suggest that designers often model shapes by constructing several non-overlapping patches placed seamlessly. The advantage of using several patches over a single continuous patch is that a much more diverse variety of geometric features and surface topologies can be created. The decomposition also allows easier interaction and editing. The goal of this work is to automate the time-consuming process of converting a 3D point cloud into a piecewise parametric surface representation as seen in Figure 6.1.

An important question is how surface patches should be represented. Patch representations in CAD and graphics are based on well-accepted geometric properties: (a) *continuity* in their tangents, normals, and curvature, making patches appear smooth, (b) *editability*, such that they can easily be modified based on a few intuitive degrees of freedom (DoFs), *e.g.*, control points or axes, and (c) *flexibility*, so that a wide variety of surface geometries can be captured. Towards this goal, we propose PARSENET, a **par**ametric **s**urfac**e** fitting **net**work architecture which produces a compact, editable representation of a point cloud as an assembly of geometric primitives, including open or closed B-spline patches.

PARSENET models a richer class of surfaces than prior work which *only* handles basic geometric primitives such as planes, cuboids and cylinders [139, 169, 202, 226]. While such primitives are continuous and editable representations, they lack the richness and flexibility

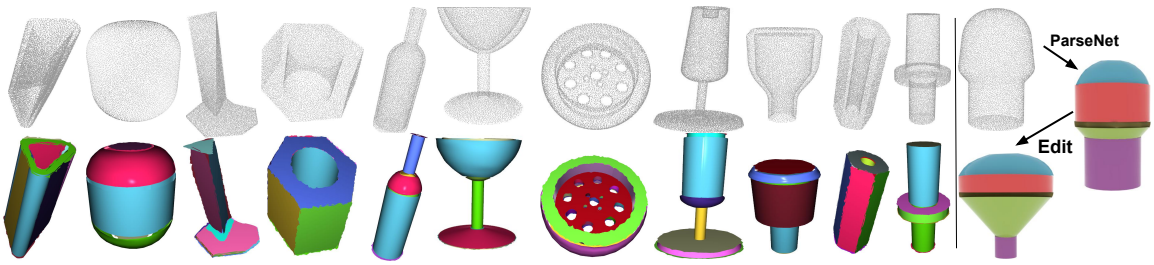


Figure 4.1: PARSENET decomposes point clouds (top row) into collections of assembled parametric surface patches including B-spline patches (bottom row). On the right, a shape is edited using the inferred parametrization.

of spline patches which are widely used in shape design. PARSENET includes a novel neural network (SPLINENET) to estimate an open or closed B-spline model of a point cloud patch. It is part of a *fitting module* (Section 4.3.2) which can also fit other geometric primitive types. The fitting module receives input from a *decomposition module*, which partitions a point cloud into segments, after which the fitting module estimates shape parameters of a predicted primitive type for each segment (Section 4.3.1). The entire pipeline, shown in Figure 5.3, is fully differentiable and trained end-to-end (Section 6.3.3). An optional geometric postprocessing step further refines the output.

Compared to purely analytical approaches, PARSENET produces decompositions that are more consistent with high-level semantic priors, and are more robust to point density and noise. To train and test PARSENET, we leverage a recent dataset of man-made parts [125]. Extensive evaluations show that PARSENET outperforms baselines (RANSAC and SPFN [139]) by 14.93% and 13.13% respectively for segmenting a point cloud into patches, and by 50%, and 47.64% relative error respectively for parametrizing each patch for surface reconstruction (Section 4.5).

To summarize, our contributions are:

- The first proposed end-to-end differentiable approach for representing a raw 3D point cloud as an assembly of parametric primitives *including* spline patches.

- Novel decomposition and primitive fitting modules, including SPLINENET, a fully-differentiable network to fit a cubic B-spline patch to a set of points.
- Evaluation of our framework vs prior analytical and learning-based methods.

4.2 Related Work

Our work builds upon related research on parametric surface representations and methods for primitive fitting. We briefly review relevant work in these areas. Of course, we also leverage extensive prior work on neural networks for general shape processing: see recent surveys on the subject [8].

4.2.0.0.1 Parametric surfaces. A parametric surface is a (typically diffeomorphic) mapping from a (typically compact) subset of $\mathbb{R}^2$ to $\mathbb{R}^3$. While most of the geometric primitives used in computer graphics (spheres, cuboids, meshes etc) can be represented parametrically, the term most commonly refers to curved surfaces used in engineering CAD modelers, represented as spline patches [57]. There are a variety of formulations – *e.g.* Bézier patches, B-spline patches, NURBS patches – with slightly different characteristics, but they all construct surfaces as weighted combinations of control parameters, typically the positions of a sparse grid of points which serve as editing handles.

More specifically, a B-spline patch is a smoothly curved, bounded, parametric surface, whose shape is defined by a sparse grid of control points $\mathbf{C} = \{\mathbf{c}_{p,q}\}$. The surface point with parameters $(u, v) \in [u_{\min}, u_{\max}] \times [v_{\min}, v_{\max}]$ and *basis functions* [57] $b_p(u)$, $b_q(v)$ is given by:

$$\mathbf{s}(u, v) = \sum_{p=1}^{P} \sum_{q=1}^{Q} b_p(u) b_q(v) \mathbf{c}_{p,q} \tag{4.1}$$

Please refer to Appendix for more details on B-spline patches.

4.2.0.0.2 Fitting geometric primitives. A variety of analytical (*i.e.* not learning-based) algorithms have been devised to approximate raw 3D data as a collection of geometric primitives: dominant themes include Hough transforms, RANSAC and clustering. The

literature is too vast to cover here, we recommend the comprehensive survey of Kaiser *et al.* [115]. In the particular case of NURBS patch fitting, early approaches were based on user interaction or hand-tuned heuristics to extract patches from meshes or point clouds [52, 96, 128]. In the rest of this section, we briefly review recent methods that *learn* how to fit primitives to 3D data.

Several recent papers [208, 215, 226, 269] also try to approximate 3D shapes as unions of cuboids or ellipsoids. Paschalidou *et al.* [169, 170] extended this to superquadrics. Sharma *et al.* [201, 202] developed a neural parser that represents a test shape as a collection of basic primitives (spheres, cubes, cylinders) combined with boolean operations. Tian *et al.* [224] handled more expressive construction rules (*e.g.*. loops) and a wider set of primitives. Because of the choice of simple primitives, such models are naturally limited in how well they align to complex input objects, and offer less flexible and intuitive parametrization for user edits.

More relevantly to our goal of modeling arbitrary curved surfaces, Gao *et al.* [68] parametrize 3D point clouds as extrusions or surfaces of revolution, generated by B-spline cross-sections detected by a 2D network. This method requires translational/rotational symmetry, and does not apply to general curved patches. Li *et al.* [139] proposed a supervised method to fit primitives to 3D point clouds, first predicting per-point segment labels, primitive types and normals, and then using a differential module to estimate primitive parameters. While we also chain segmentation with fitting in an end-to-end way, we differ from Li *et al.* in two important ways. First, our differentiable metric-learning segmentation produces improved results (Table 4.1). Second, a major goal (and technical challenge) for us is to significantly improve expressivity and generality by incorporating B-spline patches: we achieve this with a novel differentiable spline-fitting network. In a complementary direction, Yumer *et al.* [258] developed a neural network for fitting a single NURBS patch to an unstructured point cloud. While the goal is similar to our spline-fitting network, it is not combined with a decomposition module that jointly learns how to express a shape

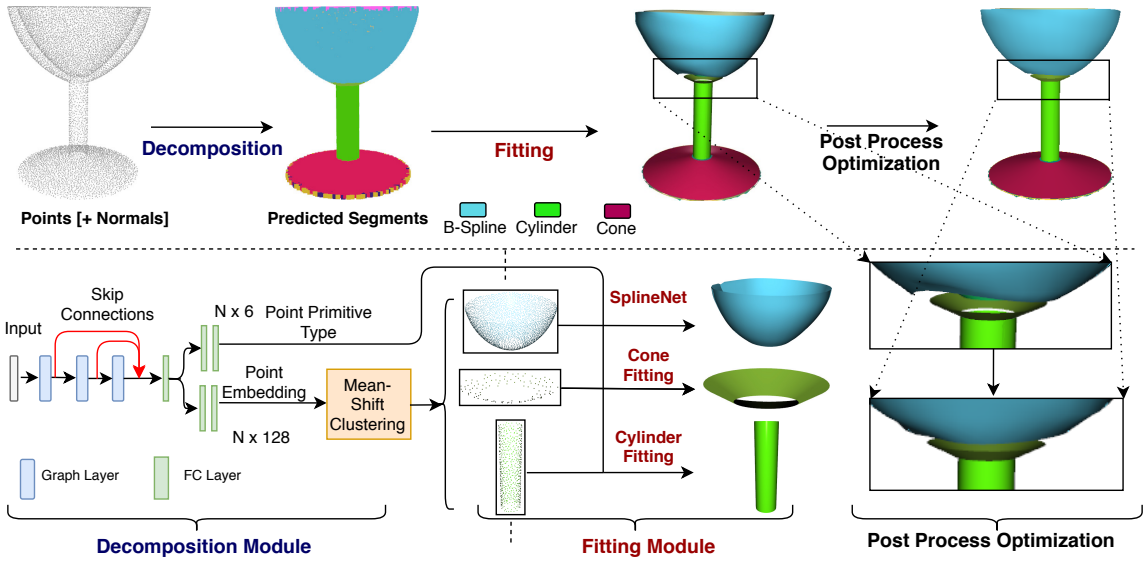


Figure 4.2: **Overview of** PARSENET **pipeline**. (1) The *decomposition module* (Section 4.3.1) takes a 3D point cloud (with optional normals) and decomposes it into segments labeled by primitive type. (2) The *fitting module* (Section 4.3.2) predicts parameters of a primitive that best approximates each segment. It includes a novel SPLINENET to fit B-spline patches. The two modules are jointly trained end-to-end. An optional postprocess module (Section 4.3.3) refines the output.

with *multiple* patches covering different regions. Further, their fitting module has several non-trainable steps which are not obviously differentiable, and hence cannot be used in our pipeline.

4.3 Method

The goal of our method is to reconstruct an input point cloud by predicting a set of parametric patches closely approximating its underlying surface. The first stage of our architecture is a *neural decomposition module* (Fig. 5.3) whose goal is to segment the input point cloud into regions, each labeled with a parametric patch type. Next, we incorporate a *fitting module* (Fig. 5.3) that predicts each patch's shape parameters. Finally, an optional post-processing *geometric optimization* step refines the patches to better align their boundaries for a seamless surface.

The input to our pipeline is a set of points $\mathbf{P} = \{\mathbf{p}_i\}_{i=1}^{N}$, represented either as 3D positions $\mathbf{p}_i = (x, y, z)$, or as 6D position + normal vectors $\mathbf{p}_i = (x, y, z, n_x, n_y, n_z)$. The output is a set of surface patches $\{\mathbf{s}_k\}$, reconstructing the input point cloud. The number of patches is automatically determined. Each patch is labeled with a type t_k, one of: sphere, plane, cone, cylinder, open/closed B-spline patch. The architecture also outputs a real-valued vector for each patch defining its geometric parameters, *e.g.* center and radius for spheres, or B-spline control points and knots.

4.3.1 Decomposition module

The first module (Fig. 5.3) decomposes the point cloud $\mathbf{P}$ into a set of segments such that each segment can be reliably approximated by one of the abovementioned surface patch types. To this end, the module first embeds the input points into a representation space used to reveal such segments. As discussed in Section 6.3.3, the representations are learned using metric learning, such that points belonging to the same patch are embedded close to each other, forming a distinct cluster.

4.3.1.0.1 Embedding network. To learn these point-wise representations, we incorporate edge convolution layers (EdgeConv) from DGCNN [239]. Each EdgeConv layer performs a graph convolution to extract a representation of each point with an MLP on the input features of its neighborhood. The neighborhoods are dynamically defined via nearest neighbors in the input feature space. We stack 3 EdgeConv layers, each extracting a 256-D representation per point. A max-pooling layer is also used to extract a global 1024-D representation for the whole point cloud. The global representation is tiled and concatenated with the representations from all three EdgeConv layers to form intermediate point-wise $(1024+256)$-D representations $\mathbf{Q} = \{\mathbf{q}_i\}$ encoding both local and global shape information. We found that a global representation is useful for our task, since it captures the overall geometric shape structure, which is often correlated with the number and type of expected patches. This representation is then transformed through fully connected layers and ReLUs, and

finally normalized to unit length to form the point-wise embedding $\mathbf{Y} = \{\mathbf{y}_i\}_{i=1}^{N}$ (128-D) lying on the unit hypersphere.

4.3.1.0.2 Clustering. A mean-shift clustering procedure is applied on the point-wise embedding to discover segments. The advantage of mean-shift clustering over other alternatives (*e.g.*, k-means or mixture models) is that it does not require the target number of clusters as input. Since different shapes may comprise different numbers of patches, we let mean-shift produce a cluster count tailored for each input. Like the pixel grouping of [126], we implement mean-shift iterations as differentiable recurrent functions, allowing back-propagation. Specifically, we initialize mean-shift by setting all points as seeds $\mathbf{z}_i^{(0)} = \mathbf{y}_i, \forall y_i \in R^{128}$. Then, each mean-shift iteration t updates each point's embedding on the unit hypersphere:

$$\mathbf{z}_i^{(t+1)} = \sum_{j=1}^{N} \mathbf{y}_j g(\mathbf{z}_i^{(t)}, \mathbf{y}_j) / (\sum_{j=1}^{N} g(\mathbf{z}_i^{(t)}, \mathbf{y}_j)) \tag{4.2}$$

where the pairwise similarities $g(\mathbf{z}_i^{(t)}, \mathbf{y}_j)$ are based on a von Mises-Fisher kernel with bandwidth β: $g(\mathbf{z}_i, \mathbf{y}_j) = \exp(\mathbf{z}_i^T \mathbf{y}_j / \beta^2)$ (iteration index dropped for clarity). The embeddings are normalized to unit vectors after each iteration. The bandwidth for each input point cloud is set as the average distance of each point to its 150^{th} neighboring point in the embedding space [206]. The mean-shift iterations are repeated until convergence (this occurs around 50 iterations in our datasets). We extract the cluster centers using non-maximum suppression: starting with the point with highest density, we remove all points within a distance β, then repeat. Points are assigned to segments based on their nearest cluster center. The point memberships are stored in a matrix $\mathbf{W}$, where $\mathbf{W}[i, k] = 1$ means point i belongs to segment k, and 0 means otherwise. The memberships are passed to the fitting module to determine a parametric patch per segment. During training, we use soft memberships for differentiating this step (more details in Section 4.4.3).

4.3.1.0.3 Segment Classification. To classify each segment, we pass the per-point representation $\mathbf{q}_i$, encoding local and global geometry, through fully connected layers and ReLUs,

followed by a softmax for a per-point probability $P(t_i = l)$, where l is a patch type (*i.e.*, sphere, plane, cone, cylinder, open/closed B-spline patch). The segment's patch type is determined through majority voting over all its points.

4.3.2 Fitting module

The second module (Fig. 5.3) aims to fit a parametric patch to each predicted segment of the point cloud. To this end, depending on the segment type, the module estimates the shape parameters of the surface patch.

4.3.2.0.1 Basic primitives. Following Li *et al.* [139], we estimate the shape of basic primitives with least-squares fitting. This includes center and radius for spheres; normal and offset for planes; center, direction and radius for cylinders; and apex, direction and angle for cones. We also follow their approach to define primitive boundaries.

4.3.2.0.2 B-Splines. Analytically parametrizing a set of points as a spline patch in the presence of noise, sparsity and non-uniform sampling, can be error-prone. Instead, predicting control points directly with a neural network can provide robust results. We propose a neural network SPLINENET, that inputs points of a segment, and outputs a fixed size control-point grid. A stack of three EdgeConv layers produce point-wise representations concatenated with a global representation extracted from a max-pooling layer (as for decomposition, but weights are not shared). This equips each point i in a segment with a 1024-D representation $\boldsymbol{\phi}_i$. A segment's representation is produced by max-pooling over its points, as identified through the membership matrix $\mathbf{W}$ extracted previously:

$$\boldsymbol{\phi}_k = \max_{i=1\ldots N} (\mathbf{W}[i,k] \cdot \boldsymbol{\phi}_i). \tag{4.3}$$

Finally, two fully-connected layers with ReLUs transform $\boldsymbol{\phi}_k$ to an initial set of 20×20 control points $\mathbf{C}$ unrolled into a 1200-D output vector. For a segment with a small number of points, we upsample the input segment (with nearest neighbor interpolation) to 1600

points. This significantly improved performance for such segments (Table 4.2). For closed B-spline patches, we wrap the first row/column of control points. Note that the network parameters to produce open and closed B-splines are not shared. Fig. 4.5 visualizes some predicted B-spline surfaces.

4.3.3 Post-processing module

SPLINENET produces an initial patch surface that approximates the points belonging to a segment. However, patches might not entirely cover the input point cloud, and boundaries between patches are not necessarily well-aligned. Further, the resolution of the initial control point grid (20×20) can be further adjusted to match the desired surface resolution. As a post-processing step, we perform an optimization to produce B-spline surfaces that better cover the input point cloud, and refine the control points to achieve a prescribed fitting tolerance.

4.3.3.0.1 Optimization. We first create a grid of 40×40 points on the initial B-spline patch by uniformly sampling its UV parameter space. We tessellate them into quads. Then we perform a maximal matching between the quad vertices and the input points of the segment, using the Hungarian algorithm with L2 distance costs. We then perform an as-rigid-as-possible (ARAP) [209] deformation of the tessellated surface towards the matched input points. ARAP is an iterative, detail-preserving method to deform a mesh so that selected vertices (pivots) achieve targets position, while promoting locally rigid transformations in one-ring neighborhoods (instead of arbitrary ones causing shearing/stretching). We use the boundary vertices of the patch as pivots so that they move close to their matched input points. Thus, we promote coverage of input points by the B-spline patches. After the deformation, the control points are re-estimated with least-squares [175].

4.3.3.0.2 Refinement of B-spline control points. After the above optimization, we again perform a maximal matching between the quad vertices and the input points of the segment. As a result, the input segment points acquire 2D parameter values in the patch's UV

parameter space, which can be used to re-fit any other grid of control points [175]. In our case, we iteratively upsample the control point grid by a factor of 2 until a fitting tolerance, measured via Chamfer distance, is achieved. If the tolerance is satisfied by the initial control point grid, we can similarly downsample it iteratively. In our experiments, we set the fitting tolerance to 5×10^{-4}. In Fig. 4.5 we show the improvements from the post-processing step.

4.4 Training

To train the neural decomposition and fitting modules of our architecture, we use supervisory signals from a dataset of 3D shapes modeled through a combination of basic geometric primitives and B-splines. Below we describe the dataset, then we discuss the loss functions and the steps of our training procedure.

4.4.1 Dataset

The ABC dataset [125] provides a large source of 3D CAD models of mechanical objects whose file format stores surface patches and modeling operations that designers used to create them. Since our method is focused on predicting surface patches, and in particular B-spline patches, we selected models from this dataset that contain at least one B-spline surface patch. As a result, we ended up with a dataset of 32K models (24K, 4K, 4K train, test, validation sets respectively). We call this ABCPARTSDATASET. All shapes are centered in the origin and scaled so they lie inside unit cube. To train SPLINENET, we also extract 32K closed and open B-spline surface patches each from ABC dataset and split them into 24K, 4K, 4K train, test, validation sets respectively. We call this SPLINEDATASET. We report the average number of different patch types in the Appendix.

4.4.1.0.1 Preprocessing. Based on the provided metadata in ABCPARTSDATASET, each shape can be rendered based on the collection of surface patches and primitives it contains (Figure 4.4). Since we assume that the inputs to our architecture are point clouds, we

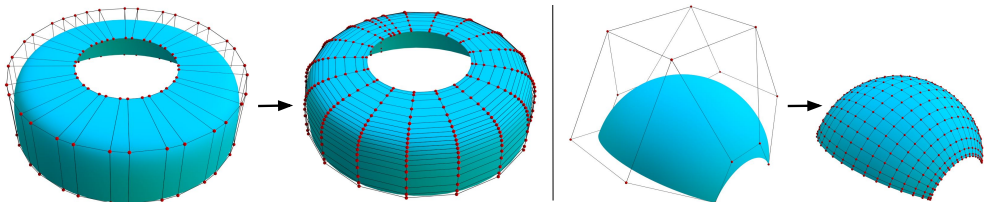


Figure 4.3: **Standardization:** Examples of B-spline patches with a variable number of control points (shown in red), each standardized with 20×20 control points. Left: closed B-spline and Right: open B-spline. (Please zoom in.)

first sample each shape with 10K points randomly distributed on the shape surface. We also add noise in a uniform range $[-0.01, 0.01]$ along the normal direction. Normals are also perturbed with random noise in a uniform range of $[-3, 3]$ degrees from their original direction.

4.4.2 Loss functions

We now describe the different loss functions used to train our neural modules. The training procedure involving their combination is discussed in Section 4.4.3.

4.4.2.0.1 Embedding loss. To discover clusters of points that correspond well to surface patches, we use a metric learning approach. The point-wise representations **Z** produced by our decomposition module after mean-shift clustering are learned such that point pairs originating from the same surface patch are embedded close to each other to favor a cluster formation. In contrast, point pairs originating from different surface patches are pushed away from each other. Given a triplet of points (a, b, c), we use the triplet loss to learn the embeddings: c where τ the margin is set to 0.9. Given a triplet set $\mathcal{T}_\mathcal{S}$ sampled from each point set $\mathcal{S}$ from our dataset $\mathcal{D}$, the embedding objective sums the loss over triplets:

$$L_{emb} = \sum_{\mathcal{S} \in \mathcal{D}} \frac{1}{|\mathcal{T}_\mathcal{S}|} \sum_{(a,b,c) \in \mathcal{T}_\mathcal{S}} \ell_{emb}(a, b, c). \tag{4.4}$$

4.4.2.0.2 Segment classification loss. To promote correct segment classifications according to our supported types, we use the cross entropy loss: $L_{class} = -\sum_{i \in \mathcal{S}} \log(p_i^t)$ where p_t^i

is the probability of the i^{th} point of shape $\mathcal{S}$ belonging to its ground truth type t, computed from our segment classification network.

4.4.2.0.3 Control point regression loss. This loss function is used to train SPLINENET. As discussed in Section 4.3.2, SPLINENET produces 20×20 control points per B-spline patch. We include a supervisory signal for this control point grid prediction. One issue is that B-spline patches have a variable number of control points in our dataset. Hence we reparametrize each patch by first sampling $M = 3600$ points and estimating a new 20×20 reparametrization using least-squares fitting [128, 175], as seen in the Figure 4.3. In our experiments, we found that this standardization produces no practical loss in surface reconstructions in our dataset. Finally, our reconstruction loss should be invariant to flips or swaps of control points grid in u and v directions. Hence we define a loss that is invariant to such permutations:

$$L_{cp} = \sum_{\mathcal{S} \in \mathcal{D}} \frac{1}{|\mathcal{S}^{(b)}|} \sum_{s_k \in \mathcal{S}^{(b)}} \frac{1}{|\mathbf{C}_k|} \min_{\pi \in \Pi} ||\mathbf{C}_k - \pi(\hat{\mathbf{C}}_k)||^2 \tag{4.5}$$

where $\mathcal{S}^{(b)}$ is the set of B-spline patches from shape $\mathcal{S}$, $\mathbf{C}_k$ is the predicted control point grid for patch $\mathbf{s}_k$ ($|\mathbf{C}_k| = 400$ control points), $\pi(\hat{\mathbf{C}}_k)$ is permutations of the ground-truth control points from the set Π of 8 permutations for open and 160 permutations for closed B-spline.

4.4.2.0.4 Laplacian loss. This loss is also specific to B-Splines using SPLINENET. For each ground-truth B-spline patch, we uniformly sample ground truth surface, and measure the surface Laplacian capturing its second-order derivatives. We also uniformly sample the predicted patches and measure their Laplacians. We then establish Hungarian matching between sampled points in the ground-truth and predicted patches, and compare the Laplacians of the ground-truth points $\hat{\mathbf{r}}_m$ and corresponding predicted ones $\mathbf{r}_n$ to improve the agreement between their derivatives as follows:

$$L_{lap} = \sum_{\mathcal{S} \in \mathcal{D}} \frac{1}{|\mathcal{S}^{(b)}| \cdot M} \sum_{\mathbf{s}_k \in \mathcal{S}^{(b)}} \sum_{\mathbf{r}_n \in s_k} ||\mathcal{L}(\mathbf{r}_n) - \mathcal{L}(\hat{\mathbf{r}}_m)||^2 \tag{4.6}$$

where $\mathcal{L}(\cdot)$ is the Laplace operator on patch points, and $M = 1600$ point samples.

4.4.2.0.5 Patch distance loss. This loss is applied to both basic primitive and B-splines patches. Inspired by [139], the loss measures average distances between predicted primitive patch s_k and uniformly sampled points from the ground truth patch as:

$$L_{dist} = \sum_{\mathcal{S} \in \mathcal{D}} \frac{1}{K_{\mathcal{S}}} \sum_{k=1}^{K_{\mathcal{S}}} \frac{1}{M_{\hat{\mathbf{s}}_k}} \sum_{n \in \hat{\mathbf{s}}_k} D^2(\mathbf{r}_n, \mathbf{s}_k), \tag{4.7}$$

where $K_{\mathcal{S}}$ is the number of predicted patches for shape $\mathcal{S}$, $M_{\hat{\mathbf{s}}_k}$ is number of sampled points $\mathbf{r}_n$ from ground patch $\hat{\mathbf{s}}_k$, $D^2(\mathbf{r}_n, \mathbf{s}_k)$ is the squared distance from $\mathbf{r}_n$ to the predicted primitive patch surface $\mathbf{s}_k$. These distances can be computed analytically for basic primitives [139]. For B-splines, we use an approximation based on Chamfer distance between sample points.

4.4.3 Training procedure

One possibility for training is to start it from scratch using a combination of all losses. Based on our experiments, we found that breaking the training procedure into the following steps leads to faster convergence and to better minima:

- We first pre-train the networks of the decomposition module using ABCPARTS-DATASET with the sum of embedding and classification losses: $L_{emb} + L_{class}$. Both losses are necessary for point cloud decomposition and classification.

- We then pre-train the SPLINENET using SPLINEDATASET for control point prediction exclusively on B-spline patches using $L_{cp} + L_{lap} + L_{dist}$. We note that we experimented training the B-spline patch prediction only with the patch distance loss L_{dist} but had worse performance. Using both the L_{cp} and L_{lap} loss yielded better predictions as shown in Table 4.2.

- We then jointly train the decomposition and fitting module end-to-end with all the losses. To allow backpropagation from the primitives and B-splines fitting to the

Method	Input	seg iou	label iou	res (all)	res (geom)	res (spline)	P cover
NN	p	54.10	61.10	-	-	-	-
RANSAC	p+n	67.21	-	0.0220	0.0220	-	83.40
SPFN	p	47.38	68.92	0.0238	0.0270	0.0100	86.66
SPFN	p+n	69.01	79.94	0.0212	0.0240	0.0136	88.40
PARSENET	p	71.32	79.61	0.0150	0.0160	0.0090	87.00
PARSENET	p+n	81.20	87.50	0.0120	0.0123	0.0077	92.00
PARSENET + e2e	p+n	82.14	88.60	0.0118	0.0120	0.0076	92.30
PARSENET + e2e + opt	p+n	**82.14**	**88.60**	**0.0111**	**0.0120**	**0.0068**	**92.97**

Table 4.1: **Primitive fitting on** ABCPARTSDATASET**.** We compare PARSENET with nearest neighbor (NN), RANSAC [193], and SPFN [139]. We show results with points (p) and points and normals (p+n) as input. The last two rows shows our method with end-to-end training and post-process optimization. We report 'seg iou' and 'label iou' metric for segmentation task. We report the residual error (res) on all, geometric and spline primitives, and the coverage metric for fitting.

embedding network, the mean shift clustering is implemented as a recurrent module. For efficiency, we use 5 mean-shift iterations during training. It is also important to note that during training, we use *soft* point-to-segment memberships, which enables backpropagation from the fitting module to the decomposition module and improves reconstructions. The soft memberships are computed based on the point embeddings $\{\mathbf{z}_i\}$ (after the mean-shift iterations) and cluster center embedding $\{\mathbf{z}_k\}$ as follows:

$$\mathbf{W}[i,k] = \frac{\exp(\mathbf{z}_k^T \mathbf{z}_i / \beta^2)}{\sum_{k'} \exp(\mathbf{z}_{k'}^T \mathbf{z}_i)/\beta^2)} \tag{4.8}$$

Please see the Appendix for more implementation details.

4.5 Experiments

Our experiments compare our approach to alternatives in three parts: (a) evaluation of the quality of segmentation and segment classification (Section 4.5.1), (b) evaluation of B-spline patch fitting, since it is a major contribution of our work (Section 4.5.2), and (c)

evaluation of overall reconstruction quality (Section 4.5.3). We include evaluation metrics and results for each of the three parts next.

4.5.1 Segmentation and labeling evaluation

4.5.1.0.1 Evaluation metrics. We use the following metrics for evaluating the point cloud segmentation and segment labeling based on the test set of ABCPARTSDATASET:

- **Segmentation mean IOU** ("seg mIOU"): this metric measures the similarity of the predicted segments with ground truth segments. Given the ground-truth point-to-segment memberships $\hat{\mathbf{W}}$ for an input point cloud, and the predicted ones $\mathbf{W}$, we measure:

  $\frac{1}{K}\sum_{k=1}^{K} IOU(\hat{\mathbf{W}}[:,k], h(\mathbf{W}[:,k]))$

  where h represents a membership conversion into a one-hot vector, and K is the number of ground-truth segments.

- **Segment labeling IOU** ("label mIOU"): this metric measures the classification accuracy of primitive type prediction averaged over segments:

  $\frac{1}{K}\sum_{k=1}^{K} \mathcal{I}\left[t_k = \hat{t}_k\right]$ where t_k and $\hat{t}_k$ is the predicted and ground truth primitive type respectively for k^{th} segment and $\mathcal{I}$ is an indicator function.

We use Hungarian matching to find correspondences between predicted segments and ground-truth segments.

4.5.1.0.2 Comparisons. We first compare our method with a nearest neighbor (NN) baseline: for each test shape, we find its most similar shape from the training set using Chamfer distance. Then for each point on the test shape, we transfer the labels and primitive type from its closest point in $\mathcal{R}^3$ on the retrieved shape.

We also compare against efficient RANSAC algorithm [193]. The algorithm only handles basic primitives (cylinder, cone, plane, sphere, and torus), and offers poor reconstruction of B-splines patches in our dataset. Efficient RANSAC requires per point normals, which we

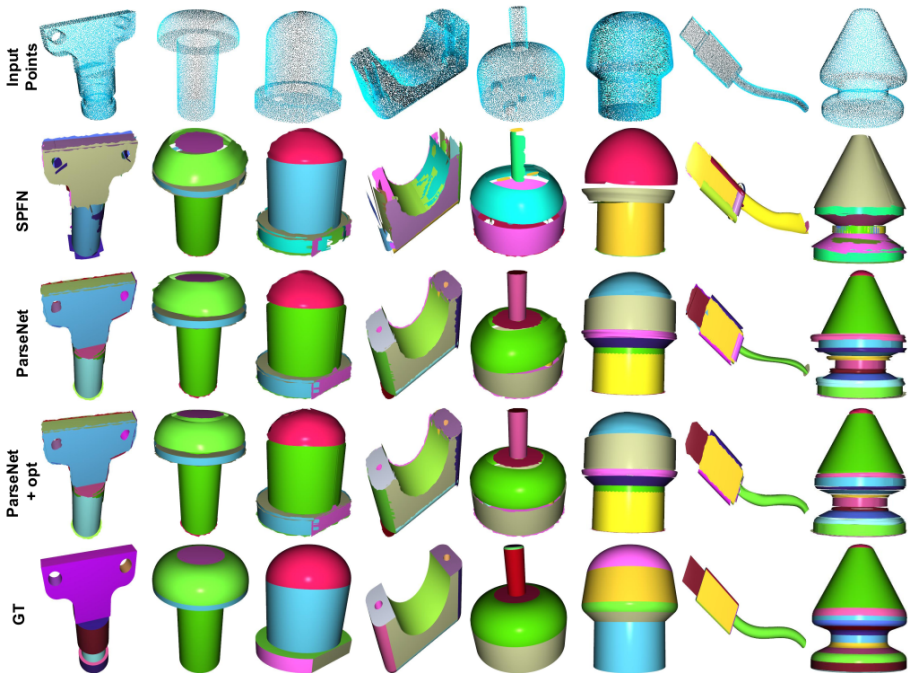


Figure 4.4: Given the input point clouds with normals of the first row, we show surfaces produced by SPFN [139] (second row), PARSENET without post-processing optimization (third row), and full PARSENET including optimization (fourth row). The last row shows the ground-truth surfaces from our ABCPARTSDATASET.

provide as the ground-truth normals. We run RANSAC 3 times and report the performance with best coverage.

We then compare against the supervised primitive fitting (SPFN) approach [139]. Their approach produces per point segment membership, and their network is trained to maximize relaxed IOU between predicted membership and ground truth membership, whereas our approach uses learned point embeddings and clustering with mean-shift clustering to extract segments. We train SPFN network using their provided code on our training set using their proposed losses. We note that we include B-splines patches in their supported types. We train their network in two input settings: (a) the network takes only point positions as input, (b) it takes point and normals as input. We train our PARSENET on our training set in the same two settings using our loss functions.

Loss				Open splines		Closed splines	
cp	dist	lap	opt	w/ ups	w/o ups	w/ ups	w/o ups
✓				2.04	2.00	5.04	3.93
✓	✓			1.96	2.00	4.9	3.60
✓	✓	✓		1.68	1.59	3.74	3.29
✓	✓	✓	✓	**0.92**	**0.87**	**0.63**	**0.81**

Table 4.2: **Ablation study for B-spline fitting.** The error is measured using Chamfer Distance (CD is scaled by 100). The acronyms "cp": control-points regression loss, "dist" means patch distance loss, and "lap" means Laplacian loss. We also include the effect of post-processing optimization "opt". We report performance with and without upsampling ("ups") for open and closed B-splines.

The performance of the above methods are shown in Table 4.1. The lack of B-spline fitting hampers the performance of RANSAC. The SPFN method with points and normals as input performs better compared to using only points as input. Finally, PARSENET with only points as input performs better than all other alternatives. We observe further gains when including point normals in the input. Training PARSENET end-to-end gives 13.13% and 8.66% improvement in segmentation mIOU and label mIOU respectively over SPFN with points and normals as input. The better performance is also reflected in Figure 4.4, where our method reconstructs patches that correspond to more reasonable segmentations compared to other methods. In the Appendix we evaluate methods on the TraceParts dataset [139], which contains only basic primitives (cylinder, cone, plane, sphere, torus). We outperform prior work also in this dataset.

4.5.2 B-Spline fitting evaluation

4.5.2.0.1 Evaluation metrics. We evaluate the quality of our predicted B-spline patches by computing the Chamfer distance between densely sampled points on the ground-truth B-spline patches and densely sampled points on predicted patches. Points are uniformly sampled based on the 2D parameter space of the patches. We use 2K samples. We use the test set of our SPLINEDATASET for evaluation.

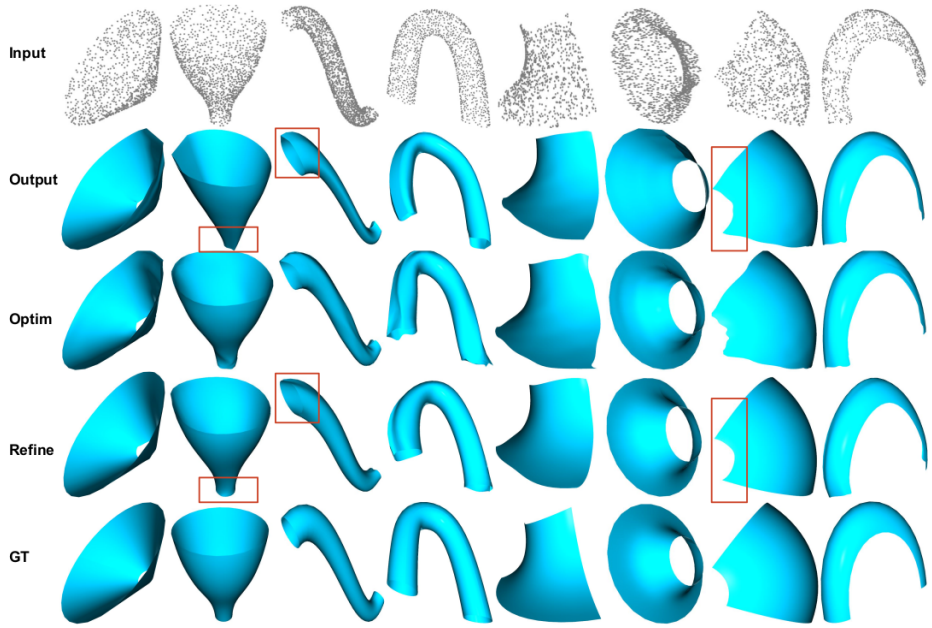


Figure 4.5: **Qualitative evaluation of B-spline fitting.** From top to bottom: input point cloud, reconstructed surface by SPLINENET, reconstructed surface by SPLINENET with post-processing optimization, reconstruction by SPLINENET with control point grid adjustment and finally ground truth surface. Effect of post process optimization is highlighted in red boxes.

4.5.2.0.2 Ablation study. We evaluate the training of SPLINENET using various loss functions while giving 700 points per patch as input, in Table 4.2. All losses contribute to improvements in performance. Table 4.2 shows that upsampling is effective for closed splines. Figure 4.5 shows the effect of optimization to improve the alignment of patches and the adjustment of resolution in the control point grid. See Appendix for more experiments on SPLINENET's robustness.

4.5.3 Reconstruction evaluation

4.5.3.0.1 Evaluation metrics. Given a point cloud $\mathbf{P} = \{\mathbf{p}_i\}_{i=1}^{N}$, ground-truth patches $\{\cup_{k=1}^{K} \hat{\mathbf{s}}_k\}$ and predicted patches $\{\cup_{k=1}^{K} \mathbf{s}_k\}$ for a test shape in ABCPARTSDATASET, we evaluate the patch-based surface reconstruction using the following:

- **Residual error** ("res") measures the average distance of input points from the predicted primitives following [139]: $L_{dist} = \sum_{k=1}^{K} \frac{1}{M_k} \sum_{n \in \hat{\mathbf{s}}_k} D(\mathbf{r}_n, \mathbf{s}_k)$ where K is

the number of segments, M_k is number of sampled points $\mathbf{r}_n$ from ground patch $\hat{\mathbf{s}}_k$, $D(\mathbf{r}_n, \mathbf{s}_k)$ is the distance of $\mathbf{r}_n$ from predicted primitive patch $\mathbf{s}_k$.

- **P-coverage** ("P-cover") measures the coverage of predicted surface by the input surface also following [139]: $\frac{1}{N}\sum_{i=1}^{N} \mathbf{I}\left[\min_{k=1}^{K} D(\mathbf{p}_i, \mathbf{s}_k) < \epsilon\right]$ ($\epsilon = 0.01$).

We note that we use the matched segments after applying Hungarian matching algorithm, as in Section 4.5.1, to compute these metrics.

4.5.3.0.2 Comparisons. We report the performance of RANSAC for geometric primitive fitting tasks. Note that RANSAC produces a set of geometric primitives, along with their primitive type and parameters, which we use to compute the above metrics. Here we compare with the SPFN network [139] trained on our dataset using their proposed loss functions. We augment their per point primitive type prediction to also include open/closed B-spline type. Then for classified segments as B-splines, we use our SPLINENET to fit B-splines. For segments classified as geometric primitives, we use their geometric primitive fitting algorithm.

4.5.3.0.3 Results. Table 4.1 reports the performance of our method, SPFN and RANSAC. The residual error and P-coverage follows the trend of segmentation metrics. Interestingly, our method outperforms SPFN even for geometric primitive predictions (even without considering B-splines and our adaptation). Using points and normals, along with joint end-to-end training, and post-processing optimization offers the best performance for our method by giving 47.64% and 50% reduction in relative error in comparison to SPFN and RANSAC respectively.

4.6 Conclusion

We presented a method to reconstruct point clouds by predicting geometric primitives and surface patches common in CAD design. Our method effectively marries 3D deep learning with CAD modeling practices. Our architecture predictions are editable and

interpretable. Modelers can refine our results based on standard CAD modeling operations. In terms of limitations, our method often makes mistakes for small parts, mainly because clustering merges them with bigger patches. In high-curvature areas, due to sparse sampling, PARSENET may produce more segments than ground-truth. Producing seamless boundaries is still a challenge due to noise and sparsity in our point sets. Generating training point clouds simulating realistic scan noise is another important future direction.

4.6.0.0.1 Acknowledgements. This research is funded in part by NSF (#1617333, #1749833) and Adobe. Our experiments were performed in the UMass GPU cluster funded by the MassTech Collaborative. We thank Matheus Gadelha for helpful discussions.

CHAPTER 5

LEARNING POINT EMBEDDING FROM SHAPE REPOSITORIES FOR FEW SHOT SEMANTIC SEGMENTATION

5.1 Introduction

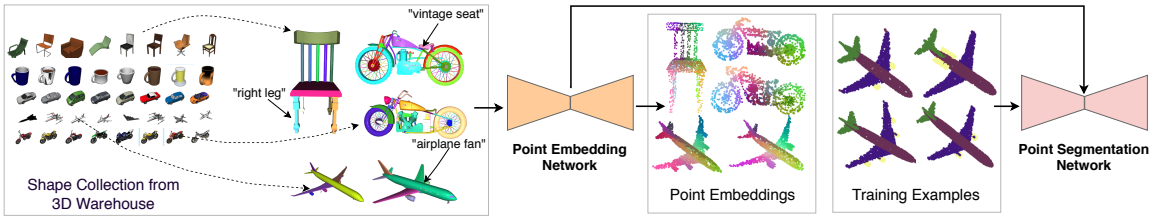


Figure 5.1: **Overview of our approach.** Shape collections in 3D shape repositories contain metadata such as polygon groupings and tags assigned to parts. These parts and tags assigned to them are highly variable, even within the same category. We use the shapes and metadata to train a point embedding network that maps each point into a fixed dimensional vector (see Section 7.3 and Figure 5.3 for details.) The embeddings for a few shapes are visualized as color channels using t-SNE mapping, where similar colors indicate correspondence across shapes. The learned parameters when used to initialize a point segmentation network leads to improved performance when few training examples are available. *(Please zoom in for details.)*

Online repositories of user-generated 3D shapes, such as the 3D Warehouse repository [6], contain rich metadata associated with each shape. These include information about geometric primitives (e.g., polygons in 3D meshes) organized in groups, often arranged in a hierarchy, as well as color, material and tags assigned to them. This information reflects the modeling decisions of the designer are likely correlated with high-level semantics.

Despite its abundance, the use of metadata for learning shape representations has been relatively unexplored in the literature. One barrier is the high degree of its variability. These models were created by designers with a diverse goals and expertise. As a result the groups

and hierarchies over parts of a shape that reflect the modeling steps taken by the designer are highly variable: two similar shapes can have significantly different number of parts as well as the number of levels in the part hierarchy. Moreover, the tags are rarely assigned to parts and are often arbitrarily named. Figures 7.1 and 5.2 illustrate this variability.

Our work systematically addresses these challenges and presents an approach to exploit the information present in the metadata to improve the performance of a state-of-the-art 3D semantic segmentation model. Our approach, illustrated in Figure 7.1, consists of a deep network that maps each point in a 3D shape to a fixed dimensional embedding. The network is trained in a way such that the embedding reflects the user-provided hierarchy and tags. We propose a robust tree-aware metric to supervise the point embedding network that offers better generalization to semantic segmentation tasks over a baseline scheme that is tree-agnostic (only considers the leaf-level groupings). The point embedding network trained on hierarchies also improves over models trained on shape reconstruction tasks that leverage the 3D shape geometry but not their metadata. Finally, when tags are available we show that the embeddings can be fine-tuned leading to further improvements in performance.

On the ShapeNet semantic segmentation dataset, an embedding network pre-trained on hierarchy metadata outperforms a network trained from scratch by reducing relative error by 10.2% across 16 categories, when trained on 8 shapes per category. Similarly, when only a small fraction of points (20 points) per shape are labeled, the relative reduction in error is 4.9%. Furthermore, on 5 categories which have sufficient tags, using both the hierarchy and tags reduces error further by 12.8% points relative to the randomly initialized network, when trained on 8 shapes per category. Our visualizations indicate that the trained networks implicitly learn correspondences across shapes.

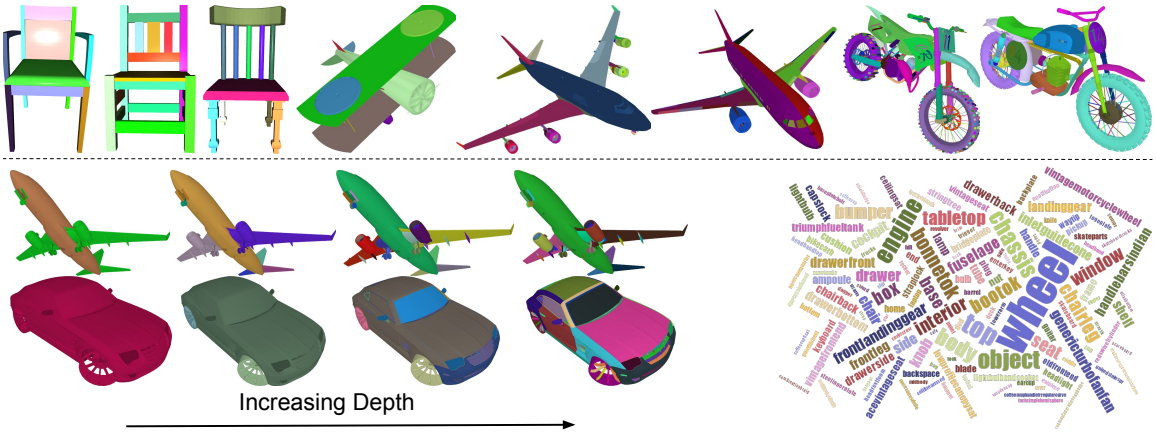


Figure 5.2: (Top row) **Example shapes from our dataset.** Our dataset consist of shapes segmented into parts without any semantic information. Notice that shapes of same category can be segmented differently from each other. Here different color represents different leaf node in the part-hierarchy. (Bottom left) **Parts at different depths of the hierarchy** for an airplane and a car. Increasing the depth increases the number and granularity of parts. (Bottom right) **A word cloud** of raw tags collected from our dataset. The font size is proportional to the square root of frequency of the dataset.

5.2 Related Work

Our work builds on the advances in deep learning architectures for point-based, or local, shape representations and metric learning approaches to guide representation learning. We briefly review relevant work in these areas.

5.2.0.0.1 Supervised learning of local shape descriptors. Several architectures have been proposed to output local representations, or descriptors, for 3D shape points or patches. The architectures can be broadly categorized according to the type of raw 3D shape representation they consume. Volumetric methods learn local patch representations by processing voxel neighborhoods either in uniform [153] or adaptively subdivided grids [121, 184, 233, 234]. View or multi-view approaches learning local image-based representations by processing local 2D shape projections [101, 219], which can be mapped back onto the 3D shape [117]. Finally, a large number of architectures have been recently proposed for processing raw point clouds. PointNet and PointNet++ are transforming individual point coordinates

and optionally normals through MLPs and then performing permutation-invariant pooling operations in local neighborhhoods [178, 179].

All the above-mentioned deep architectures are trained in a fully supervised manner using significant amound of labeled data. Although for some specific classes, like human bodies, these annotations can be easily obtained through template-based matching or synthetically generated shapes [10,13,21], for the vast majorities of shapes in online repositories, gathering such annotations often requires laborious user interaction [158, 254]. Active learning methods have also been proposed to decrease the workload, but still rely on expensive crowdsourcing [254].

5.2.0.0.2 Weak supervision for learning shape descriptors. A few methods [161, 267] have been recently proposed to avoid expensive point-based annotations. Muralikrishnan et al. [161] extracts point-wise representations by training an architecture designed to predict shape-level tags (e.g., armrest chair) by first predicting intermediate shape segmentations. Instead of using weak supervision in the form of shape-level tags, we use unlabeled part hierarchies available in massive online repositories and tags for parts (not whole shapes) when such are available. Yi et al. [253] embeds pre-segmented parts in descriptor space by jointly learning a metric for clustering parts, assigning tags to them, and building a consistent part hierarchy. In our case, our architecture learns point-wise descriptors and also relaxes the requirement of inferring consistent hierarchies, which might be hard to estimate for shape families with significant structural variability. Non-rigid geometric alignment has been used as a form of weak and noisy supervision by extracting approximate local shape correspondences between pairs of shapes of similar structure [107] or by deforming part templates [102]. However, global shape alignment can fail for shapes with different structure, while part-based alignment requires corresponding parts or definition of part templates in the first place. In a concurrent work, given a collection of shapes from a single category, Chen *et al.* [35] proposed a branched autoencoder that discovers coarse segmentations of shapes by predicting implicit fields for each part. Their network is trained with a few manually

selected labeled shapes in a few-shot semantic segmentation setting. Our method instead utilizes part hierarchies and metadata as weak supervisory signal. We also randomly select labeled sets for our few-shot experiments. In general, our method is complementary to all the above-mentioned weak supervision methods. Our weak signal in the form of unlabeled part hierarchies and part tags can be used in conjunction with geometric alignment, consistent hierarchies, or shape-level tags, whenever such are possible to obtain.

5.2.0.0.3 Triplet-based metric learning. Our approach learns a metric embedding over points that reflects the hierarchies in 3D shape collections. Metric learning has a rich literature with a diverse applications and techniques. A popular approach is to supervise the learning with "triplets" of the form (a, b, c) to denote that *"a is more similar to b than c"*. This can be written as $d(a, b) \leq d(a, c)$ where the $d(a, b)$ denotes the distance between a and b. The distance itself could be computed as the Euclidean distance in some embedding space, i.e., $d(a, b) = ||\phi(a) - \phi(b)||_2$, possibly computed with a deep network. Within this framework, techniques to sample triplets remains an active area of research. These include techniques such as hard-negative mining [91], semi-hard negative mining [195] and distance weighted sampling [247] to bias the sampling of triplets.

5.3 Mining Metadata from Shape Repositories

We first describe the source of our part hierarchy dataset that we use for training our embedding network. Then we describe the metadata (tags) present in the 3d models and how we extract this information into a consistent dataset.

5.3.0.0.1 Part hierarchies. Several 3D modeling tools, such as SketchUp, Maya, 3DS Max to name a few, allow users to model shapes, and scenes, in general, as a collection of geometric entities (e.g., polygons) organized into groups. The groups can be nested and organized in hierarchies. In our part hierarchy dataset, we endeavor to extract these hieararchies. The shapes in our dataset are a subset of Shapenet Core dataset, where we

Category	Shapes with part tagged	Avg points tagged
Motorcycle	110	11.3%
Airplane	806	5.0%
Table	392	45.7%
Chair	326	38.7%
Car	600	20.0%

Table 5.1: **Dataset with tags.** Number of shapes with at least one tagged parts, and average percentage of points tagged in these shapes in 5 categories.

focus on 16 categories from Shapenet part-segmentation dataset [254] to allow systematic benchmarking and comparison with prior work. Note that the 16 categories semantic segmentation dataset contains $16.6k$ shapes, whereas 16 categories in Shapenet Core dataset contains $28k$ shapes. We first retrieved the original files for shapes in Shapenet Core dataset provided by 3d warehouse, which are stored in the popular "COLLADA" format [243]. These files represent 3D models in a hierarchical tree structure. Leaf nodes represent shape geometry, and internal nodes represent groups of geometric primitives, or nested groups. Samples from our dataset are visualized in the Figure 5.2. Number of parts in which a shape is segmented depends on the part-hierarchy as visualized in the Figure 5.2 (bottom left). Models with too few part segmentation (less than 2) or too many (more than 500) are discarded. This gives us a total of 20776 3D models having part group information, with each model having at least one level of part grouping. We further segment the dataset into train (15625), validation (3113) and test (2038) splits. We ensure that the shapes in test split of semantic part-segmentation dataset [254] are not included in the train split of our part hierarchy dataset.

5.3.0.0.2 Tag extraction. Modeling tools allow users to explicitly give tags to parts, which are stored in their corresponding file format. Obviously, not all designers enter tags for their designed parts. Out of all the models that include part group information in our dataset, we observed that only 10.7% of the shapes had meaningful tags for at least one part (i.e., tags are sparse). Usually, these tags are not consistent, e.g., a tag for a wheel part in a car can be

"wheel_mesh". To make things worse, few tags have high frequency e.g., one may encounter wheel, chassis, windows (or synthetics of those) frequently as tags, while most of them are rare, or even be non-informative for part types e.g., "geometry123".

To extract meaningful tags, we selected the 10 most frequent tags encountered as strings, or sub-strings stored in the nodes for each shape category. We also merge synonyms into one tag to reduce number of tags in the final set. For every tag, we find the corresponding geometry nodes and then we label the points sampled from these nodes with the tag. We found that only 5 out of 16 categories have a "sufficient" number of tagged points ($> 1\%$ of the original surface points). By "sufficient", we mean that below this threshold, tags are becoming so sparse in a category that result in negligible improvements. Table 5.1 shows the distribution of tags in these 5 categories.

5.3.0.0.3 Geometric postprocessing. We finally aligned the shapes using ICP so that their orientation agrees with the canonical orientation provided for the same shapes in ShapeNet. To process the shapes through our point-based architecture, we uniformly sampled 10K points on their surface. Further details about these steps are provided in the Appendix.

5.4 Method

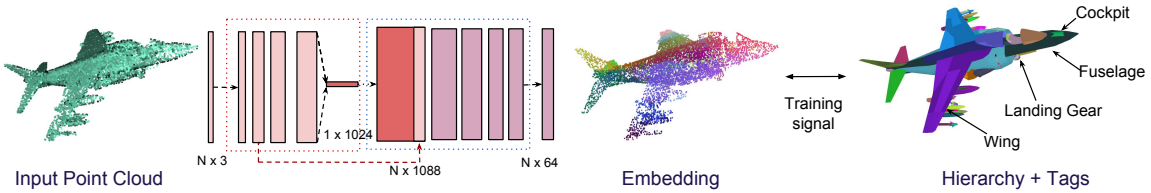


Figure 5.3: **Architecture of the Point Embedding Network (PEN)**. The network takes as input a point cloud and outputs a fixed dimensional embedding for each point, visualized here using t-SNE. These embeddings are learned using metric learning that utilizes part-hierarchy. Furthermore, embedding can be improved by supervising network using sparsely tagged point cloud from a small subset of our dataset (refer Table 5.1). Tags are pointed by arrows.

Our Point Embedding Network (PEN) takes as input a shape in the form of a point cloud set, $X = \{\mathbf{x}_i\}_{i=1}^N$, where $\mathbf{x}$ represents the 3D coordinates of each point. Our network learns to map each input shape point $\mathbf{x}$ to an embedding $\phi_{\mathbf{w}}(\mathbf{x}) \in \mathcal{R}^d$ based on learned network parameters $\mathbf{w}$. The architecture is illustrated in Figure 5.3. PEN first incorporates a PointNet module [178]: the points in the input shape are individually encoded into vectorial representations through MLPs, then the resulting point-wise representations are aggregated through max pooling to form a global shape representation. The representation is invariant to the order of the points in the input point set. At the next stage, the learned point-wise representations are concatenated with the global shape representation, and are further transformed through fully-connected layers and ReLUs. In this manner, the point embeddings reflect both local and global shape information.

We used PointNet as a module to extract the initial point-wise and global shape representation mainly due to its efficiency. In general, other point-based modules, or even volumetric [153, 184, 233] and view-based modules [101, 212] for local and global shape processing could be adapted in a similar manner within our architecture. Below we describe the main focus of our work to learn the parameters of the architecture based on part hierarchies and tag data.

5.4.0.0.1 Learning from part hierarchies. Our training takes a standard metric learning approach where the parameters of the PEN are optimized such that pairs originating from the same part sampled from the hierarchy (positive pairs) have distance smaller than pairs of points originating from different parts (negative pairs) in the embedded space. Specifically, given a triplet of points (a, b, c), the loss of the network over this triplet [93] is defined as:

$$\ell(a, b, c) = \big[d(a, b) - d(a, c) + m\big]_+, \tag{5.1}$$

where $d(a, b) = \big\|\phi_w(a) - \phi_w(b)\big\|_2^2$, m is a scalar margin, and $[x]_+ = \max(0, x)$. To avoid degenerate solutions we constrain the embeddings to lie on a unit hypersphere, i.e.,

$\left\|\phi(x)\right\|_2^2 = 1, \forall x$. Given a set of triplets $\mathcal{T}_s$ sampled from each shape s from our dataset S, the triplet objective of the PEN is to minimize the triplet loss:

$$L_{triplet} = \sum_{s \in S} \frac{1}{|\mathcal{T}_s|} \sum_{(a,b,c) \in \mathcal{T}_s} \ell(a, b, c). \tag{5.2}$$

5.4.0.0.2 Sampling triplets. One simple strategy to sample triplets is to just access the parts at the finest level of segmentation, then sample triplets by randomly taking fixed number of similar pairs (a, b) from the same part and an equal number of negative points c from another part. We call this strategy "leaf" triplet sampling.

An alternative strategy is to consider the part hierarchy tree for triplet sampling. Here, we sample negative point pairs depending on the tree distance between the part groups (tree nodes) they belong to. Given two nodes n_i and n_j, we use the sum of path lengths (number of tree edges) from nodes n_i and n_j to their lowest common ancestor as the tree distance $\delta(n_i, n_j)$ [244] . For example, if the two nodes are siblings (i.e., two parts belonging to the same larger group), then their lowest common ancestor is their parent and their tree distance is equal to 2 (i.e., count two edges that connect them to their parent). If two nodes are further away in the hierarchy, then tree distance increases. In this manner, the tree distance reflects how far two nodes (parts) are in the hierarchy.

We compute the probability of selecting the positive pair of points from node n_i and the negative pair using the point from another node n_j as follows:

$$P(n_i, n_j) \propto \frac{1}{\delta(n_i, n_j)} \tag{5.3}$$

Sampling points in this way yields more frequent triplets that consist of negative pairs closer in the hierarchy. Parts that are closer in the hierarchy tend to be spatially or geometrically closer to each other, thus also harder to discriminate. We call this sampling strategy as "hierarchy" triplet sampling. We discuss the effect of these two strategies in the experiments section.

5.4.0.0.3 Learning from noisy tag data. We can also utilize tag data for segments collected from the COLLADA files, as described in Section 5.3. To train the network using tags, we add two pointwise fully-connected layers on top of the embedding network (PEN). One way to train this network is to define a categorical cross entropy over points whose parts are tagged. However, as shown in Table 5.1, the total number of tagged points is small. We instead found that a better strategy is to use a one-vs-rest binary cross entropy loss to also make use of points in un-tagged parts. The reason is that if a part is not tagged in a shape that has other parts labeled with tags existing in the shape metadata, then most likely, that part should not be labeled with any of the existing tags for that shape (e.g., if a car has tagged parts as 'wheel' and 'window', then other un-tagged parts should most likely not be assigned with these tags).

More specifically, for every tag in our tag set $\mathcal{L}$ for a shape category, we define a binary cross entropy loss by considering all points assigned with that tag as 'positive' (set $\mathcal{P}$) while the rest of points assigned with other or no tags as 'negative' (set $\mathcal{N}$). Given an output probability prediction for assigning a point i with tag t, denoted as $P(y_{i,t} = 1)$ produced by the last classification layer (sigmoid layer) of our network, our loss function over tags is defined as follows:

$$L_{tag} = -\sum_{t \in \mathcal{T}} \left(\sum_{i \in \mathcal{P}} \log P(y_{i,t} = 1) + \sum_{i \in \mathcal{N}} \log(1 - P(y_{i,t} = 1)) \right) \tag{5.4}$$

5.4.0.0.4 Training. We first train our network to minimize the triplet loss $L_{triplet}$ based on our dataset of shapes that contains part hierarchies. Training is done in a cross-category manner on 16 categories[1] of ShapenetCore dataset, as described in Section 5.3. We use the Adam optimizer [120] with initial learning rate of 0.01 decayed by the factor of 10 whenever the triplet loss stops decreasing over validation set. The mini-batches consist of 32 shapes. For further efficiency, in each iteration we randomly sample a subset of $2.5k$ points (from

[1]These are the same 16 categories present in Shapenet semantic segmentation dataset from Yi et al. [254]

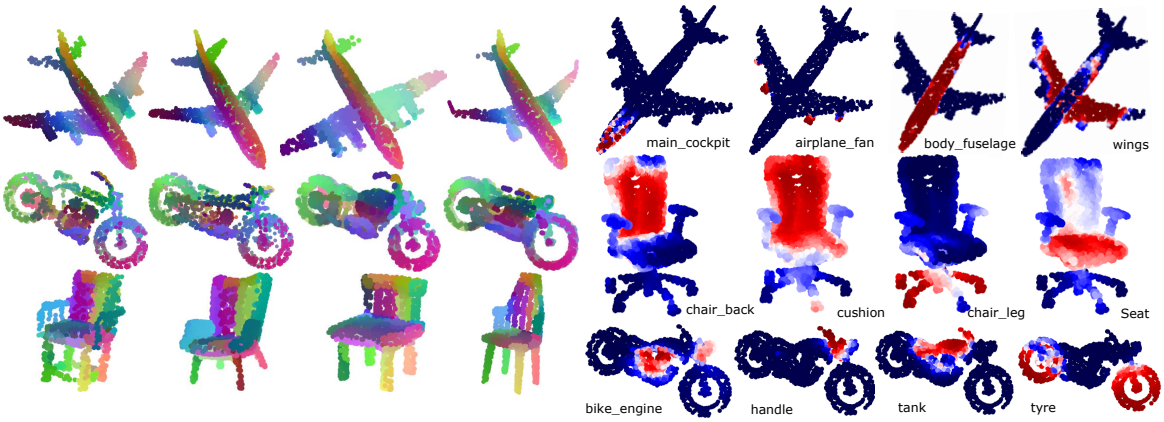


Figure 5.4: **Visualization of the embeddings.** (Left) T-SNE visualization of embedding shown as a color map. Embeddings for similar semantic parts are consistently embedded close to each other as reflected by the similarity in their color. (Right) Heat map visualization of tags predictions across a number of categories and tags. Redder values indicate a higher probability of the tag. *(Best seen magnified.)*

the $10K$ original points) for each shape during training. The total number of triplets sampled from a shape is kept constant.

Then for the 5 categories that include tags, we further fine-tune the learned embeddings by learning the two additional pointwise fully-connected layer with a Sigmoid at the end to minimize the tag loss L_{tag}. Since tags are distinct for each category, fine-tuning is done in a category-specific manner (i.e., we produce a different embedding specialized for each of these 5 categories). Although the triplet and tag loss could be combined, we choose a stage-wise training approach since the shapes with part hierarchies are significantly more numerous than the shapes that include tags as shown in Table 5.1. In our experiments we discuss the effect of training only with the triplet loss, and also the effect of fine-tuning with the tag loss in each category.

For training networks on few-shot learning task, we do hyper-parameters (batch size, epochs, regularization etc.) search using validation set of only one category ('airplane') and use the same hyper-parameters setting to train all models on all categories in the few-shot learning task.

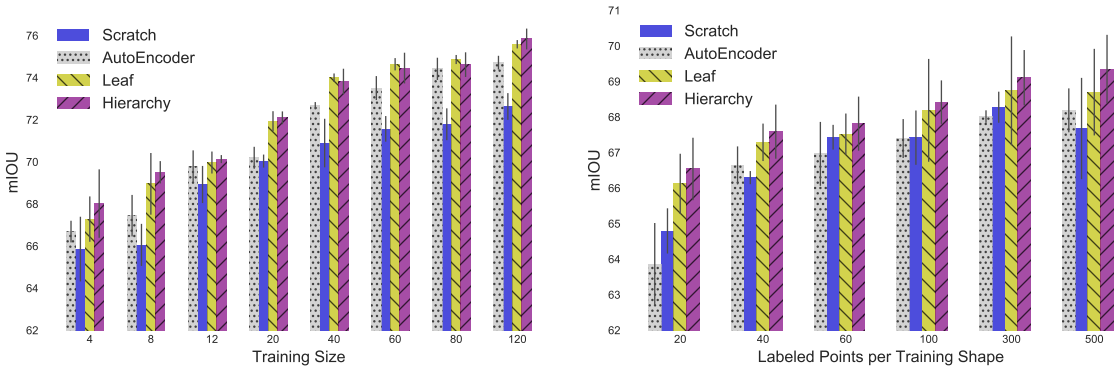


Figure 5.5: **Benifits of pretraining PEN using metric learning. Left:** mIoU evaluation for varying number of training shapes. **Right:** mIoU evaluation for varying number of labeled points and fixing the number of training shapes to 8. We compare different baselines and variants of PEN, including training from scratch, autoencoder for pre-training, as well as PEN trained with metric learning triplets sampled from the leaf of the tree (Leaf) or based on the hierarchy (Hierarchy). PEN outperforms both baselines with the hierarchy-based sampling offering a slight advantage over the leaf-based one.

5.4.0.0.5 Few-shot learning. Given our network pre-trained on our shape datasets based on part hierarchies and/or tags, we can further train it on other, much smaller, datasets of shapes that include semantic part labels. To do this, once again we add two point-wise fully-connected layers on top of the embedding layer, and a softmax layer to produce semantic part label probabilities. In our experiments, we observe that the part labeling performance is significantly increased when compared to training our network from scratch using semantic part labels only as supervision.

5.4.0.0.6 Implementation details. In our implementation, the encoder of our network extracts 1024-dimensional global shape embedding. The decoder concatenates the global embedding with $64d$ point features from encoder, and finally transform it into a 64-dimensional point-wise embeddings. Further details of the layers used in PEN are discussed in the Appendix. Our implementation is based on PyTorch[2].

[2]https://pytorch.org

5.5 Results

We now discuss experiments performed to evaluate our method and alternatives. First, we present qualitative analysis of learned embeddings, then we discuss a new benchmark we introduce for few-shot segmentation and evaluation metrics, and finally we present results and comparisons of our network with various baselines.

5.5.0.0.1 Visualization of the embeddings. We first present a qualitative analysis of the PEN embeddings. The embeddings learnt using metric learning only (without the tag loss) are visualized in Figure 5.4 (left). We use the t-SNE algorithm to embed the 64-dimensional point embedding in $3D$ space. Interestingly, the descriptors produced by PEN consistently embed the points belonging to similar parts close to each other without explicit semantic supervision. We also visualize the embeddings predicted by PEN trained with metric learning and fine-tuned with tag loss in Figure 5.4 (right). The embeddings have better correspondence with the tags. Interestingly, the network predicts correct embeddings for points with tags that are not mutually exclusive e.g. 'cushion' and 'back' of the chair.

5.5.0.0.2 Few-shot Segmentation Benchmark. We anticipate that learning from metadata can improve semantic shape segmentation tasks, especially in the few-shot learning scenario. To this end we have created a new benchmark on ShapeNet segmentation dataset [254], in which we randomly select x fully labeled examples from the complete training set for training the network, where $x \in \{4, 8, 12, 20, 40, 60, 120\}$. In this manner, we can test the behaviour of methods with increasing training number of shapes, starting with the few-shot scenario where only a handful of shapes (i.e., 4 or 8) is labeled. The performance is measured as the mean intersection over union (mIOU) across all part labels and shapes in the test splits. We exclude the shapes existing in our part hierarchy and tag datasets used for pre-training PEN from the test splits.

We also introduce one more evaluation setting, where for each shape category, the training shapes have smaller fractions of their original points labeled (20, $40 \ldots 500$) labeled points compared to the original $2.5K$ points) The case of $\sim$ 20-40 labeled point simulates

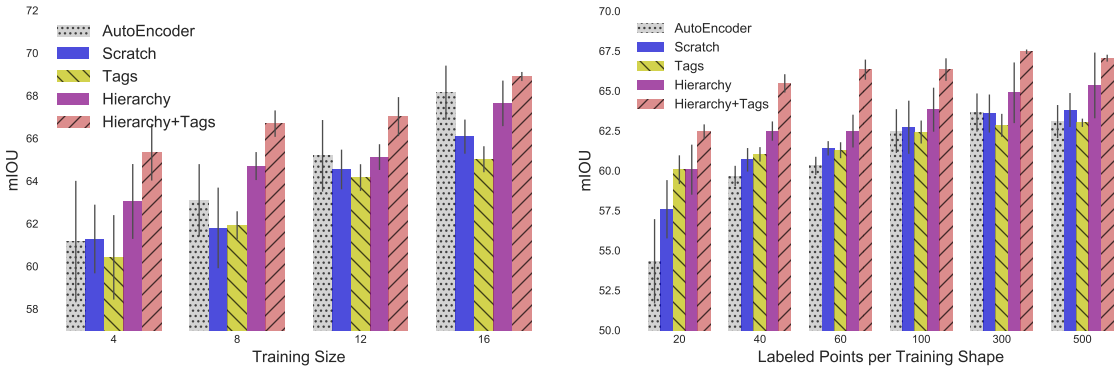


Figure 5.6: **Benefit of training with tag supervision.** The mIoU evaluation when tags are available (5 categories: motorcycle, airplane, table, chair, car). We include the same baselines and PEN variants as Figure 5.5, including two more PEN variants: one trained with tags only (Tags) and another trained both on hierarchy and tags (Hierarchy + Tags). **Left:** Shows the performance in the few-shot setting. **Right:** Shows the performance in the few-point setting. In both cases the tag data (Hierarchy + Tags) provides additional benefits over the PEN models trained with the hierarchy supervision (Hierarchy). Tag data alone is not as effective as the autoencoder since the supervision is highly sparse.

the scenario where semantic annotations are collected through sparse user input (e.g., click few points on shapes and label them).

5.5.0.0.3 Baselines. Since we utilize a vast number of unlabeled data from the same domain it is important to compare with baselines. Our first baseline simply trains PEN from scratch on the training splits of our few-shot segmentation benchmark using only semantic label supervision (without using metadata). Second, we also compare with another baseline, where we train an autoencoder network that leverages only geometry as an alternative to produce point-wise embeddings. This network first encodes the input point cloud to point-wise embeddings producing a 1024-dimensional point-wise representations exactly as in PEN, then a decoder uses upconvolution to reconstruct the original point cloud. The Chamfer distance between generated points and input points is used as a loss function to train this network. We first pre-train the autoencoder on the shapes included in our part hierarchy dataset. After this pre-training step, we retain the encoder and replace the geometry decoder with PEN's decoder and add two pointwise fully connected layers and a classification layer to produce semantic part label probabilities. The resulting network is then trained in stages,

first the decoder and then the entire network at smaller learning rate, on the training splits of our few-shot segmentation benchmark.

Finally, we also evaluate the two strategies to pretrain the embedding network using different triplet sampling techniques *i.e.* leaf-level shape parts ("leaf" triplet sampling) and based on using the hierarchy tree ("hierarchy" triplet sampling) as described in (Section 7.3).

Next, we compare the performance of our method with the baselines and different sampling strategies under the scenario of using only the triplet loss and cross-category training. Then, we discuss the performance in the case where we additionally use the tag loss.

5.5.0.0.4 Few-shot Segmentation Evaluation. In Figure 5.5 (left), we plot the mIOU of the baselines along with our method. The plotted mIOU is obtained by taking the average of the mIOU on our test splits over all categories and repeating each experiment 5 times. The network trained from scratch (without any pre-training) has the worst performance. The network based on the pre-trained autoencoder shows some improvement since its point-wise representations reflect local and global geometric structure for the point cloud reconstruction, which can be also relevant to the segmentation task. Our method consistently outperforms the baselines. In particular, the "hierarchy" triplet sampling that uses the part hierarchy trees to choose triplets for training our network performs the best on average. A 3.5% mIOU improvement (10.2% drop in relative error) is observed compared to training from scratch at 8 training examples - interestingly, the improvement is retained even for 120 training examples. The "hierarchy" triplet sampling also improves over the "leaf" triplet sampling until 20 training examples, then their difference gap between these two strategies is closed.

5.5.0.0.5 Evaluating with limited labeled points per shape. In the previous section we observed the performance of our method and baselines by changing the number of training shapes. Here we also examine the performance in the few-shot setting where we keep the number of training shapes fixed and vary the number of labeled points per training shape. We

retrain the above baselines (train from scratch, autoencoder) and triplet sampling strategies ("leaf" and "hierarchy") with 8 training examples, and vary the number of labeled points as shown in the Figure 5.5 (right). Again our network using the "hierarchy" triplet sampling performs better than the baselines. It offers 1.7% better mIOU (4.9% drop in relative error) compared to training from the scratch using 20 labeled points.

5.5.0.0.6 Are tags useful? Here we repeat the two few-shot sementation tasks on 5 shape categories (motorcycle, airplane, table, chair, car) that include some tagged parts in their shape metadata. Here, we examine two more PEN variants: (a) PEN pre-trained using the tag loss only (no triplet loss), then fine-tuned on the training splits of our semantic segmentation benchmark (this baseline is simply called "tags" network), 2) our network pre-trained using triplets loss based on the "hierarchy" sampling, then fine-tuned with the tag loss, and finally further fine-tuned on the training splits of our semantic segmentation benchmark (this baseline is called "Hierarchy+Tags" network). The two PEN variants are trained per each category of the 5 categories. The results are shown in Figure 5.6.

When using 8 training examples, the Hierarchy+Tags network offers 4.8% better mIOU (12.8% drop in relative error) on average compared to training from scratch in these 5 categories (refer Figure 5.6 (left)). An improvement of 2.8% mIOU (8.3% drop in relative error) is maintained for 16 training examples. Similarly, when using 20 labeled points per shape, Hierarchy+Tags performs 4.9% mIOU better (11.47% drop in relative error) than training from scratch (refer Figure 5.6 (right)). In general, the Hierarchy+Tags PEN variant outperforms all other baselines (training from scratch, autoencoder) and also the variant pre-trained using tags only ("Tags" network) on both evaluation settings with limited number of training shapes and limited number of training points. This shows that the combination of pre-training through metric learning on part hierarchies and fine-tuning using tags results in a better, warm starting model for semantic segmentation task.

5.6 Conclusion

We presented a method to exploit existing part hierarchies and tag metadata associated with 3D shapes found in online repositories to pre-train deep networks for shape segmentation. The trained network can be used to "warm start" a model for semantic shape segmentation, improving performance in the few-shot setting. Future directions include investigating alternative architectures and combining other types of metadata, such as geometric alignment or material information.

Acknowledgements. This research is funded by NSF (CHS-161733 and IIS-1749833). Our experiments were performed in the UMass GPU cluster obtained under the Collaborative Fund managed by the Massachusetts Technology Collaborative.

CHAPTER 6

REPRESENTATION LEARNING BY SURFACE FITTING

6.1 Introduction

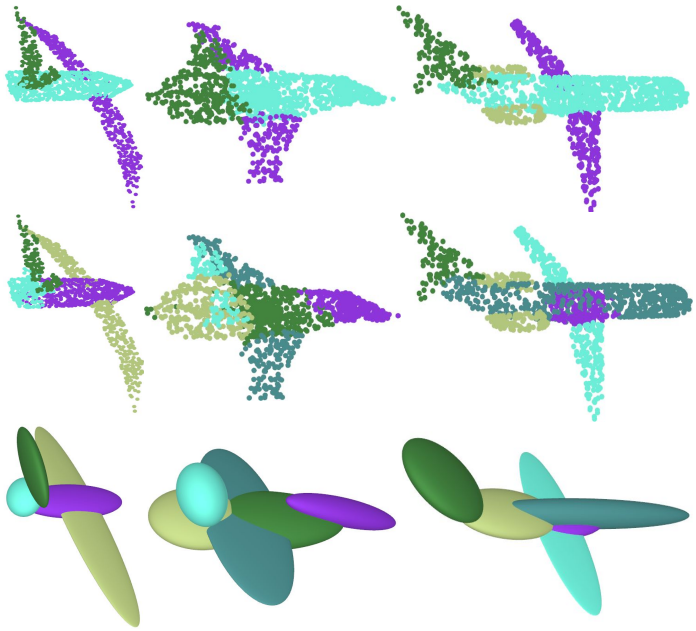


Figure 6.1: **SURFIT uses primitive fitting within a semi-supervised learning framework for learning 3D shape representations.** *Top row:* 3D shapes represented as point clouds, where the color indicates the parts such as wings and engines. The induced partitions and shape reconstruction obtained by fitting ellipsoids to each shape using our approach are shown in the *middle row* and *bottom row* respectively. The induced partitions often have a significant overlap with semantic parts.

In Chapter 5 we proposed a self-supervision approach that utilized part-hierarchies present in CAD models to learn per-point features for 3D shapes. These part-hierarchies are often inconsistent as they vary depending on the expertise and goals of the designers creating models. The goal of this work is to avoid reliance on pre-built inconsistent part-hierarchies

and decomposition of shapes, and discover these decomposition via primitive-fitting. To this end we present SURFIT, a *semi-supervised* approach for learning 3D point-based representations guided by 3D shape decomposition into geometric primitives. Our approach exploits the fact that parts of 3D shapes are often aligned with simple geometric primitives, such as ellipsoids and cuboids. Even if these primitives capture 3D shapes at a rather coarse level, the induced partitions provide a strong prior for learning point representations useful for part segmentation networks, as seen in Fig. 6.1. This purely geometric task allows us to utilize vast amounts of unlabeled data in existing 3D shape repositories to guide representation learning for part segmentation. We show that the resulting representations are especially useful in the few-shot setting, where only a few labeled shapes are provided as supervision.

The overall framework for SURFIT is based on a *point embedding* module and a *primitive fitting* module, as illustrated in Fig. 6.2. The point embedding module is a deep network that generates per-point embeddings for a 3D shape. Off-the-shelf networks can be used for this purpose (e.g., PointNet++ [179], DGCNN [240]). The primitive fitting module follows a novel iterative clustering and primitive parameter estimation scheme based on the obtained per-point embeddings. It is fully differentiable, thus, the whole architecture can be trained end-to-end. The objective is to minimize a reconstruction loss, computed as the Chamfer distance between the 3D surface and the collection of fitted primitives. We experimented with various geometric primitives, including ellipsoids or cuboids due to their simplicity. We also considered parameterized geometric patches based on an Atlas [78], as an alternative surface primitive representation.

Our method achieves 63.4% part IoU performance in ShapeNet segmentation dataset [28] with just one labeled example per-class, outperforming the prior state-of-the-art [64] by 1.6%. We also present results on the PartNet dataset [159] where our approach provides 2.1% improvement compared to a baseline approach while using 10 labeled examples per-class. We also perform extensive analysis of the impact of various design choices and

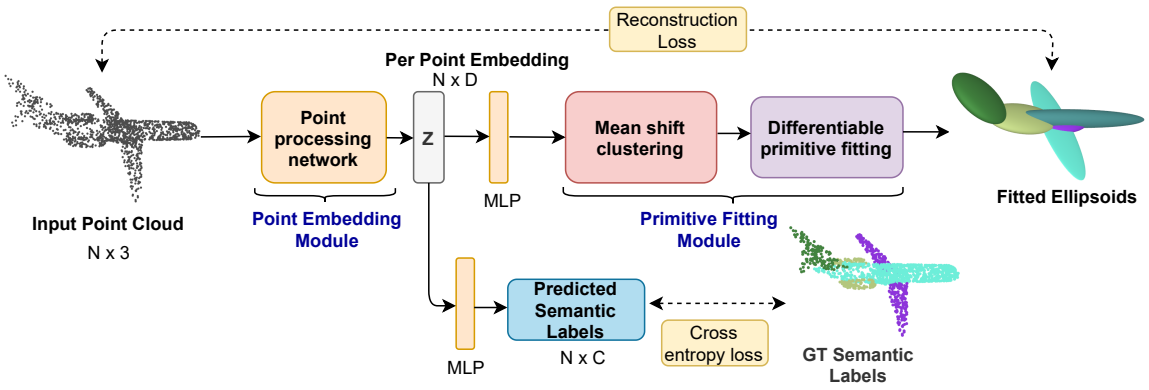


Figure 6.2: **Overview of SURFIT.** Given a point cloud, the *point-embedding module* outputs a feature representation for each point. This is processed through the *primitive-fitting module*, that uses mean-shift clustering to cluster the points and fit a geometric primitive to each cluster. We train the network with a reconstruction loss between the fitted primitives and the input point cloud over the unlabeled shapes, and a categorical cross entropy loss over a small number of labeled shapes.

primitive types on the resulting shape segmentations. Our experiments indicate that the use of ellipsoids as geometric primitives provide the best performance, followed by cuboids, then AtlasNet patches.

6.2 Related Work

We are interested in learning per-point representations of 3D shapes in a semi-supervised manner given a large number of unlabeled shapes and only a few labeled examples. To this end, we briefly review the literature on geometric primitive fitting and shape decomposition, few-shot learning, and deep primitive fitting. We also discuss the limitations of prior work and how we address them.

Geometric primitives and shape decomposition. Biederman's recognition-by-components theory [18] attempts to explain object recognition in humans by the ability to assemble basic shapes such as cylinders and cones, called *geons*, into the complex objects encountered in the visual world. Early work in cognitive science [94] shows that humans are likely to decompose a 3D shape along regions of maximum concavity, resulting in parts that tend to

be convex, often referred to as the "*minima rule*". Classical approaches in computer vision have modeled 3-D shapes as a composition of simpler primitives, *e.g.* work by Binford [19, 20] and Marr [151]. More recent work in geometric processing has developed shape decomposition techniques that generate different types of primitives which are amenable to tasks like editing, grasping, tracking and animation. Those have explored primitives like 3D curves [66, 74, 154], cages [249], sphere-meshes [223], generalized cylinders [266], radial basis functions [25, 148] and simple geometric primitives [192]. This motivates the use of our geometric primitive fitting as a self-supervised task for learning representations.

Unsupervised learning for 3D data. Several previous techniques have been proposed to learn 3D representations without relying on extra annotations. Many such techniques rely on reconstruction approaches [65, 78, 250, 251, 265]. FoldingNet [251] uses an auto-encoder trained with permutation invariant losses to reconstruct the point cloud. Their decoder consists of a neural network representing a surface parametrized on a 2D grid. AtlasNet [78] proposes using several such decoders that result in the reconstructed surface being represented as a collection of surface patches. Li *et al.* [137] presents SO-Net that models spatial distribution of point cloud by constructing a self-organizing map, which is used to extract hierarchical features. The proposed architecture trained in auto-encoder fashion learns representation useful for classification and segmentation. Chen *et al.* [36] proposes an auto-encoder with multiple branches, where each branch is used to reconstruct the shape by producing implicit fields instead of point clouds. However, this requires one decoder for separate part, which restricts its use to category-specific models. Several techniques also proposed models for generating implicit functions from point clouds [44, 70, 155], but it is unclear how well the representations learned by those methods perform in recognition tasks.

Several works use reconstruction losses along with other self supervised tasks. Hassani *et al.* propose multiple tasks: reconstruction, clustering and classification to learn point representation for shapes. Thabet *et al.* [222] propose a self-supervision task of predicting the

next point in a space filling curve (Morton-order curve) using RNN. The output features from the RNN are used for semantic segmentation tasks. Several works have proposed learning point representation using noisy labels and semantic tags available from various shape repositories. Sharma *et al.* [203] learn point representations using noisy part hierarchies and designer-labeled semantic tags for few-shot semantic segmentation. Muralikrishnan *et al.* [161] design a U-Net to learn point representations that predicts user-prescribed shape-level tags by first predicting intermediate semantic segmentation. More recently, Xie *et al.* [189] learn per-point representation for 3D scenes, where point embeddings of matched points from two different views of a scene are pushed closer than un-matched points under a contrastive learning framework.

Closely related to our work, Gadelha et al. [64] use approximate convex decomposition of watertight meshes as source of self-supervision by training a metric over point clouds that respect the given decomposition. Our approach directly operates on point clouds and integrates the decomposition objectives in a unified and end-to-end trainable manner. Empirically we observe that this improves performance. It also removes the need for having a black-box decomposition approach that is separated from the network training.

Semi-supervised learning for 3D data. Similar to our approach, Alliegro *et al.* [11] use joint supervised and self-supervised learning for learning 3D shape features. Their approach is based on solving 3D jigsaw puzzles as a self-supervised task to learn shape representation for classification and part segmentation. Wang *et al.* [232] proposed semi-supervised approach that aligns a labeled template shape to unlabeled target shapes to transfer labels using learned deformation.

Deep primitive fitting. Several approaches have investigated the use of deep learning models for shape decomposition. Their common idea is to learn point-level representations used to generate primitives. Several primitive types have been proposed, including superquadrics [173, 174], cuboids [63, 225] and radial basis functions [70]. However, all these approaches have focused on *generative* tasks with the goal of editing or manipulating a

3D shape. Our insight is that reconstructing a shape by assembling simpler components improves representation learning for *discriminative* tasks, especially when only a few labeled training examples are available.

6.3 Method

Our method assumes that one is provided with a small set of *labeled* shapes $\mathcal{X}_l$ and a large set of *unlabeled* shapes $\mathcal{X}_u$. Each shape $X \in \{\mathcal{X}_l, \mathcal{X}_u\}$ is represented as a point cloud with N points, i.e., $X = \{x_i\}$ where $x_i \in \mathbb{R}^3$. The shapes in $\mathcal{X}_l$ additionally come with part label $Y = \{y_i\}$ for each point. In our experiments we use the entire set of shapes from the ShapeNet core dataset [28] and few labeled examples from the ShapeNet semantic segmentation dataset and PartNet dataset.

The architecture of SURFIT consists of a *point embedding* module Φ and a *primitive fitting* module Ψ. The point embedding module $\Phi(X)$ maps the shape into embeddings corresponding to each point $\{\Phi(x_i)\} \in \mathbb{R}^D$. The primitive fitting module Ψ maps the set of point embeddings to a set of primitives $\{P_i\} \in \mathcal{P}$. Thus $\Psi \circ \Phi : X \to \mathcal{P}$ is a mapping from point clouds to primitives. In addition the point embeddings can be mapped to point labels via a classification function Θ and thus, $\Theta \circ \Phi : X \to Y$. We follow a *joint training* approach where shapes from $\mathcal{X}_l$ are used to compute a supervised loss and the shapes from $\mathcal{X}_u$ are used to compute a self-supervised loss for learning by minimizing the following objective:

$$\min_{\Phi,\Psi,\Theta} \mathcal{L}_{\mathsf{ssl}} + \mathcal{L}_{\mathsf{sl}}, \text{ where} \tag{6.1}$$

$$\mathcal{L}_{\mathsf{ssl}} = \mathop{\mathbb{E}}_{X \sim \mathcal{X}_u} \Big[\ell_{\mathsf{ssl}}\big(X, \Psi \circ \Phi(X)\big) \Big], \text{ and} \tag{6.2}$$

$$\mathcal{L}_{\mathsf{sl}} = \mathop{\mathbb{E}}_{(X,Y) \sim \mathcal{X}_l} \Big[\ell_{\mathsf{sl}}\big(Y, \Theta \circ \Phi(X)\big) \Big]. \tag{6.3}$$

Here ℓ_{ssl} is defined as a *reconstruction error* between the point cloud and a set of primitives, while ℓ_{sl} is the cross entropy loss between predicted and true labels. We describe the details

of the point embedding module in Sec.6.3.1 and the primitive fitting module in Sec. 6.3.2. Finally in Sec. 6.3.3 we describe various loss functions used to train SURFIT.

6.3.1 Point embedding module

This module produces an embedding of each point in a point cloud. While any point cloud architecture [178, 179, 240] can be used, we experiment with PointNet++ [179] and DGCNN [240], two popular architectures for point cloud segmentation. These architectures have also been used in prior work on few-shot semantic segmentation making a comparison easier.

6.3.2 Primitive fitting module

The primitive fitting module is divided into a *decomposition* step that groups the set of points into clusters in the embedding space, and a *fitting* step that estimates the parameters of the primitive for each cluster.

Decomposing a point cloud. These point embeddings $\Phi(x_i) \in \mathbb{R}^D$ are grouped into M clusters using a differentiable mean-shift clustering. The motivation behind the choice of mean-shift over other clustering approaches such as k-means is that it allows the number of clusters to vary according to a kernel bandwidth. In general we expect that different shapes require different number of clusters. We use recurrent mean-shift updates in a differentiable manner as proposed by [127]. Specifically, we initialize seed points as $Y^{(0)} = Z \in \mathbb{R}^{N \times D}$ and update them as follows:

$$Y^{(t)} = KZD^{-1} \tag{6.4}$$

We use the von Mises-Fisher kernel [150] $K = \exp(Y^{(t-1)}Z^T/b^2)$, where $D = \texttt{diag}(K1)$ and b is the bandwidth. K is updated after every iteration. The embeddings are normalized to unit norm, i.e., $||z_i||_2 = 1$, after each iteration. We perform fixed number of iterations during training. After these updates, a non-max suppression step yields M cluster centers $c_m, m = \{1, \ldots, M\}$ while making sure number of clusters are bounded. Having updated

the embeddings with the mean-shift iterations, we can now define a *soft membership* W for each point x_i, represented by the embedding vector y_i[1], to the cluster center c_m:

$$w_{i,m} = \frac{\exp(c_m^\top y_i)}{\sum_m \exp(c_m^\top y_i)} \tag{6.5}$$

where $w_{i,m} = 1$ represents the full membership of the i^{th} point to the m^{th} cluster. The supplementary material contains more details about non-max suppression and bandwidth computation.

Ellipsoid fitting. Given the clustering, we then fit an ellipsoid to each of the clusters. Traditionally, fitting an ellipsoid to a point cloud is formulated as a *minimum volume enclosing ellipsoid* [56] and solved using the Khachiyan algorithm. However, this is harder to incorporate in an end-to-end training pipeline and is also susceptible to outliers (see supplementary materials for details). We instead use a simpler procedure based on singular value decomposition (SVD). Given the membership of the point to m^{th} cluster we first center the points and compute the SVD as:

$$\mu = W_m X \mathcal{Z}^{-1} \tag{6.6}$$

$$\overline{X} = X - \mu \tag{6.7}$$

$$U, S, V = \texttt{SVD}(\overline{X}^T W_m \overline{X} \mathcal{Z}^{-1}) \tag{6.8}$$

where W_m is the diagonal matrix with $w_{i,m}$ as its diagonal entries ($W_m[i,i] = w_{i,m}$ for m^{th} cluster) and $\mathcal{Z} = \texttt{trace}(W_m)$. The orientation of an ellipsoid is given by V. The length of the principal axes can be computed from the singular values as $a_i = \kappa\sqrt{S_i}$. We select κ by cross validation. The matrix W_m selects the points with membership to the m^{th} cluster in a

[1]Superscript t is dropped.

'soft' or weighted fashion, and the SVD in Eq. 6.8 gives us the parameters of the ellipsoid that fits these weighted points.

Discussion — alternate choices for primitives. Our approach can be used to fit cuboids instead of ellipsoids by considering the bounding box of the fitted ellipsoids instead. This may induce different partitions over the point clouds, and we empirically compare its performance in the Sec. 7.4. Another choice is to represent the surface using an Atlas – a collection of parameterized patches. We use the technique proposed in AtlasNet [78] where neural networks f_θ parameterize the coordinate charts $f_\theta : [0, 1]^2 \rightarrow (x, y, z)$ conditioned on a latent code computed from the point cloud. The decoders are trained along with the encoder using gradient descent to minimize Chamfer distance between input points and output points across all decoders. An encoder trained in this fashion learns to decompose input points into complex primitives, *i.e.* via arbitrary deformations of the 2D plane. Point representations learnt in this fashion by the encoder can be used for downstream few-shot semantic segmentation task as shown in Sec. 7.4 and Tab. 6.2. However, this approach requires adding multiple decoder neural networks. Our *ellipsoid fitting* approach does not require significant architecture changes and avoids the extra parameters required by AtlasNet.

6.3.3 Loss functions

Reconstruction loss. This is computed as the Chamfer distance of input point clouds from predicted primitives. For the distance of a point on the input surface to the predicted surface we use

$$\mathcal{L}_1 = \sum_{i=1}^{N} \min_{m}^{M} D_m^2(x_i), \tag{6.9}$$

where M is the number of clusters and $D_m(x_i)$ is the distance of input point x_i from the m^{th} primitive. This is one side of Chamfer distance, ensuring that the predicted primitives cover the input surface. D_m is approximated using analytic distance function [2] which performs better than a sampling based approach (see the supplementary material for more details). To

ensure that the input surface covers the predicted primitives we minimize the following loss:

$$\mathcal{L}_2 = \sum_{m=1}^{M} \sum_{p \sim E_m} \min_{i=1}^{N} ||x_i - p||_2^2 \tag{6.10}$$

where E_m is the m^{th} fitted primitive. We sample $10k$ points over all ellipsoids, weighted by the surface area of each primitive. We uniformly sample each primitive surface. Please refer to Supplementary material for more information on uniformly sampling an ellipsoid surface. We use the two-sided loss to minimize reconstruction error:

$$\ell_{recon} = \mathcal{L}_1 + \mathcal{L}_2 \tag{6.11}$$

The hypothesis is that for a small number of primitives the above losses encourage the predicted primitives to fit the input surface. Since the fitting is done using a union of convex primitives, each diagonal entry of the matrix W_m in Eq. 6.8 should have higher weights to sets of points that belong to convex regions, thereby resulting in a convex (or approximately convex) segmentation of a point cloud. The point representations learnt in this manner are helpful for point cloud segmentation as shown in Table 6.2.

Intersection loss. To encourage spatially compact clusters we introduce a loss function that penalizes overlap between ellipsoids. Note that the clustering objective does not guarantee this as they operate on an abstract embedding space. Specifically, for each point p sampled inside the surface of predicted shape should be contained inside a single primitive. Alternatively the corresponding primitive should have negative signed distance $S_m(p)$ at that point p, whereas the the signed distance (SD) should be positive for the remaining primitives. Let $\mathcal{V}_m$ be the set of points sampled inside the primitive m. Then intersection loss is defined as

$$\ell_{inter} = \sum_{m} \sum_{p \sim \mathcal{V}_m} \sum_{j \neq m} \lfloor S_j(p) \rfloor_-^2, \tag{6.12}$$

where $\lfloor S_m(p) \rfloor_- = \min(S_m(p), 0)$ includes only the points with negative SD, as points with positive SD are outside the primitive and do not contribute in intersection. We use a differentiable approximation of the SD function of an ellipsoid proposed in [2].

Similarity loss. We found that all the per-point embedding are similar at initialization resulting in a mode collapse of the clusters. This local minima can be avoided by spreading the point embeddings across the space. We add a small penalty only at early stage of training that minimizes the similarity of output point embeddings Y from mean-shift iterations (Eq. 6.4) as follows:

$$\ell_{sym} = \sum_{i \neq j} (1 + y_i y_j^\top)^2 \tag{6.13}$$

6.3.4 Training details

We train our network jointly using both a self-supervised loss and a supervised loss. We alternate between our self-supervised training while sampling point clouds from the entire unlabeled $\mathcal{X}_u$ dataset and supervised training while taking limited samples from the labeled $\mathcal{X}_l$ set. Hence our joint supervision and self-supervision approach follows semi-supervised learning paradigm.

$$L = \underbrace{\sum_{X \sim \mathcal{X}_u} \ell_{recon} + \lambda_1 \ell_{inter} + \lambda_2 \ell_{sym}}_{\ell_{\text{ssl}}(\text{self-supervision})} + \underbrace{\sum_{X \sim \mathcal{X}_l} \ell_{ce}}_{\ell_{\text{sl}}(\text{supervision})} \tag{6.14}$$

where ℓ_{ce} is a cross entropy loss, λ_1 and λ_2 are constants.

Back-propagation and numerical stability

- To back-propagate the gradients through SVD computation we use analytic gradients derived by Ionescu *et al.* [149]. When back-propagating gradients through SVD, gradients can go to infinity when singular values are indistinct. This happens when membership weights in a cluster are concentrated on a line, point or sphere. We

Samples/cls.	k=1	k=3	k=5	k=10	k=20	k=50	k=100	k=max
Baseline	53.15 ± 2.4	59.54 ± 1.4	68.14 ± 0.9	71.32 ± 0.5	75.22 ± 0.8	78.79 ± 0.4	79.67 ± 0.3	$\mathbf{81.40 \pm 0.4}$
SURFIT	$\mathbf{63.14 \pm 3.4}$	$\mathbf{71.24 \pm 1.3}$	$\mathbf{73.75 \pm 0.7}$	$\mathbf{75.03 \pm 0.9}$	$\mathbf{76.73 \pm 0.5}$	$\mathbf{79.28 \pm 0.2}$	$\mathbf{80.16 \pm 0.2}$	80.40 ± 0.1

Table 6.1: **Few-shot segmentation on the ShapeNet dataset** (*class avg. IoU* over 5 rounds). The number of shots or samples per class is denoted by k for each of the 16 ShapeNet shape categories used for supervised training. Our proposed method SURFIT consistently outperforms the baselines.

implemented a custom Pytorch layer following [138], that changes the term $K = \frac{1}{\sigma_i - \sigma_j}$ from Eq: 13 in [108] to $K = \frac{1}{\text{sign}(\sigma_i - \sigma_j)(\max(|\sigma_i - \sigma_j|, \epsilon)}$ with $\epsilon = 1e-6$. SVD computation can still be unstable when the condition number of the input matrix is large. In this case, we remove that cluster from the backward pass when condition number is greater than $1e5$.

- *Differentiability of mean shift clustering procedure:* The membership matrix $W \in \mathbb{R}^{N \times M}$ is constructed by doing non-max suppression (NMS) over output Y of mean shift iteration. The derivative of NMS w.r.t embeddings Y is either zero or undefined, thereby making it non-differentiable. Thus we remove NMS from the computation graph and back-propagate through the rest of the graph, which is differentiable through Eq. 6.5. This can be seen as a straight-through estimator [196], which has been used in previous shape parsing works [138, 204].

We will release code of our implementation upon acceptance of the manuscript.

6.4 Experiments

6.4.1 Datasets

As a source of unlabeled data for the task of self supervision, we use the Shapenet Core dataset [28], which consists of 55 categories with 55,447 meshes in total. We sample these meshes uniformly to get 2048 points per shape. For the task of few-shot semantic segmentation, we use the Shapenet Semantic Segmentation dataset, which consists of 16,881 labeled point clouds across 16 shape categories with total 50 part categories.

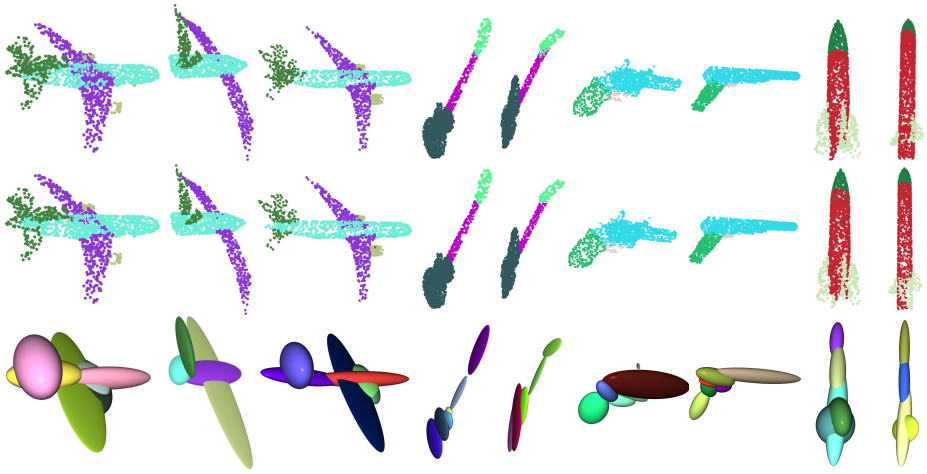


Figure 6.3: **Visualization of predicted semantic labels and ellipsoids on the Shapenet dataset. Top:** Ground truth point clouds, **middle:** predicted labels using our fitting approach, trained using $k = 10$ labeled examples per category, **bottom:** predicted ellipsoids. SURFIT predicts variable number of ellipsoids to approximate the input point cloud while maintaining correspondence with semantic parts.

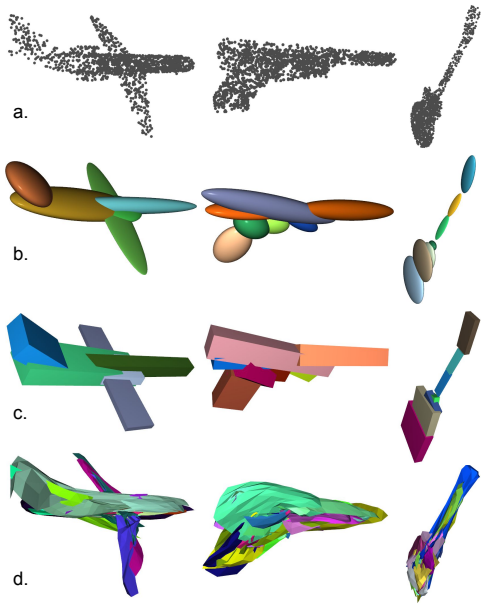


Figure 6.4: **Visualization of various primitive fitting approaches. a)** input point cloud. **b)** Ellipsoid fitting using our approach. **c)** cuboid fitting using our approach. **d)** different primitives from AtlasNet. Different colors are used to depict different primitives. For AtlasNet we visualize each chart with a unique color. Notice that geometric primitives are better localized and approximate the shape in fewer primitives in comparison to AtlasNet.

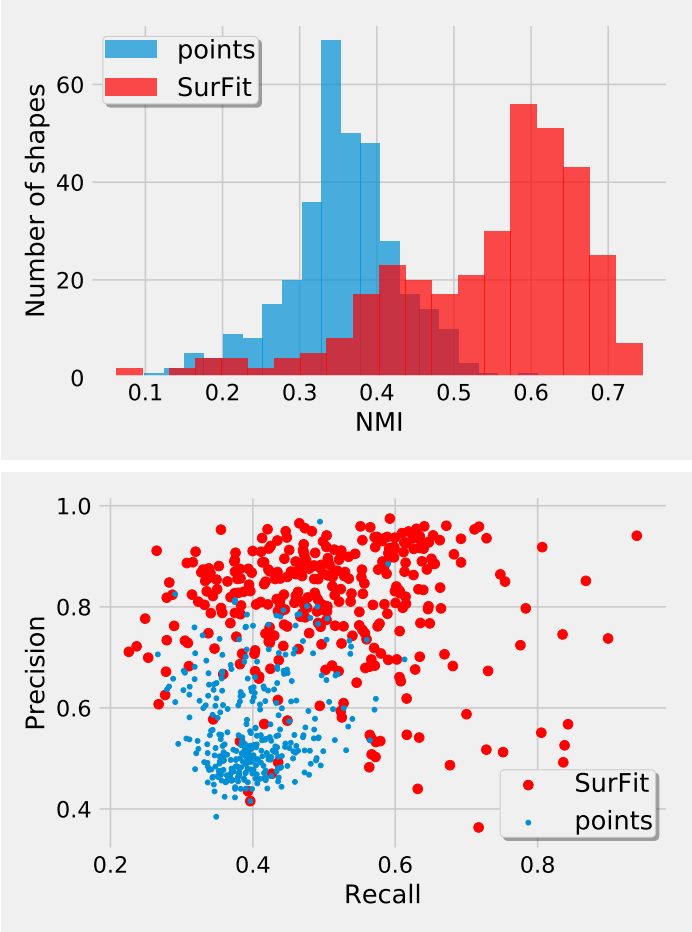


Figure 6.5: **Analysis of clustering.** We analyze two clustering approaches, 1) SURFIT and 2) directly clustering *points* using K-Means. *Top:* normalized mutual information (NMI) and *bottom:* precision vs recall between predicted clusters and semantic part labels. SURFIT gives higher average NMI (54.3 vs 35.4) and higher precision than clustering with only points as features.

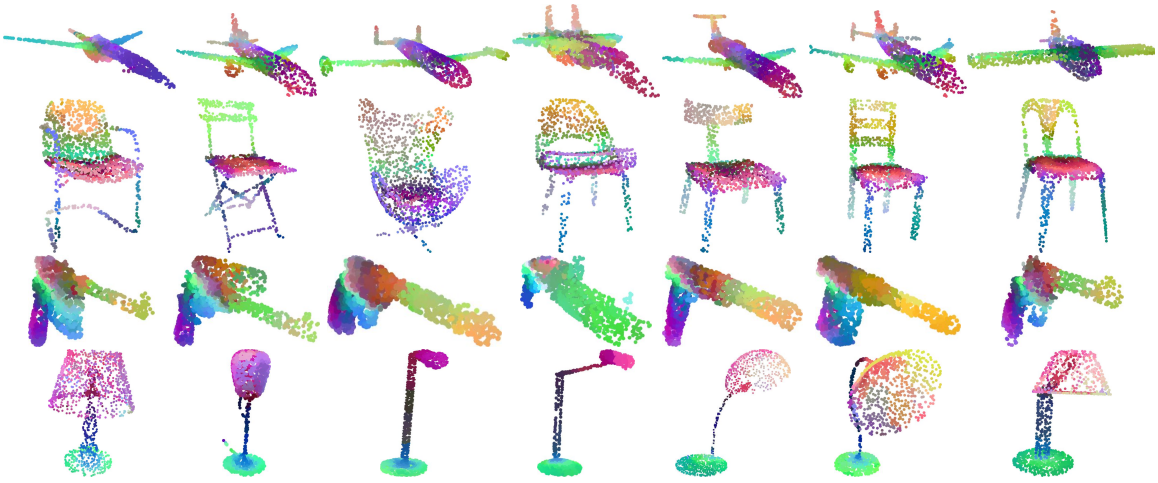


Figure 6.6: **TSNE visualization of learned embeddings.** For each shape category, we take a fixed number of shapes and extract point embeddings from SURFIT. We run TSNE on each category separately to project the 128-D embeddings to 3D color space. Notice that points belonging to same semantic parts are colored similarly, which indicates the consistency of learned embeddings.

We also evaluate our method on the PartNet dataset [160]. This dataset provides *fine-grained* semantic segmentation annotation for various 3D shape categories, unlike the more coarse-level shape parts in the ShapeNet dataset. We use 12 categories from "level-3", which denotes the finest level of segmentation. For training different approaches in few-shot framework, we remove test shapes of labeled dataset $\mathcal{X}_l$ from our self-supervision dataset $\mathcal{X}_u$. This avoids train-test set overlap.

6.4.2 Few-shot part segmentation on Shapenet

For each category from the ShapeNet part segmentation dataset, we randomly sample k labeled shapes ($16 \times k$ in total) and use these for training the semantic segmentation component of our model. We use the entire training set from the ShapeNet core dataset for self supervision task. We train a *single* model on all 16 shape categories of ShapeNet.

Baselines. Our first baseline takes the PointNet++ as the base architecture and trains it from scratch on only labeled training examples, using k labeled shapes per category in a few-shot setup. Second, we create a reconstruction baseline – we use a PointNet++ as a

Method	1% ins IoU	5% ins IoU	1% cls IoU	5% cls IoU
SO-Net [136]	64.0	69.0	-	-
PointCapsNet [265]	67.0	70.0	-	-
MortonNet [222]	-	77.1	-	-
Deformation [232]	68.9	-	66.2	-
JointSSL [11]	71.9	77.4	-	-
Multi-task [87]	68.2	77.7	-	72.1
ACD [64] †	75.1	78.6	74.6	77.5
SURFIT w/ Atlas	73.8	78.6	74.5	78.9
SURFIT w/ Cuboid	75.2	78.6	74.6	78.6
SURFIT w/ Ellipsoid	**75.4**	**78.7**	**75.3**	**79.0**

Table 6.2: **Comparison with state-of-the-art few-shot part segmentation methods on ShapeNet.** Performance is evaluated using *instance-averaged* and *class-averages* IoU. † - We re-ran the publicly-released code from ACD [64] on our data splits, ensuring fair comparison.

shared feature extractor, which extracts a global shape encoding that is input to an AtlasNet decoder [78] with 25 charts. A separate decoder is used to predict per-point semantic labels. This network is trained using a Chamfer distance-based reconstruction loss using the entire unlabeled training set and using k labeled training examples. We use 5 different rounds with sampled labeled sets at various values of k and report their average performance. We train a single model for all categories.

Discussion of results. Table 6.1 shows results on few-shot segmentation at different number of labeled examples. Our method SURFIT performs better than the supervised baseline showing the effectiveness of our method as semi-supervised approach.

In Table 6.2 we compare our approach with previous methods using instance IOU and class IOU [179], using 1% and 5% of the labeled training set to train different methods. Note that instance IOU is highly influenced by the shape categories with large number of testing shapes *e.g.* Chair, Table. Class IOU, on the other hand gives equal importance to all categories, hence it is a more robust evaluation metric. SURFIT performs better than previous self-supervised [64, 87, 222] and semi-supervised [11, 232] approaches. Notable

our approach significantly outperforms learned deformation based approach proposed by Wang *et al.* [232] [2]. SURFIT with ellipsoid as primitive outperforms previous methods including SURFIT baseline with AtlasNet. Interestingly, SURFIT with AtlasNet outperforms all previous baselines except ACD[3]. We speculate that since primitives predicted by AtlasNet are highly overlapped and less localized in comparison to our ellipsoid and cuboid fitting approaches as shown in Figure 6.4, this results in worse performance of AtlasNet.

SURFIT with ellipsoid primitives gives the best results, outperforming ACD without having to rely on an external black-box method for the self-supervisory training signal. This shows that our approach of primitive fitting in an end-to-end trainable manner is better than training a network using contrastive learning guided by approximate convex decomposition of water-tight meshes as proposed by ACD.

In Figure 6.3 we show predicted semantic labels using SURFIT along with fitted ellipsoids. In Figure 6.4 we show outputs of various self supervision techniques using primitive fitting. We experimented with both cuboid and ellipsoid fitting as a self supervision task. We observed that both performs similar qualitatively and quantitatively. The fitted primitives using ellipsoid/cuboid fitting approaches are more aligned with different parts of the shape in comparison to the outputs of AtlasNet.

Effect of the size of unlabeled dataset. In the Table 6.3 we show the effect of size of unlabeled dataset used for self-supervision. We observe improvement in performance of 5-shot semantic segmentation task with increase in unlabeled dataset.

Effect of similarity loss. Similarity loss is only used in the initial stage of training as it prevents the network converging to a local minima at this early phase. Without this procedure, our performance is similar to Baseline (training from scratch). However, using

[2]We run their code on all 16 categories on 5 different random splits

[3]We run their code on our random split for fair comparison.

size	0	2.5k	25k	52k
class avg. IOU	68.1	68.9	72.6	**73.7**

Table 6.3: Effect of the size of unlabeled dataset used for self-supervision on 5-shot semantic segmentation on ShapeNet.

only similarity loss (without reconstruction loss) leads to worse results (44.2 mIOU) than Baseline (68.1 mIOU) on few-shot k=5 setting.

Analysis of learned point embeddings In Figure 6.5 we quantitatively analyze the performance of clustering induced by SURFIT and compare it with clustering obtained by running the K-Means algorithm directly on point clouds. We take 340 shapes from the Airplane category of ShapeNet for this experiment and show the histogram of Normalized Mutual Information (NMI) [211] and the precision-recall curve [62] between predicted clusters and ground truth part labels. Our approach produces clusters with higher NMI (54.3 vs 35.4), which shows better alignment of our predicted point clusters with the ground truth part labels. Our approach also produces higher precision clusters in comparison to K-Means at equivalent recall, which shows the tendency of our algorithm to over-segment a shape. To further analyze the consistency of learned point embedding across shapes, we use TSNE [227] to visualize point embedding by projecting them to 3D color space. We use a fixed number of shapes for each category and run TSNE on each category separately. Figure 6.6 shows points belonging to same semantic parts are consistently projected to similar colors, further confirming the consistency of learned embeddings.

6.4.3 Few-shot segmantic segmentation on PartNet

Here we experiment on the PartNet dataset for the task of few-shot semantic segmentation. For each category from this dataset, we randomly sample k labeled shapes and use these for training semantic segmentation part of the architecture. Similar to our previous experiment, we use the complete training shapes from the ShapeNet Core dataset for the self-supervision task. We choose DGCNN as a backbone architecture for this experiment.

Samples/cls.	**k=10**	**k=20**	**k=40**
Baseline	27.2±0.7	31.6±0.6	36.7±0.9
SURFIT w/ Atlas	28.5±0.7	31.7±0.7	36.5±0.7
SURFIT w/ Cuboid	29.3±0.4	32.4±0.7	37.5±0.5
SURFIT w/ Ellipsoid	**29.4±0.7**	**32.6± 0.6**	**37.6±0.4**

Table 6.4: **Few-shot segmentation on the PartNet dataset** (*part avg. IoU* over 5 rounds). The number of shots or samples per class is denoted by k for each of the 12 PartNet categories used for supervised training. Our proposed method SURFIT consistently out performs the baseline.

Unlike our Shapenet experiments, in the PartNet experiment we train a separate model for each category, as done in the original paper [160].

Similar to our previous experiment, we create two baselines – 1) we train a network from scratch providing only k labeled examples, and 2) we train the AtlasNet on the entire unlabeled training set using the self-supervised reconstruction loss and only k labeled examples as supervision.

Table 6.4 shows part avg. IOU for the different methods. The AtlasNet method shows improvement over the purely-supervised baseline. SURFIT shows improvement over both AtlasNet and Baseline, indicating the effectiveness of our approach in the *fine-grained* semantic segmentation setting as well. We further experiment with cuboids as the primitive, which achieves similar performance as using ellipsoid primitives, consistent with our previous ShapeNet results.

Effect of intersection loss. We also experiment with adding intersection loss while training SURFIT with ellipsoid primitive in Table 6.5. Intersection loss gives improvement only in PartNet dataset. We speculate that since PartNet contains fine-grained segmentation of shapes, minimizing overlap between primitives here is more helpful than in Shapenet dataset which contains only coarse level of segmentation. Table 6.2 and 6.4 shows best performing approach.

Method	Shapenet cls. IoU		PartNet part avg. IoU	
	k=1%	k=5%	k=10	k=20
SURFIT w/ Ellipsoids	**75.3**	**79.0**	28.9	32.1
SURFIT w/ Ellipsoids+inter	75.0	**79.0**	**29.4**	**32.6**

Table 6.5: **Effect of intersection loss.** Average performance over all categories for SURFIT trained with intersection loss (+*inter*) and without intersection loss on ShapeNet and PartNet dataset. SURFIT trained with intersection loss (+*inter*) gives improvement on PartNet dataset.

6.5 Conclusion

We proposed a simple semi-supervised learning approach for learning point embeddings for few shot semantic segmentation. Our approach learns to decompose an unlabeled point cloud using a set of geometric primitives, such as ellipsoids and cuboids, or alternatively deformed planes as in AtlasNet. We provide an end-to-end trainable framework for incorporating this task into standard network architectures for point cloud segmentation. Our method can be readily applied to existing architectures for semantic segmentation and shows improvements over fully-supervised baselines and other approaches on the ShapeNet and PartNet datasets. This indicates that learning to reconstruct a shape using primitives can induce representations useful for discriminative downstream tasks. Our method also has limitations. We primarily rely on basic primitives of a single type (*i.e.* ellipsoid, cuboids, or deformed planes). Predicting combinations of primitives or other types of primitives for each shape could be useful to capture more part variability. In addition, our primitives capture the shape rather coarsely, making our representations less fit for fine-grained tasks. Our current implementation requires an input unlabeled set of shapes together with a small labeled one. It would be interesting to also explore unsupervised approaches based on primitive and other forms of surface fitting.

CHAPTER 7

MVDECOR: MULTI-VIEW DENSE CORRESPONDENCE LEARNING FOR FINE- GRAINED 3D SEGMENTATION

7.1 Introduction

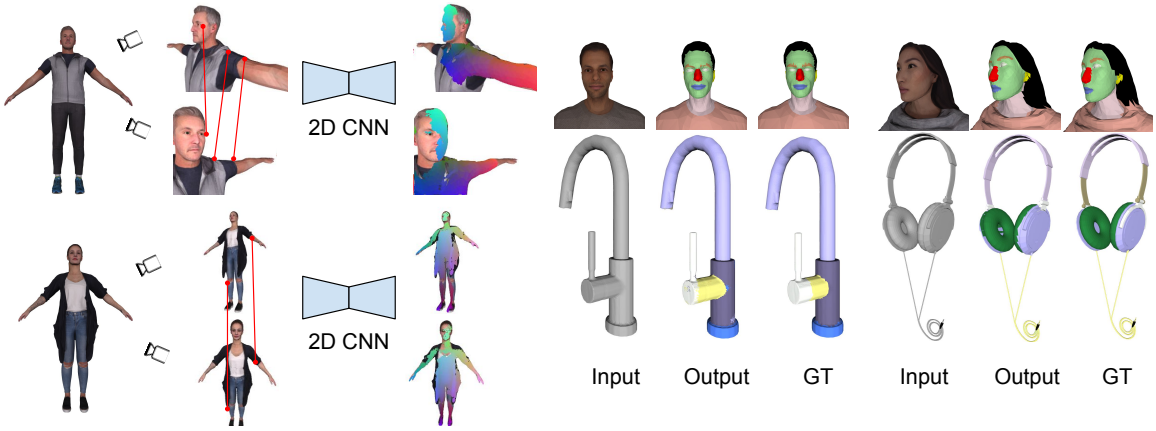


Figure 7.1: **The pipeline for MVDECOR**. *Left:* Dense 2D representations are learned using a pixel-level correspondence learning framework guided by the 3D shape. *Right:* The 2D representations can be fine-tuned on a few labeled examples for 3D shape segmentation tasks in a multi-view setting.

Part-level interpretation of 3D shapes is critical for many applications in graphics and vision. Specifically, our goal in this work is to perform fine-grained shape segmentation from limited available data. This poses two main challenges. First, training deep networks relies on large-scale labeled datasets that require tremendous annotation effort. For this reason, previous methods have proposed self-supervised feature extraction, however these mostly rely on point cloud or voxel-based networks. This brings us to the second challenge – these 3D networks have a limited ability to capture fine-grained surface details since their input points and voxels often fail to capture such due to discretization artifacts.

We present MVDECOR, a self-supervised technique for learning dense 3D shape representations based on the task of learning correspondences across views of a 3D shape (Fig. 7.1). At training time we render 3D shapes from multiple views with known correspondences and setup a contrastive learning task to train 2D CNNs. In doing so, we take advantage of the excellent abilities of 2D networks to capture fine details. The learned 2D representations can be directly used for part segmentation on images, or projected onto the shape surface to produce a 3D representation for 3D tasks (Fig. 7.1 & 7.2). The approach works well in standard few-shot fine-grained 3D part segmentation benchmarks, outperforming prior work based on 2D and 3D self-supervised learning (Section 7.3, Tab. 7.1 & 7.2).

Many previous representation learning methods for 3D shapes are based on self reconstruction loss [7, 36, 65, 78, 251] or contrastive learning [98, 189, 261] where point clouds and voxels are the main choices of 3D data formats. In contrast, our work is motivated from the observation that view-based surface representations are more effective at modeling high-resolution surface details and texture than their 3D counterparts based on point clouds or voxel occupancy. We also benefit from recent advances in network architectures and self-supervised learning for 2D CNNs. In addition, our approach allows training the network using 2D labeled views rather than fully labeled 3D shapes. This is particularly beneficial because annotating 3D shapes for fine-grained semantic segmentation is often done using 2D projections of the shape to avoid laborious 3D manipulation operations [254].

Compared to techniques based on 3D self-supervision, MVDECOR demonstrates significant advantages. On the PartNet dataset [159] with fine-grained (Level-3) parts, our method achieves 32.6% mIOU compared to a PointContrast [189], a self-supervised learning technique that achieves 31.0% mIOU (Tab. 7.1). While some of the benefit comes from the advantages of view-based representations, e.g., off-the-self 2D CNNs trained from scratch outperform their 3D counterparts, this alone does not explain the performance gains. MVDECOR outperforms both ImageNet pretrained models (29.3% mIOU) and dense contrastive learning [238] (30.8% mIOU), suggesting that our self-supervision is benefi-

cial. These improvements over baselines are even larger when sparse view supervision is provided — MVDECOR generalizes to novel views as it has learned a view invariant local representations of shapes (Tab. 7.2).

We also present experiments on the RenderPeople [5] dataset consisting of textured 3D shapes of humans, which we label with 13 parts (Section 7.4). We observe that 2D self-supervised techniques performs better than their 3D counterparts, while MVDECOR offers larger gains over both 2D and 3D baselines (Tab. 7.4). Surprisingly on this dataset we find that when texture is available, the view-based representations do not require the use of depth and normal information, and in fact the models generalize better without those, as explained in Section 7.4.6. MVDECOR gives 17.3% mIOU improvement over training a network from scratch when only a few labeled examples are provided for supervision.

To summarize, we show that multi-view dense correspondence learning induces view-invariant local representations that generalize well on few-shot 3D part segmentation tasks. Our approach MVDECOR outperforms state-of-the art 2D contrastive learning methods, as well as 3D contrastive learning methods that operate on point cloud representations. After a discussion of prior work on 2D and 3D self-supervised learning in Section 7.2, we describe our method and experiments in Section 7.3 and Section 7.4 respectively.

7.2 Related Works

Our work lies at the intersection of 3D self-supervision, 2D self-supervision, and multi-view representations.

7.2.0.0.1 ***3D self-supervision.*** Many self-supervised approaches in 3D shape are based on training an autoencoder with a reconstruction loss. For example, Achlioptas *et al.* [7] train a PointNet [178] with a Chamfer or EMD loss. FoldingNet [251] deforms a 2D grid using a deep network conditioned on the shape encoding to match the output shape. AtlasNet [78] uses multiple decoders to reconstruct the surface as a collection of patches. BAE-NET [36] splits reconstruction across decoding branches, but adopted an implicit field

shape representation instead of point clouds. Once trained the representations from the encoder can be used for downstream tasks. Alternatives to reconstruction include prediction based on k-means [11], convex decomposition [141] and 3D jigsaw puzzles [191]. Recently Wang *et al.* [231] proposed an occlusion completion as a pre-training task for 3D shapes, and show improvement in object classification and segmentation tasks. Unsupervised learning for recovering dense correspondences between non-rigid shapes has been studied in [73, 80], however it relies on a near-isometry assumption that does not fit clothed people and furniture parts, used in our work. We instead use partial correspondences from the 3D models to supervise 2D networks. Wang *et al.* [232] proposed a deep deformation approach that aligns a labeled template shape to unlabeled target shapes to transfer labels. However this method is not effective for fine-grained segmentation of shapes as shown in Section 7.4 since deformation often distorts surface details. A few recent works [189, 235, 262] have learned per-point representations for point clouds under a contrastive learning framework. The networks pre-trained in this fashion are further fine-tuned for 3D downstream tasks. However, point cloud based shape representations limit the ability to capture fine-grained surface details and texture.

7.2.0.0.2 ***2D self-supervision.*** While early work focused on training networks based on proxy tasks such as image colorization, rotation prediction, and jigsaw puzzles, contrastive learning [32, 76, 89, 248] has emerged as a popular technique. Most of these representations are based on variants of InfoNCE loss [168], where the mutual information between two views of an image obtained by applying synthetic transformations is maximized. DenseCL [238] modifies the contrastive approach to include information across locations within an image to learn dense representations. We use this method as the representative 2D self-supervised baseline. However, the above methods work on the 2D domain and lack any 3D priors incorporated either in the network or in the training losses. Correspondence learning has been used as self-supervision task to learn local descriptors for geometric matching in structure from motion applications [99, 146, 190, 238]. However, much of this

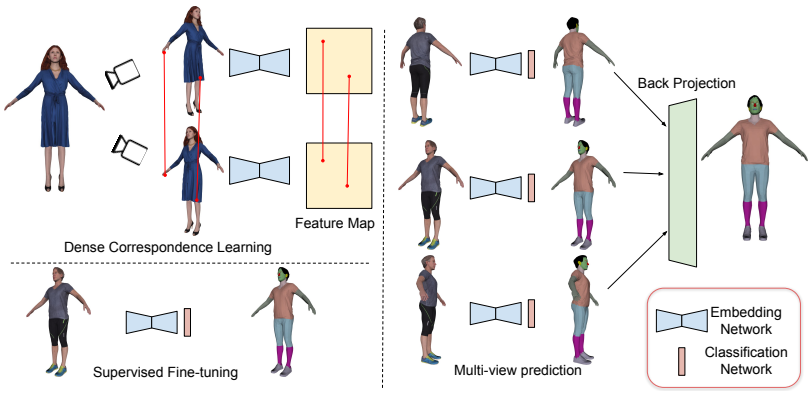


Figure 7.2: **Overview of MVDECOR.** *Top left:* Our self-supervision approach takes two overlapping views (RGB image, with optional normal and depth maps) of a 3D shape and passes it through a network that produces per-pixel embeddings. We define a dense contrastive loss promoting similarity between matched pixels and minimizing similarity between un-matched pixels. *Bottom left:* Once the network is trained we add a segmentation head and fine-tune the entire architecture on a few labeled examples to predict per-pixel semantic labels. *Right:* Labels predicted by the 2D network for each view are back-projected to the 3D surface and aggregated using a voting scheme.

work has focused on instance matching, while our goal is to generalize across part instances within a category. The most related work to ours is Pri3D [99] who also propose to learns geometry-aware embedding with a contrastive loss based on pixel correspondences across views. Their work focuses on improving 2D representations using 3D supervision for 2D tasks such as scene segmentation and object detection, while we deal with fine-grained 3D segmentation.

7.2.0.0.3 ***Multi-view representation.*** Our method is motivated by earlier multi-view approaches for 3D shape recognition and segmentation [39, 55, 101, 117, 212]. In these approaches, multiple views of the shapes are processed by a 2D network to obtain pixel-level representations. These are back-projected to the 3D shape to produce 3D surface representations. More recently, Kundu *et al.* [55] applies multi-view fusion for 3D semantic segmentation of scenes. The above approaches benefit from large-scale ImageNet pretraining, and the ability of 2D CNNs to handle higher image resolutions compared to voxel grids and point clouds. They continue to outperform 3D deep networks on many 3D tasks

(e.g., [75, 214]). Still, all the above view-based methods are still trained in a fully supervised manner, while ours is based on self-supervision.

7.3 Method

Our goal is to learn a multi-view representation for 3D shapes for the task of fine-grained 3D shape segmentation. For this task, we assume a large dataset of unlabeled 3D shapes and a small number of labeled examples. For the latter, we consider two settings (1) when labels are provided on the surface of the 3D shape, and (2) labels are provided on the images (projections) of the 3D shape. To that end, we use multi-view dense correspondence learning for pre-training, followed by a feature fine-tuning stage on the downstream segmentation task. In the pre-training stage described in Section 7.3.1, we have a set of unlabeled 3D shapes (either with or w/o textures) from which we render 2D views, and build ground-truth dense correspondences between them. After pre-training on the dense correspondence learning task, the network learns pixel-level features that are robust to view changes and is aware of fine-grained details.

In the fine-tuning stage (Section 7.3.2), we train a simple convolutional head on top of the pixel-level embeddings, supervised by a small number of annotated examples, to segment the multi-view renderings of the 3D shapes. The network pre-trained in this fashion produces better segmentation results under the few-shot semantic segmentation regime in comparison to baselines. We aggregate multi-view segmentation maps onto 3D surface via an entropy-based voting scheme (Section 7.3.3). Figure 7.2 shows the overview of our approach.

7.3.1 Multi-view dense correspondence learning

Let us denote the set of *unlabeled* shapes as $\mathcal{X}_u$. Each shape instance $X \in \mathcal{X}_u$ can be rendered from a viewpoint i into color, normal and depth images denoted as V^i. We use a 2D CNN backbone $\mathbf{\Phi}$ which maps each view into pixel-wise embeddings $\{\mathbf{\Phi}(V^i)_p\} \in \mathbb{R}^D$,

where p is an index of a pixel and D is the dimensionality of the embedding space. We pre-train the network $\mathbf{\Phi}$ using the following self-supervised loss:

$$\mathcal{L}_{\mathsf{ssl}} = \mathop{\mathbb{E}}_{\substack{V^i, V^j \sim \mathcal{R}(X) \\ X \sim \mathcal{X}_u}} \Big[\ell_{\mathsf{ssl}}\big(\mathbf{\Phi}(V^i), \mathbf{\Phi}(V^j)\big) \Big] \tag{7.1}$$

where $\mathcal{R}(X)$ is the set of 2D renderings of shape X, and V^i and V^j are sampled renderings in different views from $\mathcal{R}(X)$. The self-supervision loss ℓ_{ssl} is applied to the pair of sampled views V^i and V^j.

Since V^i and V^j originate from the 3D mesh, each foreground pixel in the rendered images corresponds to a 3D point on the surface of a 3D object. We find matching pixels from V^i and V^j when their corresponding points in 3D lie within a small threshold radius. We use the obtained dense correspondences in the self-supervision task. Specifically, we train the network to minimize the distance between pixel embeddings that correspond to the same points in 3D space and maximize the distance between unmatched pixel embeddings. This encourages the network to learn pixel embeddings to be invariant to views, which is a non-trivial task, as two rendered views of the same shape may look quite different, consisting of different contexts and scales. We use InfoNCE [168] as the self-supervision loss. Given two rendered images (V^i and V^j) from the same shape X and pairs of matching pixels p and q, the InfoNCE loss is defined as:

$$\begin{aligned} &\ell_{\mathsf{ssl}}(\mathbf{\Phi}(V^i), \mathbf{\Phi}(V^j)) = \\ &- \sum_{(p,q)\in M} \log \frac{\exp(\mathbf{\Phi}(V^i)_p \cdot \mathbf{\Phi}(V^j)_q / \tau)}{\sum_{(.,k)\in M} \exp(\mathbf{\Phi}(V^i)_p \cdot \mathbf{\Phi}(V^j)_k / \tau)} \end{aligned} \tag{7.2}$$

where $\mathbf{\Phi}(V^i)_p$ is the embedding of pixel p in view i, M is the set of paired pixels between two views that correspond to the same points in 3D space, and the temperature $\tau = 0.07$ in our experiments. We use two views that have at least 15% overlap. The output $\mathbf{\Phi}(V^i)_p$ and $\mathbf{\Phi}(V^j)_q$ of the embedding module are normalized to a unit hyper-sphere. Pairs of matching pixels are treated as positive pairs. The above loss also requires sampling of negative pairs.

Given the matching pixels $(p, q) \in M$ from V^i and V^j respectively, for each pixel p from the first view, the rest of the pixels $k \neq q$ appearing in M and belonging to the second view, yield the negative pixel pairs (p, k).

7.3.2 Semantic segmentation of 3D shapes

In the fine-grained shape segmentation stage, the network learns to predict pixel level segmentation labels. Once the embedding module is pre-trained using the self-supervised approach, it is further fine-tuned in the segmentation stage, using a small labeled shape set $\mathcal{X}_l$ to compute a supervised loss, as follows:

$$\min_{\mathbf{\Phi},\mathbf{\Theta}} \lambda \mathcal{L}_{\mathsf{ssl}} + \mathcal{L}_{\mathsf{sl}}, \text{ where} \tag{7.3}$$

$$\mathcal{L}_{\mathsf{sl}} = \mathop{\mathbb{E}}_{(X,Y)\sim\mathcal{X}_l} \Big[\mathop{\mathbb{E}}_{(V^i,L^i)\in\mathcal{R}(X,Y)} \ell_{\mathsf{sl}}\big(L^i, \mathbf{\Theta} \circ \mathbf{\Phi}(V^i)\big) \Big] \tag{7.4}$$

where $\mathbf{\Theta}$ is the segmentation module, λ is a hyper-parameter set to 0.001, and ℓ_{sl} is the semantic segmentation loss implemented using cross-entropy loss applied to each view of the shape separately. $\mathcal{R}(X, Y)$ is the set of renderings for shape X and its 3D label map Y. L^i represents the projected labels from the 3D shape for view V^i. Since the labeled set is much smaller than the unlabeled set, the network could overfit to the small set. To avoid this over-fitting during the fine-tuning stage, we use the self-supervision loss $\mathcal{L}_{\mathsf{ssl}}$ as an auxiliary loss along with supervision loss $\mathcal{L}_{\mathsf{sl}}$ as is shown in Eq. 7.3. In Table 7.5 we show that incorporating this regularization improves the performance.

During inference, we render multiple overlapping views of the 3D shape, and segment each view. The per-pixel segmentation labels are then projected back onto the surface. We use ray-tracing to encode the triangle index of the mesh to which each pixel of the rendered view corresponds to. To aggregate the segmentation labels from different views for each triangle of the mesh, one option is to use majority-voting. An illustration of the process is shown in Fig. 7.2. However, not all views should be allowed to vote equally towards the

final segmentation label for each triangle, as some views are ambiguous and are not suitable to recognize a particular part of the shape. We instead define a weighted voting scheme based on the average entropy of the probability distribution predicted by the network for a view. More specifically, a weight $W^i = (1 - \sum_{p \in F} H^{i,p}/|F|)^\gamma$ is given to the view i, where F is the set of foreground pixels, H is the entropy of the probability distribution predicted by the network at pixel p, and γ is a hyperparameter set to 20 in our experiments. More weight is given to the view with less entropy. Consequently, for each triangle t on the mesh of the 3D shape, the label is predicted as follows:

$$l_t = \arg\max_{c \in C} \sum_{i \in I, p \in t} W^i P^{(i,p)} \tag{7.5}$$

where I is the set of views where the triangle t is visible, $P^{(i,p)}$ is the probability distribution of classes at a pixel $p \in t$ in view i, and C is the set of segmentation classes.

7.3.3 Implementation details

The embedding module $\mathbf{\Phi}$ is implemented as the DeepLabV3+ network [31] originally proposed for image segmentation with ResNet-50 backbone. We add extra channels in the first layer to incorporate depth and normal maps. Specifically, it takes a K-channel image (V^i) as input of size $H \times W \times K$ and outputs $\mathbf{\Phi}(V^i)$ per pixel features of size $H \times W \times 64$, where the size of pixel embedding is 64. In the second stage, we add a segmentation head (a 2D convolutional layer with a softmax) on top of the pixel embedding network to produce per-pixel semantic labels. Additional architecture details are provided in the Appendix.

To generate the dataset for the self-supervision stage, we start by placing a virtual camera at 2 unit radius around the origin-centered and unit normalized mesh. We then render a fixed number of images by placing the camera at uniform positions and adding random small perturbations in the viewing angle and scale. In practice, we use approximately 90 rendered images per shape to cover most of the surface area of the shapes. We also render depth and normal maps for each view. Normal maps are represented in a global coordinate system.

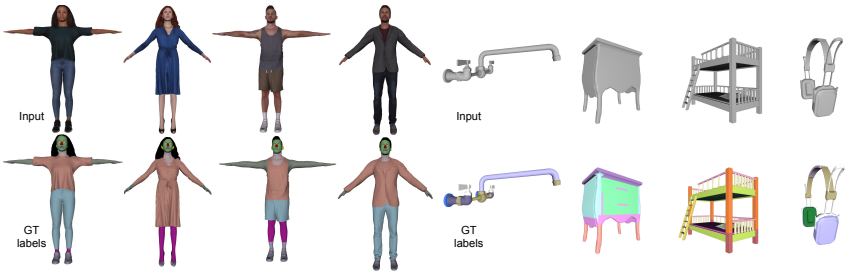


Figure 7.3: **Examples from datasets used in our experiments.** *Left*: RenderPeople [5] dataset. *Right*: PartNet [159] dataset

Depth maps are normalized within each view. We use ray tracing to record the triangle index to which each pixel corresponds to and also the point-of-hit for each pixel. This helps in identifying correspondences between two views of the same shape. More information is provided in the Appendix. We will release our code for full reproducibility.

7.4 Experiments and results

7.4.1 Dataset

We use the following datasets in our experiments, samples from which are visualized in Figure 7.3. The license information is provided in the Appendix.

7.4.1.0.1 ***PartNet*** *[159].* This dataset provides *fine-grained* semantic segmentation annotation for various 3D shape categories, unlike the more coarse-level shape parts in the ShapeNet-Part dataset. We use 17 categories from "level-3", which denotes the finest level of segmentation. On average the categories contain 16 parts, ranging from 4 for the Display category, to 51 for Table category. For training in the few-shot framework, we use the entire training and validation set as the self-supervision dataset $\mathcal{X}_u$, and select k shapes from the train set as labeled dataset $\mathcal{X}_l$ for the fine-tuning stage. Note that we provide experiments on three fine-grained categories (Chair, Table and Lamp) of PartNet in the Appendix. as described in Section 7.4.5.

7.4.1.0.2 ***RenderPeople*** *[5].* This dataset contains 1000 human textured models represented as triangle meshes in two poses. We use 936 shapes of them for self-supervision. We label the remaining 64 shapes with 13 different labels, while focusing more on facial semantic parts. The 64 labeled shapes consist of 32 different identities in 2 poses. We randomly split the 32 identities into 16 and 16, so that we get 32 shapes as the labeled training set, and 32 shapes as the test set for evaluation. More details of the labeling and individual semantic parts are provided in the Appendix.

7.4.1.0.3 ***ShapeNet-Part*** *[254].* We also show experiments on ShapeNet-Part dataset in the Appendix where we outperform previous works on class-average mIOU metric.

7.4.2 Experiment settings

7.4.2.0.1 ***Segmentation using limited labeled shapes.*** We pre-train our 2D embedding network and fine-tune it with a segmentation head using k labeled shapes. During fine-tuning, each shape is rendered from 86 different views. We render an extra 10 images for the RenderPeople dataset that focuses more on details of facial regions. Each view consists of a grayscale image for PartNet dataset and textured image for RenderPeople dataset, a normal and a depth map for both datasets. For PartNet dataset, pre-training is done using all shape categories and fine-tuning is done on individual category specific manner. All experiments in this few-shot setting are run 5 times on randomly selected k labeled training shapes and the average part mIOU is reported.

7.4.2.0.2 ***Segmentation using limited labeled views per shape.*** In this setting supervision is available in 2D domain in the form of sparse set of labeled views per shape. Specifically, a small number of k shapes are provided with a small number of v labeled views per shape. For training 3D baselines, labeled views are projected to 3D mesh and corresponding points are used for supervision. Similar to the first setting, all experiments are run 5 times on randomly selected k labeled training shapes and the average part mIOU is reported.

7.4.3 Baselines

We compare our method against the following baselines:

- **(2D) Scratch.** In this baseline, we train our 2D networks (Φ and segmentation head Θ) directly using the small set of labeled examples, without pre-training, and apply the same multi-view aggregation.

- **(2D) ImageNet.** We create a baseline in which the backbone ResNet in our embedding network Φ is initialized with ImageNet pre-trained weights and the entire network is fine-tuned using few-labeled examples. The first layer of ResNet trained on Imagenet is adapted to take the extra channels (normal and depth maps) following the method proposed in ShapePFCN [117].

- **(2D) DenseCL.** To compare with the 2D dense contrastive learning method we also create a baseline using DenseCL [238]. We pre-train this method on our unlabeled dataset $\mathcal{X}_u$ using their original codebase. Once this network is trained, we initialize the backbone network with the pre-trained weights and fine-tune the entire architecture using the available labeled set. Further details about training these baselines are in the Appendix.

- **(3D) Scratch.** In this baseline, we train a 3D ResNet based on sparse convolutions (Minkowski Engine [38]) that takes uniformly sampled points and their normals from the surface and predicts semantic labels for each point. We train this network directly using the small set of labeled examples.

- **(3D) PointContrast.** To compare with the 3D self-supervision methods, we pre-train the above 3D ResNet (Minkowski Engine) on $\mathcal{X}_u$ using the approach proposed in PointContrast [189]. The pre-trained network is later fine-tuned by adding a segmentation head (a 3D convolution layer and softmax) on top to predict per-point semantic labels. We use codebase provided by authors to train the network. Implementation details are provided in Appendix.

- **(3D) non-learning based nearest neighbor transfer.** In this baseline, for each test shape we find nearest training shape using Chamfer distance. Then for each point in the test shape, we find closest point on the retrieved shape and transfer labels to each point cloud. Finally, we evaluate the performance using these transferred labels.

- **(3D) Weak supervision via learned deformation.** In this baseline, we use an approach proposed by Wang *et al.* [232] which uses learning based deformation and transfer of labels from labeled set to unlabeled shape. We use our labeled and unlabeled set to train this method using the code provided by the authors.

7.4.4 Visualization of learned embeddings.

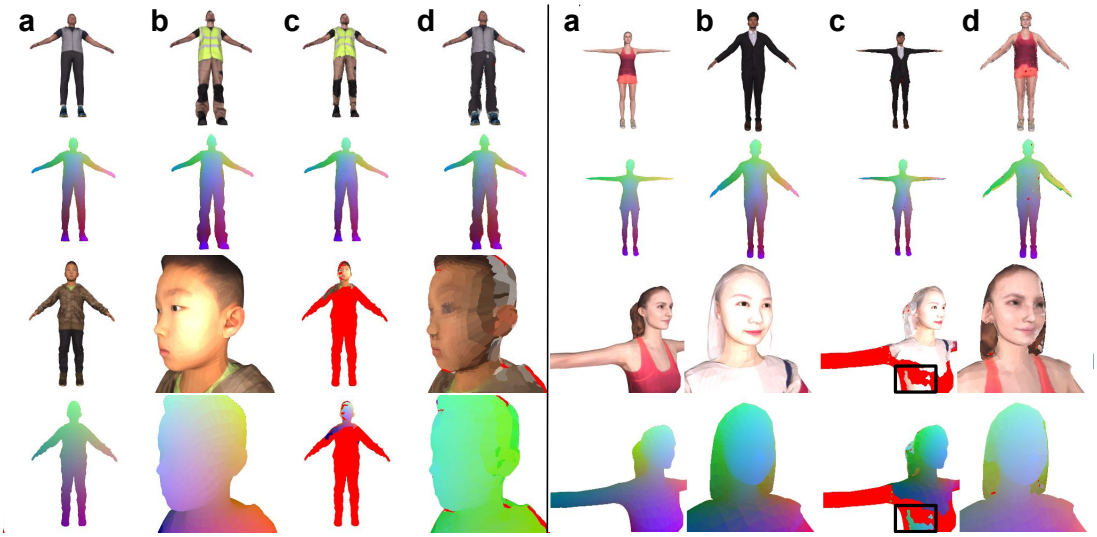


Figure 7.4: **Visualization of learned embeddings.** Given a pair of images in a) and b), our network produces per-pixel embedding for each image. We map pixels from (b) to (a) according to feature similarity, resulting in (c). Similarly d) is generated by transferring texture from (a) to (b). For pixels which have similarity below a threshold are colored red. We visualize the smoothness of our learned correspondence in the second and forth row. Our method learns to produce correct correspondences between human subject in different clothing and same human subject in different camera poses (left). Our approach also finds correct correspondences between different human subjects in different poses (right). Mistakes are highlighted using black boxes.

Our self-supervision task is based on enforcing consistency in pixel embeddings across views for pixels that corresponds to the same point in 3D. In Figure 7.4 we visualize corre-

spondences using our learned embeddings between human subjects from the RenderPeople dataset in different costumes and poses. The smoothness and consistency in correspondences implies that the network can be fine-tuned with few labeled examples and still perform well on unseen examples.

Methods	**Mean**	**Fau.**	**Vase**	**Earph.**	**Knife**	**Bed**	**Bot.**	**Dishw.**	**Clock**	**Door**	**Micro.**	**Fridge**	**Stor.F.**	**Trash**	**Dis**
# semantic parts		12	6	10	10	15	9	7	11	5	6	7	24	11	4
(3D) NN transfer	24.4	22.7	23.3	22.7	17.6	14.1	24.6	31.4	16.6	20.1	30.0	25.1	11.5	22.3	60.4
(3D) NN transfer (aniso)	25.7	22.4	29.0	23.7	17.9	14.3	29.3	34.1	13.2	19.2	31.1	26.6	15.7	23.8	59.5
(2D) Scratch	32.0	29.9	31.5	32.4	25.0	27.3	30.5	34.1	19.2	26.7	35.1	26.8	19.3	33.6	76.4
(2D) ImageNet	32.8	30.1	34.5	33.1	23.8	29.2	30.8	32.9	20.1	28.1	36.1	27.5	19.6	34.7	**78.7**
(2D) DenseCL [238]	34.2	**31.4**	35.4	33.6	22.7	30.8	33.7	36.7	19.7	28.9	41.9	30.2	21.2	34.7	78.3
(3D) Scratch	30.3	27.7	28.8	28.4	19.8	24.5	25.8	39.4	15.9	24.3	37.7	30.9	23.5	30.0	67.8
(3D) Deformation [232]	27.5	28.4	27.2	24.7	20.6	12.4	34.7	30.9	17.4	26.1	38.8	24.6	14.2	21.0	63.8
(3D) PointContrast [189]	34.1	29.0	35.8	31.0	**25.6**	27.8	32.5	39.9	**22.8**	**29.1**	41.3	**32.5**	**25.2**	31.1	73.4
(2D+3D) MVDECOR	**35.9**	31.1	**39.1**	**34.8**	25.2	**32.4**	**39.2**	**40.0**	20.7	28.7	**44.3**	29.8	22.6	**36.3**	78.2

Table 7.1: **Few-shot segmentation on partnet dataset with limited labeled shapes.** 10 fully labeled shapes are provided for training. Evaluation is done on test set of PartNet using mean part-iou metric (%). Training is done on each category separately and results are reported by averaging over 5 random runs.

Methods	**Mean**	**Fau.**	**Vase**	**Earph.**	**Knife**	**Bed**	**Bot.**	**Dishw.**	**Clock**	**Door**	**Micro.**	**Fridge**	**Stor.F.**	**Trash**	**Dis**
# semantic parts		12	6	10	10	15	9	7	11	5	6	7	24	11	4
(2D) Scratch	25.9	21.7	25.2	26.1	19.3	19.8	25.0	27.3	16.6	25.0	27.9	22.3	13.2	24.2	68.7
(2D) ImageNet	27.1	23.2	27.9	28.1	20.1	21.2	25.4	28.2	16.6	25.6	29.3	23.4	12.6	27.4	70.1
(2D) DenseCL [238]	28.9	23.9	31.3	29.0	21.3	22.4	28.9	29.6	16.4	27.4	33.6	25.0	15.9	28.4	71.9
(3D) Scratch	17.1	14.8	17.6	16.4	12.1	8.2	15.7	19.0	7.7	20.7	20.9	15.8	6.8	10.6	52.9
(3D) PointContrast [189]	28.4	22.3	32.5	28.6	21.2	18.9	25.9	**31.3**	**18.9**	**28.5**	31.4	24.8	15.5	25.8	**72.1**
(2D+3D) MVDECOR	**30.3**	**25.5**	**33.7**	**31.6**	**22.4**	**24.9**	**31.7**	31.0	16.2	25.8	**35.7**	**25.6**	**17.0**	**31.4**	71.2

Table 7.2: **Few-shot segmentation on PartNet dataset with limited labeled 2D views.** With 10 shapes, each containing $v = 5$ randomly selected labeled views provided for training. Evaluation is done on test set of PartNet using mean part-iou metric (%). Training is done on each category separately and results are reported by averaging over 5 random runs.

7.4.5 Few-shot segmentation on PartNet

7.4.5.0.1 ***k fully labeled shapes.*** Table 7.1 shows results using the part mIOU metric on few-shot semantic segmentation on PartNet dataset using $k = 10$ fully labeled shapes. We first start with a baseline that transfers labels labels from training set to test set by retrieving a training shape most similar to a test shape using Chamfer distance. We create two versions of this baseline–1) without any deformation of shapes and 2) with anisotropic scaling to unit length in all dimensions of shapes. The anisotropic scaling leads to slight improvement

Methods	k=30, v=all				k=30, v=5			
	Mean	**Chair**	**Table**	**Lamp**	**Mean**	**Chair**	**Table**	**Lamp**
# semantic labels		39	51	41		39	51	41
(2D) Scratch	13.7	20.8	10.1	10.3	10.6	15.2	7.3	9.2
(2D) Imagenet	13.7	20.4	9.9	10.8	11.4	16.4	7.7	10.1
(2D) DenseCL [238]	15.7	22.6	11.7	**12.6**	12.0	17.1	8.5	**10.4**
(3D) Scratch 3d	11.5	17.8	8.0	8.7	6.3	10.1	4.5	4.3
(3D) Deformation [232]	6.5	8.4	4.9	6.1	-	-	-	-
(3D) Pointcontrast [189]	14.5	23.0	10.8	9.8	11.8	**20.4**	7.8	7.1
(2D+3D) Ours	**16.6**	**25.3**	**12.9**	11.7	**12.8**	19.3	**9.8**	9.3

Table 7.3: **Few-shot segmentation on the PartNet dataset. Left:** 30 fully labeled shapes are provided for training. **Right:** 30 shapes are used for training, each containing $v = 5$ randomly selected labeled views. Evaluation is done on test set of PartNet using mean part-iou metric (%) and results are reported by averaging over 5 random runs.

over un-scaled version. This baseline performs significantly worse in comparison to training a network from scratch. This shows that a part-agnostic nearest-neighbor transfer is not sufficient to get good performance in fine-grained semantic segmentation under few-shot setting. Our method performs better than the network trained from scratch by 4% (part mIOU) showing the effectiveness of our self-supervision approach. Our architecture initialized with ImageNet pre-trained weights, improves performance over training from scratch, implying pre-training on large labeled datasets is helpful even when the domain is different. The DenseCL baseline, which is trained on our dataset, improves performance over ImageNet pre-trained weights, owing to the effectiveness of contrastive learning at instance level and at dense level. Interestingly, 2D training from scratch performs better than 3D training from scratch. The learned deformation based alignment approach [232] performs worse because aligning shapes of different topology and structure does not align semantic parts well. Furthermore, alignment is agnostic to the difference in the set of fine-grained semantic parts between shapes. The 3D sparse convolution network pre-trained using point contrastive learning on our dataset and fine-tuned with few labeled shapes performs better than all previous baselines. Finally our approach, that uses dense contrastive learning at pixel level outperforms all baselines.

7.4.5.0.2 ***Sparse labeled 2D views.*** Table 7.2 shows the results on few-shot semantic segmentation on PartNet dataset using sparse 2D views for supervision. Here, the DenseCL

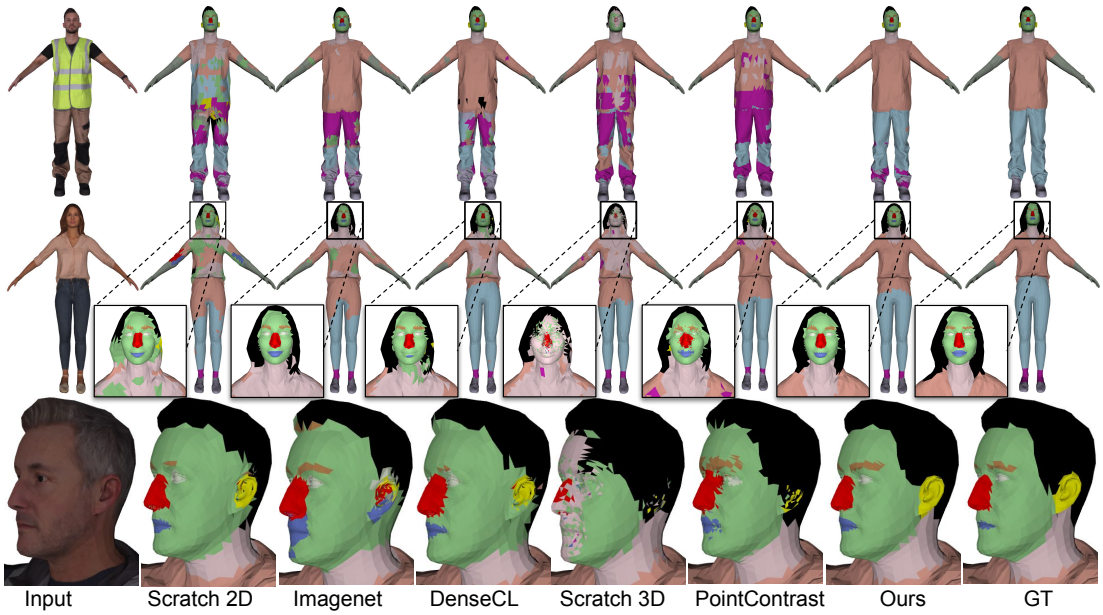


Figure 7.5: **Visualization of predicted semantic labels on Renderpeople dataset in few-shot setting when $k = 5$ fully labeled shapes are used in fine-tuning.** We visualize the predictions of all baselines. Our method produces accurate semantic labels for 3D shapes, even for small parts such as ears and eyebrows.

baseline outperforms training from scratch and the ImageNet initialized network. DenseCL also outperforms the 3D PointContrast baseline, showing the effectiveness of 2D architectures and 2D self-supervision. Finally, our approach outperforms all baselines.

Note that evaluation on Chairs, Lamps and Tables categories is shown separately in Table 7.3 with $k = 30$, because our randomly selected $k = 10$ shapes do not cover all the part labels of these classes. In this setting our approach outperforms the baselines.

7.4.6 Few-shot segmentation on RenderPeople

To evaluate our method on the textured dataset, we use the RenderPeople dataset. We use the same set of 2D and 3D baselines as descrived in Section 7.4.3. Note that, we provide color and normal with point cloud input to 3D Scratch and 3D PointContrast. In addition to the settings described in S7.4.2, we further analyze the effect of different inputs given to the network, *i.e.* when only RGB images are used as input to the network for self-supervision and fine-tuning, and when both RGB images and geometry information (normal + depth maps) are available for self-supervision and fine-tuning. We train all baselines in these two

Methods	RGB				RGB+Geom.			
	k=5, v=all	k=10, v=all	k=5,v=3	k=10,v=3	k=5, v=all	k=10, v=all	k=5,v=3	k=10,v=3
(2D) Scratch	50.2	60.5	38.7	46.3	55.3	62.6	40.6	50.4
(2D) ImageNet	58.8	67.6	48.9	58.1	55.3	63.7	44.3	51.9
(2D) DenseCL [238]	58.3	66.8	46.5	55.5	56.0	64.0	31.0	41.5
(3D) Scratch	-	48.1	-	26.1	35.0	-	14.5	-
(3D) PointContrast [189]	-	61.3	-	56.7	53.0	-	48.5	-
(2D+3D) MVDECOR	**67.5**	**73.8**	**59.6**	**67.4**	**58.8**	**65.0**	**50.3**	**55.1**

Table 7.4: **Few-shot segmentation on RenderPeople dataset.** We evaluate the segmentation performance using part mIOU metric. We experiment with two kinds of input, 1) when both RGB+Geom. (depth and normal maps) are input and 2) when only RGB is input to the network. We evaluate all methods when $k = 5, 10$ fully labeled shapes are used for supervision and when $k = 5, 10$ shapes with 3 2D views are available for supervision. MVDECOR consistently outperform baselines on all settings.

settings, except 3D baselines that take geometry by construction. The results are shown in Table 7.4.

7.4.6.0.1 ***RGB+Geom.*** In the first setting when RGB is used as input along with geometry information (normal + depth), our approach outperforms all the baselines, with 3.5% and 9.7% improvement on training from scratch when only $k = 5$ labeled shapes are given and when $k = 5$ shapes with $v = 3$ views are given for supervision respectively. We use only 3 views for RenderPeople dataset because of its simpler topology in comparison to 5 views for PartNet . The ImageNet pre-trained model, which is modified to take depth and normal maps as input performs similar to training from scratch, that implies that the domain shift is too large between ImageNet and our dataset. DenseCL applies dense correspondence learning at coarse grid and hence does not perform well in the dense prediction task when only a few labeled examples are given.

7.4.6.0.2 ***RGB only.*** In the second setting, when only RGB image is input to the network, MVDECOR gives 17.3% and 20.9% improvement over training from scratch when only $k = 5$ labeled shapes are given and when $k = 5$ shapes with $v = 3$ views are are given for supervision respectively. The ImageNet and DenseCL both perform better than training from scratch, including their counterpart which takes both RGB+geometry as input. MVDECOR with only RGB as input also performs significantly better than its RGB+geometry counter-

Method	RGB+Geom.	RGB
MVDECOR w/o closeup views	51.6	57.4
MVDECOR w/o reg.	58.0	67.2
MVDECOR	58.8	67.5

Table 7.5: **Effect of selection of renderings and regularization on RenderPeople dataset.** MVDECOR without closeup views used for pre-training and fine-tuning performs worse than when closeup views are used. Our regularization term in the loss also shows improvement.

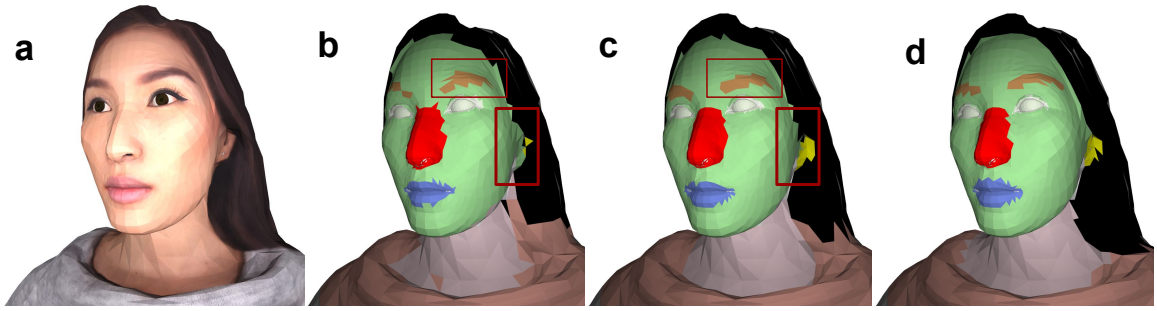


Figure 7.6: **View aggregation.** Given the input in (a), MVDECOR produces 2D labels (b) which can further be improved by multi-view aggregation (c) as is highlighted in boxes and produces segmentation close to the ground truth (d).

part. We expect this behaviour because of the following: first when geometry is also used as input to the network, the pre-training task focuses more on geometry to produce consistent embeddings, as is shown in Figure 7.4, where consistent embeddings are produced between the same human subject in two different costumes. However, when only RGB is input to the network, the pretraining task focuses on RGB color only to learn correspondences. Second, the semantic segmentation of human models requires high reliance on RGB features compared to geometry, and the additional geometry input tends to confuse the pre-trained network.

Figure 7.5 shows qualitative results of different methods. We consistently outperform all baselines and produce correct segmentation for tiny parts such as eyes, ears and nose. Refer to the Appendix for more qualitative visualization. In Figure 7.6 we show the effect of multi-view aggregation on 3D segmentation.

7.4.6.0.3 ***Regularization.*** During fine-tuning stage we use an extra regulrization term $\lambda \mathcal{L}_{\text{ssl}}$ applied on shapes from $\mathcal{X}_u$, to prevent network from overfitting on small training set

$\mathcal{X}_l$ as described in Section 7.3.2. In Table 7.5 we show that regularization improves our performance on RenderPeople dataset.

7.4.6.0.4 ***Effect of view selection.*** We also analyze the effect of view selection. In our previous experiment, we select views by placing camera farther away from the shape to obtain a full context along with placing the camera close to the shape to get finer details. Now we remove close-up views during pre-training and fine-tuning stage and report results in Table 7.5. We observe that closeup views are important for accurate segmentation of small parts. Finally, we also observe that on RenderPeople dataset, the segmentation performance improves as more views are provided during inference time. Specifically, as number of views are increased from 5 to 96, the segmentation performance improves from 54.7 to 58.8.

7.5 Conclusion

In this paper, we present MVDECOR, a self-supervision method that learns a multi-view representation for 3D shapes with geometry consistency enforced across different views. We pre-train our network with a multi-view dense correspondence learning task, and show that the learned representation outperforms state-of-art methods in the experiments of few-shot fine-grained part segmentation, giving most benefits for textured 3D shapes.

7.5.0.0.1 ***Limitations.*** Our method relies on 2D renderings of 3D shapes, thus a few surface regions may not be covered due to self-occlusion. In this case, the label predictions in these regions are unreliable. Our self-supervision also requires rendering several views of shapes to make the representations view invariant, which increases the computational cost. Our view selection during pre-training and fine-tuning stage is heuristic-based and can be improved by a learnable approach [82]. A useful future avenue is to combine our approach of 2D correspondence learning with 3D correspondence learning [189] to obtain the best of both worlds. MVDECOR may also open up other potential supervision sources, such as reusing existing image segmentation datasets to segment 3D shapes, exploiting motion in videos to provide correspondence supervision for pre-training.

CHAPTER 8

CONCLUSIONS AND FUTURE WORKS

8.1 Conclusions

This thesis attempted to answer two questions. 1) How to design neural network architectures to decompose 3D shapes into programs and surface patches? 2) Does the ability to decompose the 3D shape into parts help learn useful 3D shape representation? Below, I discuss the conclusions from the research conducted in each of the two directions. Then I proceed with future work extensions.

8.1.1 Learning to decompose 3D shapes

We first designed a neural network architecture that decomposes shapes into CSG programs. We demonstrated that the model generalizes across domains, including logos, 2D silhouettes, and 3D CAD shapes. However, not all shapes are suitable to be approximated by CSG programs. Designers often use parametric surface patches while modeling shapes. Towards that end, we presented a neural network architecture to reconstruct point clouds by predicting geometric primitives and surface patches common in CAD design. Our method effectively marries 3D deep learning with CAD modeling practices.

Our learning-based approaches to shape parsing outperform previous non-learning and learning-based methods, resulting in predictions that are robust and incorporate human shape-decomposition priors. Our architecture's predictions are editable, interpretable and compact. Modelers can refine our results based on standard CAD modeling operations, opening several applications in computer graphics.

8.1.2 Learning representation for 3D shape understanding

To alleviate the need of large labeled set used for training neural networks, we proposed three approaches to learn shape representations useful for downstream semantic segmentation tasks.

We presented a method to exploit existing part hierarchies and tag metadata associated with 3D shapes found in online repositories to pre-train deep networks for shape segmentation. These part-hierarchies are often inconsistent as they vary depending on the expertise and goals of the designers creating models. To overcome this problem, we proposed a simple self-supervision task for learning point embeddings – learning to fit an unlabeled point-cloud using a set of geometric primitives such as ellipsoids and cuboids. We provide an end-to-end trainable framework for incorporating this task into a standard network architectures for point cloud segmentation. Finally, in order to learn features for fine-grained semantic segmentation of textured shapes, we proposed a self-supervision method that learns a multi-view representation for 3D shapes with geometry consistency enforced across different views. Our approach outperforms previous state-of-the-art approaches to few-shot semantic segmentation of 3D shapes, which helps in reducing manual and monetary cost of shape annotations.

8.2 Future works

Below we present two possible extensions of the work presented in this thesis.

8.2.1 General program induction for objects and scenes.

Man-made objects that we interact with on a daily basis often are created using CAD software. Given a scan taken of an object, we would like to get a program that can reconstruct the same. Often shapes are made using a combination of many graphics modelling tools– a) polygonal modeling operations (e.g, extrusions, lofting, revolving, boolean operations, sculpting, deformations), b) parametric curves and surfaces, and c) subdivision modeling.

Existing approaches have targeted reverse-engineering one of the above modelling tools at a time. A combination of all these modelling tools can create most objects, providing us with a complete and versatile representation of our surroundings. We will encounter several challenges while solving this problems. First, predicting all these modelling instructions together requires development of *domain-specific languages* capable of handling variety of modeling tools and development of datasets with rich set of modeling instructions for supervision [143]. Second, several modelling operations are non-differentiable, which makes it challenging to include them in the computation graph of modern neural networks and learn using gradient descent. A differentiable approximation [41] of these modelling instructions will make learning these instructions efficient within the neural network pipeline. Finally, producing programs for shapes requires predicting discrete and continuous entities, and often requires efficient search procedures in large search space [53]. Recent advances in continuous relaxation of discrete optimization can help to tackle these problems.

Program induction at scene level opens up applications in the efficient storage of large scene and ease of editing within augmented reality. This helps in creating a form of "version-control" where diffs are calculated using elements of the programs. However, program induction at the level of scene is more challenging because of the added complexity by having several objects. The solution to this problem perhaps lie in the bottom-up approach where outputs from an object detector are combined in hierarchical fashion to reconstruct the scene.

8.2.2 Unified representation learning for dynamic surroundings.

Recently, we have seen excellent progress on representation learning in the image domain [89, 248], giving a superior performance on many image-based tasks. Similarly, several self-supervision methods [189, 203] in 3D (scanned point clouds) have also alleviated the reliance on a large scale labelled dataset. Notice that, scans of scenes and corresponding high-resolution images provide complementary information–3D scans provide sparse information

about the scene, albeit in the native 3D domain. Whereas, high-resolution 2D images better encode texture information of the scene but are devoid of 3D awareness. If a scene is dynamic, then time-varying scans are needed to capture the dynamics of the scene. These time-stamped scans provide motion information, along with non-rigid motion often hints at part segmentation. Furthermore, the process of humans perceiving the surrounding also involves the perception of sounds. Oftentimes, human activities produce sound and are useful for humans to navigate and interact with their surroundings. Thus learning representation for geometry, motion, texture and sound jointly provides a holistic representation of 3D scene. Learning these representations requires innovations in network architectures, dataset, training losses and optimization.

APPENDIX A

NEURAL SHAPE PARSERS FOR CONSTRUCTIVE SOLID GEOMETRY

In this supplementary material, we include the following topics in more detail: a) synthetic dataset creation in the 2D and the 3D case, b) neural network architecture used in our experiments, c) more qualitative results on our test dataset.

A.1 Dataset

A.1.1 Synthetic 2D shapes.

We use the grammar described in the Section 4.1 to create our 2D dataset. The dataset is created by randomly generating programs of lengths 3 to 13 following the grammar. While generating these programs we impose additional restrictions as follows: a) Primitives must lie completely inside the canvas, b) Each operation changes the number of ON pixels by at least a threshold set to 10% of sum of pixels in two shapes. This avoids spurious operations such as subtraction between shapes with little overlap. c) The number of ON pixels in the final image is above a threshold. d) The previous rules promotes programs with the $union$ operation. To ensure a balanced dataset we boost the probabilities of generating programs with $subtract$ and $intersect$ operations. Finally we remove duplicates. We only use upright, equilateral triangles and upright squares. Note that locations (L) are discretized to lie on square grid with spacing of 8 units and size (R) are discretized with spacing of 4 units.

A.1.2 Synthetic 3D shapes.

We use the grammar described in the Section 4.1 to create our 3D dataset. While generating shapes we followed a strategy similar to the 2D case. For 3D case, we only use

Layers	**Output**
Input image + stack	$64 \times 64 \times (k+1)$
Dropout(Relu(Conv: 3×3, $1 \rightarrow 8$))	$64 \times 64 \times 8$
Max-pool(2×2)	$32 \times 32 \times 8$
Dropout(Relu(Conv: 3×3, $8 \rightarrow 16$))	$32 \times 32 \times 16$
Max-pool(2×2)	$16 \times 16 \times 16$
Dropout(Relu(Conv: 3×3, $16 \rightarrow 32$))	$16 \times 16 \times 32$
Max-pool(2×2)	$8 \times 8 \times 32$
Flatten	2048

Table A.1: **Encoder architecture for 2D shapes experiments.** For StackCSGNet $k = 4$ and for CSGNet $k = 0$.

programs of up to length 7 (up to 4 shape primtives and upto 3 boolean operations). Note that the cube and cylinder are upright. The dataset contains $64 \times 64 \times 64$ voxel-grid shapes and program pairs. Also note that locations (L) are discretized to lie on cubic grid with spacing of 8 units, and size (R) and height (H) are discretized with spacing of 4 units.

A.1.3 CSG execution engine.

We implemented a CSG engine that reads the instructions one by one. If it encounters a primitive (e.g. `c(32, 32, 16)`) it draws it on an empty canvas and pushes it on to a stack. If it encounters an operation (e.g. `union`, `intersect`, or `subtract`) it pops the top two canvases on its stack, applies the operation to them, and pushes the output to the top of the stack. The execution stops when no instructions remain at which point the top canvas represents the result. The above can be seen as a set of shift and reduce operations in a LR-parser [123]. Figure A.1 describes execution procedure to induce programs for 3D shapes.

A.2 Network Architecture

A.2.1 Architecture for 2D shape experiments.

Table A.1 shows the CNN architecture used as the encoder. The input I is an image of size 64×64 concantenated with top-k elements of stack S_t of size $64 \times 64 \times k$. Note that

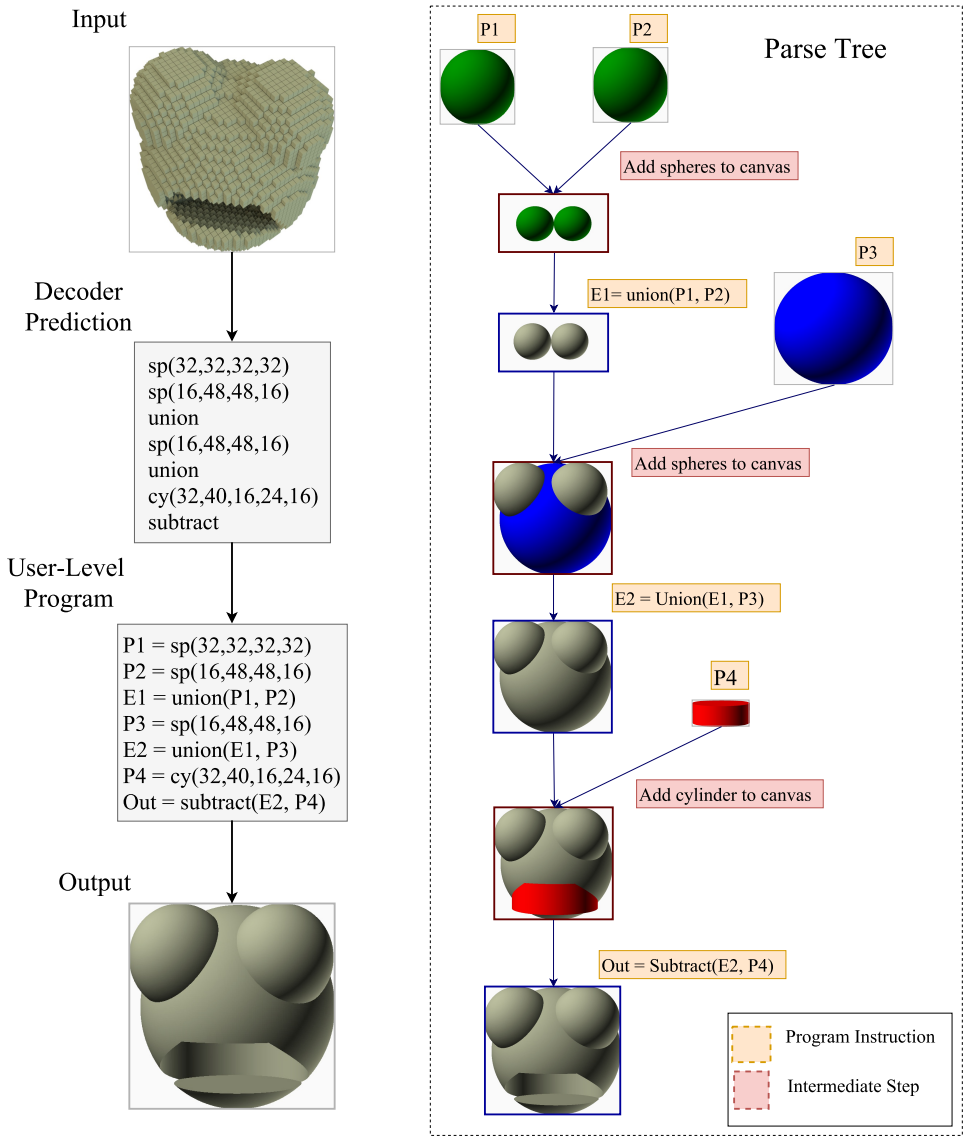


Figure A.1: **Detailed execution procedure followed by an induced CSG program in a characteristic 3D case.** The input is a voxel based representation of size $64 \times 64 \times 64$. The RNN decoder produces a program, which can be executed following the grammar described in the Section A.1, to give the output shown at the bottom. The user-level program is shown for illustration. On the right side is shown a parse tree corresponding to the execution of the program.

stack is input to our StackCSGNet architecture and no stack is input to CSGNet architecture where $k = 0$. The output $\Phi(I)$ is a vector of size 2048. Table A.2 describes the architecture used in the decoder. The RNN decoder is based on a GRU unit that at every time step takes as input the encoded feature vector and previous instruction encoded as a 128 dimensional

Index	Layers	Output
1	Input shape encoding	2048
2	Input previous instruction	401
3	Relu(FC ($401 \rightarrow 128$))	128
4	Concatenate (1, 3)	2176
5	Drop(GRU (hidden dim: 2048))	2048
6	Drop(Relu(FC($2048 \rightarrow 2048$)))	2048
7	Softmax(FC($2048 \rightarrow 400$))	400

Table A.2: **Decoder architecture for 2D shapes experiments.** The same architecture is used for all both CSGNet and StackCSGNet in our experiments in the Section 4.3.1. FC: Fully connected dense layer, Drop: dropout layer with 0.2 probability. Dropout on GRU are applied on outputs but not on recurrent connections.

vector obtained by a linear mapping of the 401 dimensional one-hot vector representation. At first time step, the previous instruction vector represents the `START` symbol. Embedded vector of previous instruction is concantenated with $\Phi(I)$ and is input to the GRU. The hidden state of GRU is passed through two dense layer to give a vector of dimension 400, which after `softmax` layer gives a probability distribution over instructions. The output distribution is over 396 different shape primitives, 3 operations (`intersect`, `union` and `subtract`) and a `STOP`. We exclude the `START` symbol from the output probability distribution. Note that the circle, triangle or square at a particular position in the image and of a particular size represents an unique primitive. For example, $c(32, 32, 16)$, $c(32, 28, 16)$, $s(12, 32, 16)$ are different shape primitives.

A.2.2 Architecture for 3D shape experiments.

The input I is a voxel representation of 3D shape of size $64 \times 64 \times 64$ concantenated with top-k elements of stack S_t of size $64 \times 64 \times 64 \times k$. Note that stack is input to our 3D-StackCSGNet architecture and no stack is input to 3D-CSGNet architecture where $k = 0$. The outputs is an encoded vector $\Phi(I)$ of size 2048, as shown in the Table A.3. Similar to the 2D case, at every time step, GRU takes as input the encoded feature vector and previous ground truth instruction. The previous ground truth instruction is a 6636-

Layers	**Output**
Input Voxel + stack	$64 \times 64 \times 64 \times (k+1)$
Relu(Conv3d: $4 \times 4 \times 4$, $1 \rightarrow 32$)	$64 \times 64 \times 64 \times 32$
BN(Drop(Max-pool($2 \times 2 \times 2$)))	$32 \times 32 \times 32 \times 32$
Relu(Conv3d: 4×4, $32 \rightarrow 64$)	$32 \times 32 \times 32 \times 64$
BN(Drop(Max-pool($2 \times 2 \times 2$)))	$16 \times 16 \times 16 \times 64$
Relu(Conv3d: 3×3, $64 \rightarrow 128$))	$16 \times 16 \times 16 \times 128$
BN(Drop(Max-pool($2 \times 2 \times 2$)))	$8 \times 8 \times 8 \times 128$
Relu(Conv3d: 3×3, $128 \rightarrow 256$))	$8 \times 8 \times 8 \times 256$
BN(Drop(Max-pool($2 \times 2 \times 2$)))	$4 \times 4 \times 4 \times 256$
Relu(Conv3d: 3×3, $256 \rightarrow 256$))	$4 \times 4 \times 4 \times 256$
BN(Drop(Max-pool($2 \times 2 \times 2$)))	$2 \times 2 \times 2 \times 256$
Flatten	2048

Table A.3: **Encoder architecture for 3D shape experiments.** Drop: dropout layer, BN: batch-normalization layer and Drop: dropout layer with 0.2 probability. For 3D-StackCSGNet $k = 1$ and for 3D-CSGNet $k = 0$.

Index	**Layers**	**Output**
1	Input shape encoding	2048
2	Input previous instruction	6636
3	Relu(FC($6636 \rightarrow 128$))	128
4	Concatenate (1, 3)	2176
5	Drop(GRU (hidden dim: 1500))	1500
6	Drop(Relu(FC($1500 \rightarrow 1500$)))	1500
7	Softmax(FC($1500 \rightarrow 6635$))	6635

Table A.4: **Decoder network architecture for 3D shapes experiments.** FC: Fully connected dense layer, Drop: dropout layer with 0.2 probability. Dropout on GRU are applied on outputs but not on recurrent connections. Same decoder is used for both 3D-CSGNet and 3D-StackCSGNet.

dimensional (also includes the `start` symbol) one-hot vector, which gets converted to a fixed 128-dimensional vector using a learned embedding layer. At first time step the last instruction vector represents the `START` symbol. Embedded vector of previous instruction is concatenated with $\Phi(I)$ and is input to the GRU. The hidden state of GRU is passed through two dense layers to give a vector of dimension 6635, which after Softmax layer gives a probability distribution over instructions. The output distribution is over 6631 different shape primitives, 3 operations (`intersect`, `union` and `subtract`) and a `STOP`. We exclude the `START` symbol from the output probability distribution. Similar to 2D case, $cu(32, 32, 16, 16)$, $cu(32, 28, 16, 12)$, $sp(12, 32, 16, 28)$ are different shape primitives. Table A.4 shows details of decoder.

A.3 Qualitative Evaluation

In this section, we show more qualitative results on different dataset. We first show peformance of our CSGNet trained using only Supervised learning on 2D synthetic dataset, and we compare top-10 results from nearest neighbors and and top-10 results from beam search, refer to the Figure A.2 and A.3. Then we show performance of our full model (using RL + beam search + visually guided search) on CAD 2D shape dataset, refer to the Figure A.4 and A.5.

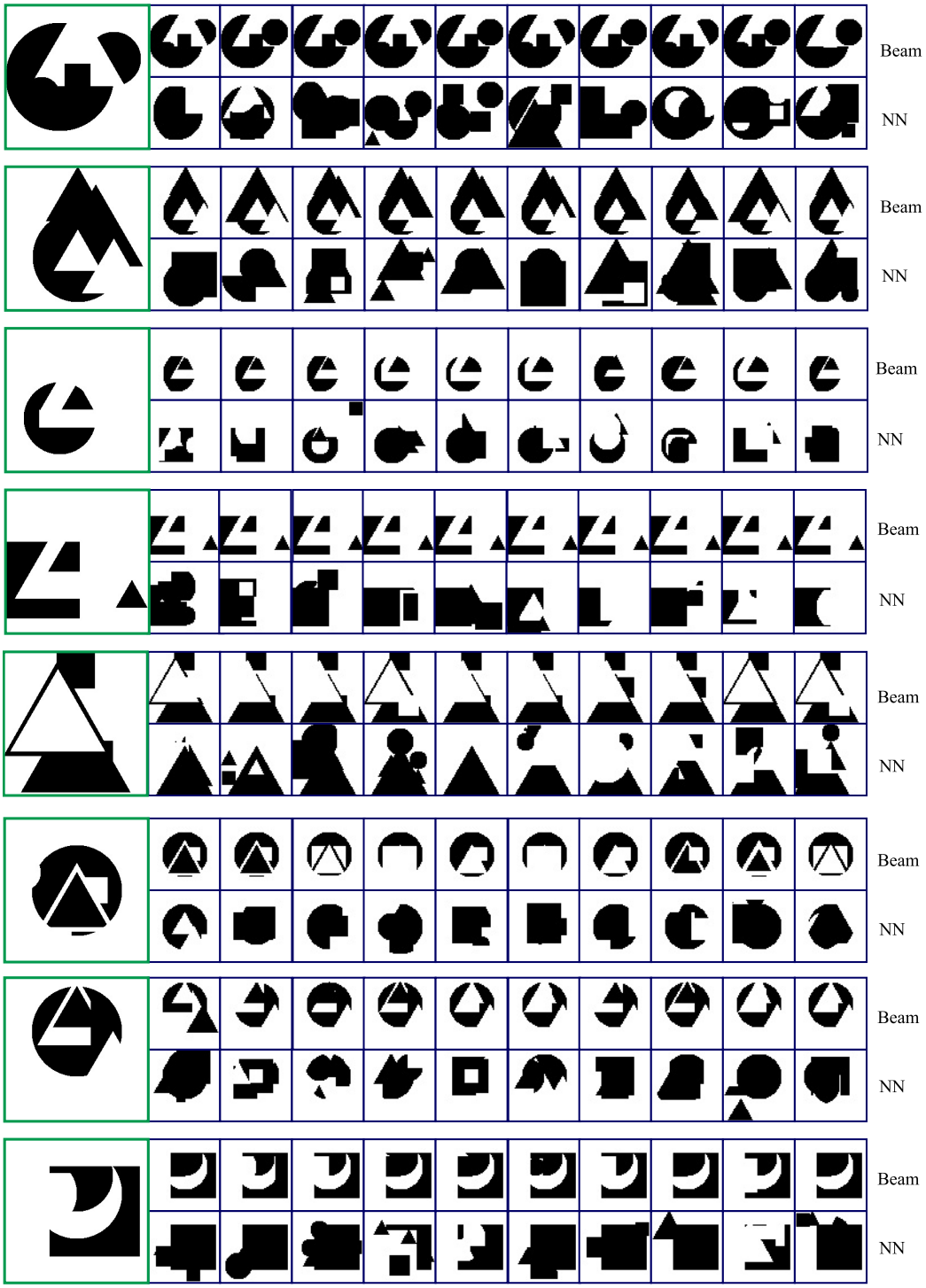


Figure A.2: **Qualitative evaluation on 2D synthetic dataset.** In green outline is the groundtruth, top row represent top-10 beam search results, bottom row represents top-10 nearest neighbors.

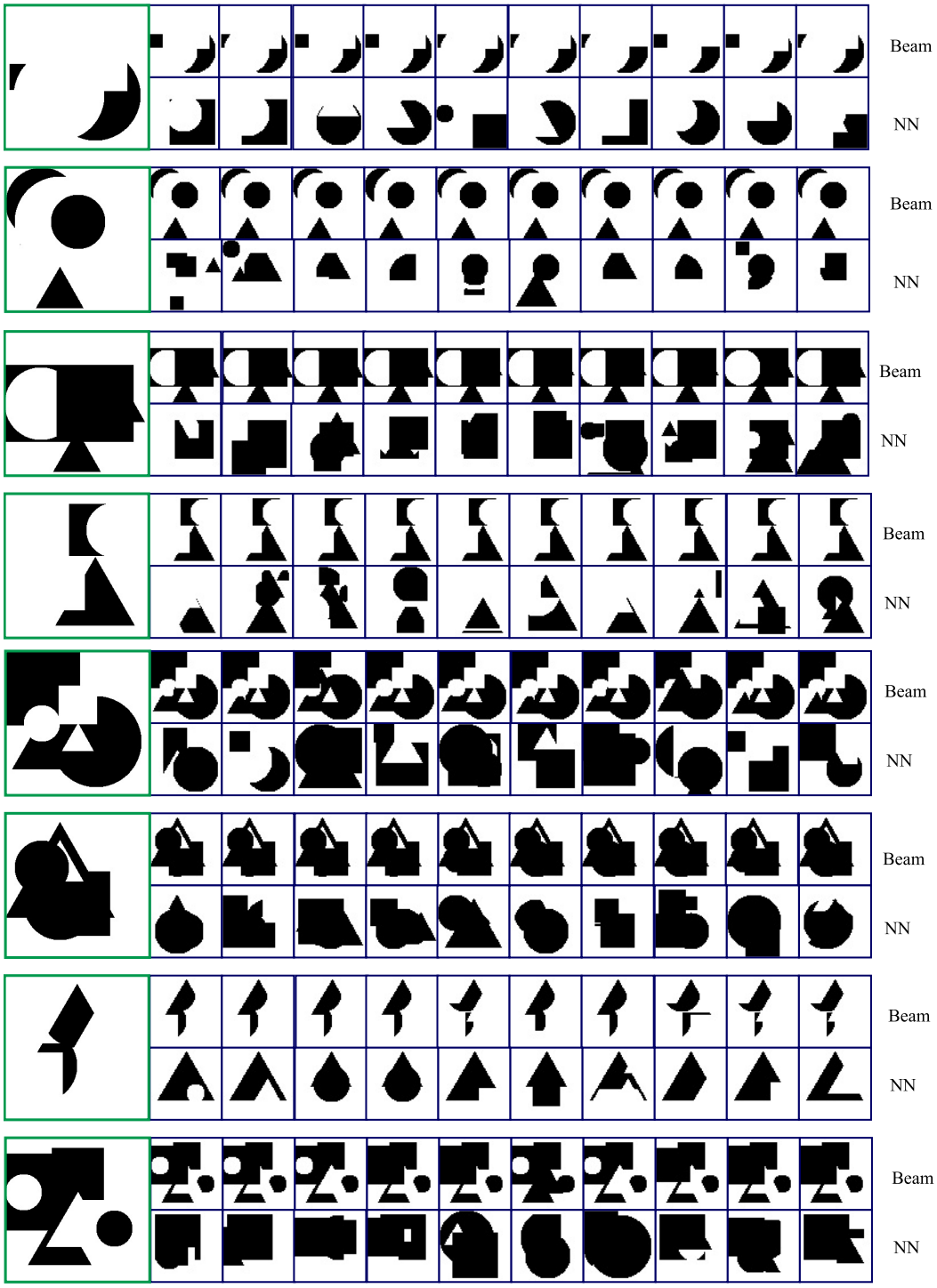


Figure A.3: **Qualitative evaluation on 2D synthetic dataset.** In green outline is the groundtruth, top row represent top-10 beam search results, bottom row represents top-10 nearest neighbors.

Figure A.4: **Performance of our full model on 2D CAD images**. a) Input image, b) output from our full model, c) Outlines of primitives present in the generated program, triangles are in green, squares are in blue and circles are in red d) Predicted program. s, c and t are shape primitives that represents $square$, $circle$ and $triangle$ respectively, and $union$, $intersect$ and $subtract$ are boolean operations.

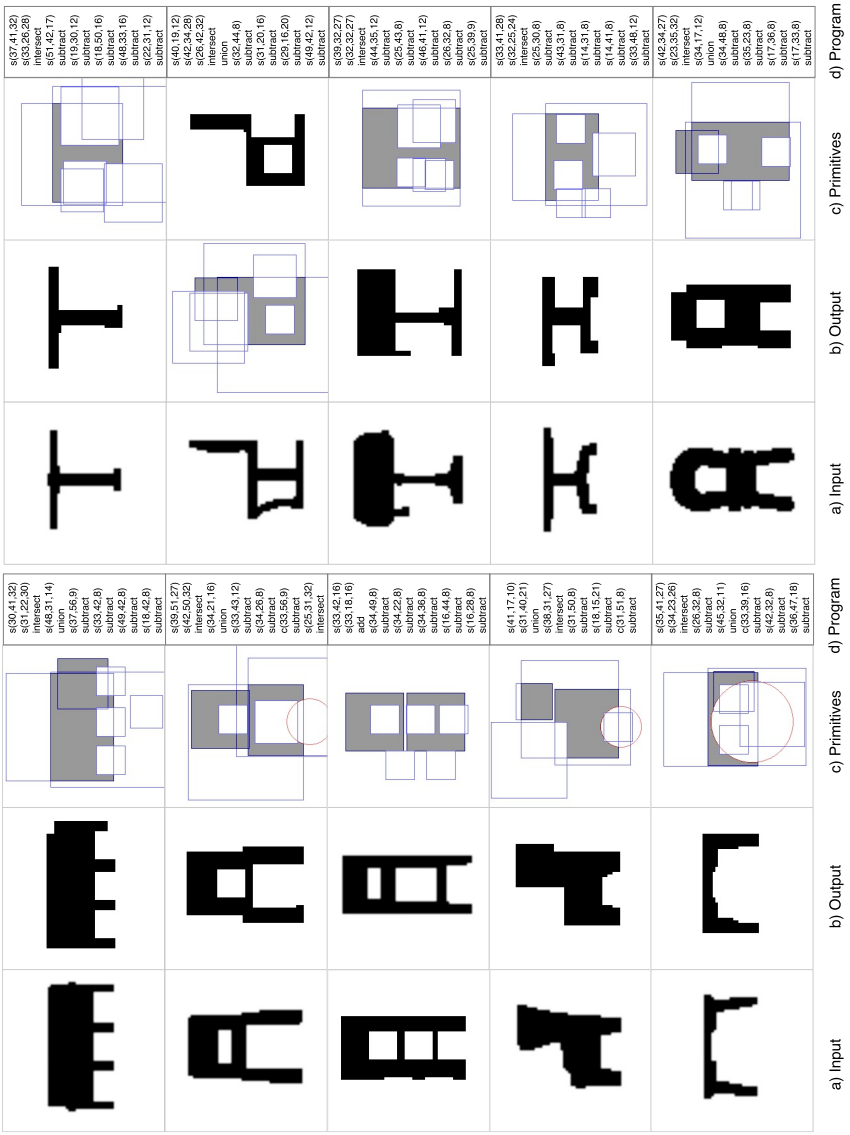


Figure A.5: **Performance of our full model on 2D CAD images**. a) Input image, b) output from our full model, c) Outlines of primitives present in the generated program, triangles are in green, squares are in blue and circles are in red d) Predicted program. s, c and t are shape primitives that represents $square$, $circle$ and $triangle$ respectively, and $union$, $intersect$ and $subtract$ are boolean operations.

APPENDIX B

PARSENET: PARAMETRIC SURFACE FITTING FOR 3D POINT CLOUDS

In our Supplementary Material, we:

- provide background on B-spline patches;
- provide further details about our dataset, architectures and implementation;
- evaluate the robustness of SPLINENET as a function of point density;
- evaluate our approach for reconstruction on the ABCPARTSDATASET;
- show more visualizations of our results; and
- evaluate the performance of our approach on the TraceParts dataset [139].

B.1 Background on B-spline patches.

A B-spline patch is a smoothly curved, bounded, parametric surface, whose shape is defined by a sparse grid of control points $\mathrm{C} = \{\mathbf{c}_{p,q}\}$. The surface point with parameters $(u, v) \in [u_{\min}, u_{\max}] \times [v_{\min}, v_{\max}]$ is given by:

$$\mathbf{s}(u, v) = \sum_{p=1}^{P} \sum_{q=1}^{Q} b_p(u) b_q(v) \mathbf{c}_{p,q} \tag{B.1}$$

where $b_p(u)$ and $b_q(v)$ are polynomial B-spline *basis functions* [57].

To determine how the control points affect the B-spline, a sequence of parameter values, or *knot vector*, is used to divide the range of each parameter into intervals or *knot spans*.

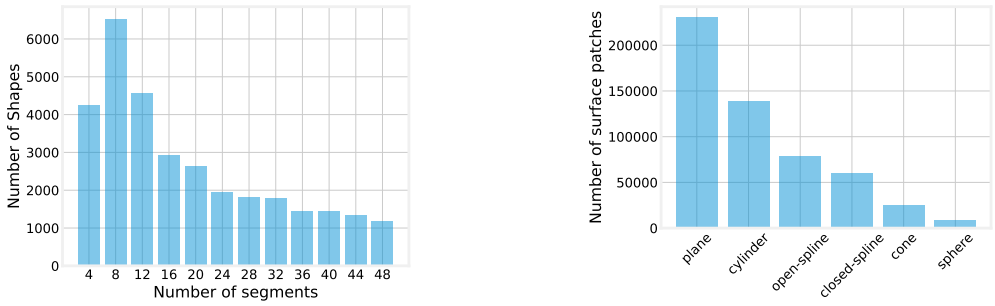


Figure B.1: **Histogram of surface patches in** ABCPARTSDATASET. Left: shows histogram of number of segments and Right: shows histogram of primitive types.

Whenever the parameter value enters a new knot span, a new row (or column) of control points and associated basis functions become active. A common knot setting repeats the first and last ones multiple times (specifically 4 for cubic B-splines) while keeping the interior knots uniformly spaced, so that the patch interpolates the corners of the control point grid. A closed surface is generated by matching the control points on opposite edges of the grid. There are various generalizations of B-splines *e.g.*, with rational basis functions or non-uniform knots. We focus on predicting cubic B-splines (open or closed) with uniform interior knots, which are quite common in CAD [57, 61, 175, 194].

B.2 Dataset

The ABCPARTSDATASET is a subset of the ABC dataset obtained by first selecting models that contain at least one B-spline surface patch. To avoid over-segmented shapes, we retain those with up to 50 surface patches. This results in a total of $32k$ shapes, which we further split into training ($24k$), validation ($4k$), and test ($4k$) subsets. Figure B.1 shows the distribution of number and type of surface patches in the dataset.

B.3 Implementation Details of PARSENET

B.3.0.0.1 Architecture details. Our decomposition module is based on a dynamic edge convolution network [239]. The network takes points as input (and optionally normals)

and outputs a per point embedding $Y \in R^{N\times 128}$ and primitive type $T \in R^{N\times 6}$. The layers of our network are listed in Table B.1. The edge convolution layer (EdgeConv) takes as input a per-point feature representation $f \in \mathbb{R}^{N\times D}$, constructs a kNN graph based on this feature space (we choose $k = 80$ neighbors), then forms another feature representation $h \in \mathbb{R}^{N\times k\times 2D}$, where $h_{i,j} = \left[f_i, f_i - f_j\right]$, and i,j are neighboring points. This encodes both unary and pairwise point features, which are further transformed by a MLP ($D \to D'$), Group normalization and LeakyReLU (slope=0.2) layers. This results in a new feature representation: $h' \in \mathbb{R}^{N\times k\times D'}$. Features from neighboring points are max-pooled to obtain a per point feature $f' \in \mathbb{R}^{N\times D'}$. We express this layer which takes features $f \in \mathbb{R}^{N\times D}$ and returns features $f' \in \mathbb{R}^{N\times D'}$ as EdgeConv(f, D, D'). Group normalization in EdgeConv layer allows the use of smaller batch size during training. Please refer to [239] for more details on edge convolution network.

SPLINENET is also implemented using a dynamic graph CNN. The network takes points as input and outputs a grid of spline control points that best approximates the input point cloud. The architecture of SPLINENET is described in Table B.2. Note that the EdgeConv layer in this network uses batch normalization instead of group normalization.

B.3.1 Training details.

We use the Adam optimizer for training with learning rate 10^{-2} and reducing it by the factor of two when the validation performance saturates. For the EdgeConv layers of the decomposition module, we use 100 nearest neighbors, and 10 for the ones in SPLINENET. For pre-training SPLINENET on SPLINEDATASET, we randomly sample $2k$ points from the B-spline patches. Since ABC shapes are arbitrarily oriented, we perform PCA on them and align the direction corresponding to the smallest eigenvalue to the $+x$ axis. This procedure does not guarantee alignment, but helps since it reduces the orientation variability in the dataset. For pre-training the decomposition module and SPLINENET we augment the

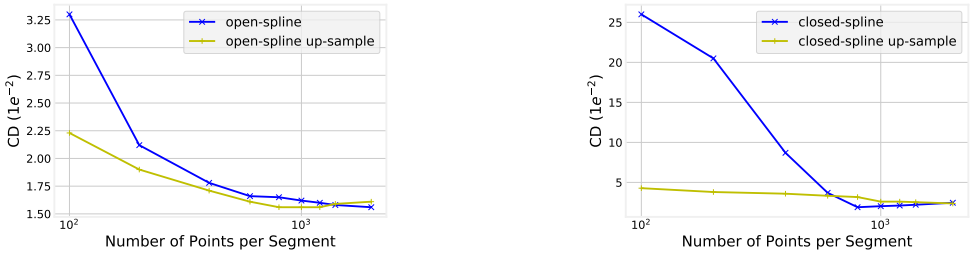


Figure B.2: **Robustness analysis of** SPLINENET**.** Left: open B-spline and Right: closed B-spline. Performance degrades for sparse inputs (blue curve). Nearest neighbor up-sampling of the input point cloud to $1.6K$ points reduces error for sparser inputs (yellow curve). The horizontal axis is in log scale. The error is measured using Chamfer distance (CD).

training shapes represented as points by using random jitters, scaling, rotation and point density.

B.3.2 Back propagation through mean-shift clustering.

The W matrix is constructed by first applying non-max suppression (NMS) on the output of mean shift clustering, which gives us indices of K cluster centers. NMS is done externally *i.e.* outside our computational graph. We use these indices and Eq. 9 to compute the W matrix. The derivatives of NMS w.r.t point embeddings are zero or undefined (i.e. non-differentiable). Thus, we remove NMS from the computational graph and back-propagate the gradients through a partial computation graph, which is differentiable. This can be seen as a straight-through estimator [196]. A similar approach is used in back-propagating gradients through Hungarian Matching in [139]. Our experiments in Table 1 shows that this approach for end-to-end training is effective. Constructing a fixed size matrix W will result in redundant/unused columns because different shapes have different numbers of clusters. Possible improvements may lie in a continuous relaxation of clustering similar to differentiable sorting and ranking [41], however that is out of scope for our work.

Index	Layer	out
1	Input	N × 3
2	EdgeConv(out(1), 3, 64)	N × 64
3	EdgeConv(out(2), 64, 64)	N × 64
4	EdgeConv(out(3), 64, 128)	N × 128
5	CAT(out(2), out(3), out(4))	N × (256)
6	RELU(GN(FC(out(5), 1024)))	N × 1024
7	MP(out(6), N, 1)	1024
8	Repeat(out(7), N)	N × 1024
9	CAT(out(8), out(5))	N × 1280
10	RELU(GN(FC(out(9), 512)))	N × 512
11	RELU(GN(FC(out(10), 256)))	N × 256
12	RELU(GN(FC(out(11), 256)))	N × 256
13	**Embedding**=Norm(FC(out(12), 128))	N × 128
14	RELU(GN(FC(out(11), 256))	N × 256
15	**Primitive-Type**=Softmax(FC(out(14), 6))	N × 6

Table B.1: **Architecture of the Decomposition Module.** EdgeConv: edge convolution, GN: group normalization, RELU: rectified linear unit, FC: fully connected layer, CAT: concatenate tensors along the second dimension, MP: max-pooling along the first dimension, Norm: normalizing the tensor to unit Euclidean length across the second dimension.

B.4 Robustness analysis of SPLINENET

Here we evaluate the performance of SPLINENET as a function of the point sampling density. As seen in Figure B.2, the performance of SPLINENET is low when the point density is small (100 points per surface patch). SPLINENET is based on graph edge convolutions [239], which are affected by the underlying sampling density of the network. However, upsampling points using a nearest neighbor interpolation leads to a significantly better performance.

B.5 Evaluation of Reconstruction using Chamfer Distance

Here we evaluate the performance of PARSENET and other baselines for the task of reconstruction using Chamfer distance on ABCPARTSDATASET. Chamfer distance between reconstructed points P and input points $\hat{P}$ is defined as:

Index	Layer	Output
1	Input	N × 3
2	EdgeConv(out(1),3, 128)	N × 128
3	EdgeConv(out(2),128, 128)	N × 128
4	EdgeConv(out(3),128, 256)	N × 256
5	EdgeConv(out(4),256, 512)	N × 512
6	CAT(out(2), out(3), out(4), out(5)))	N × (1152)
7	RELU(BN(FC(out(6), 1024))	N × 1024
8	MP(out(7), N, 1)	1024
9	RELU(BN(FC(out(8), 1024))	1024
10	RELU(BN(FC(out(9), 1024))	1024
11	Tanh(FC(out(10), 1200))	1200
12	**Control Points** = Reshape(out(11), (20, 20, 3))	20 × 20 × 3

Table B.2: **Architecture of** SPLINENET**.** EdgeConv: edge convolution layer, BN: batch noramlization, RELU: rectified linear unit, FC: fully connected layer, CAT: concatenate tensors along second dimension, and MP: max-pooling across first dimension

$$p_{cover} = \frac{1}{|P|} \sum_{i \in P} \min_{j \in \hat{P}} \|i - j\|_2 \,,$$

$$s_{cover} = \frac{1}{|\hat{P}|} \sum_{i \in \hat{P}} \min_{j \in P} \|i - j\|_2 \,,$$

$$CD = \frac{1}{2}(p_{cover} + s_{cover}).$$

Here $|P|$ and $|\hat{P}|$ denote the cardinality of P and $\hat{P}$ respectively. We randomly sample $10k$ points each on the predicted and ground truth surface for the evaluation of all methods. Each predicted surface patch is also trimmed to define its boundary using bit-mapping with epsilon 0.1 [193]. To evaluate this metric, we use all predicted surface patches instead of the *matched* surface patches that is used in Section 5.3.

Results are shown in Table B.3. Evaluation using Chamfer distance follows the same trend of residual error shown in Table 1. PARSENET and SPFN with points as input performs better than NN and RANSAC. PARSENET and SPFN with points along with normals as input performs better than with just points as input. By training PARSENET end-to-end and also using post-process optimization results in the best performance. Our full PARSENET

Method	Input	p cover (1×10^{-4})	s cover (1×10^{-4})	CD (1×10^{-4})
NN	p	10.10	12.30	11.20
RANSAC	p+n	7.87	17.90	12.90
SPFN	p	7.17	13.40	10.30
SPFN	p+n	6.98	13.30	10.12
PARSENET	p	6.07	12.40	9.26
PARSENET	p+n	4.77	11.60	8.20
PARSENET + e2e + opt	p+n	**2.45**	**10.60**	**6.51**

Table B.3: **Reconstruction error measured using Chamfer distance on** ABCPARTS-DATASET**.** 'e2e': end-to-end training of PARSENET and 'opt': post-process optimization applied to B-spline surface patches.

gives 35.67% and 49.53% reduction in relative error in comparison to SPFN and RANSAC respectively. We show more visualizations of surfaces reconstructed by PARSENET in Figure B.3.

B.6 Evaluation on TraceParts Dataset

Here we evaluate the performance of PARSENET on the TraceParts dataset, and compare it with SPFN. Note that the input points are normalized to lie inside a unit cube. Points sampled from the shapes in TraceParts [139] have a fraction of points not assigned to any cluster. To make this dataset compatible with our evaluation approach, each unassigned point is merged to its closest cluster. This results in evaluation score to differ from the reported score in their paper [139].

First we create a nearest neighbor (NN) baseline as shown in the Section 5.3. In this, we first scale both training and testing shape an-isotropically such that each dimension has unit length. Then for each test shape, we find its most similar shape from the training set using Chamfer Distance. Then for each point on the test shape, we transfer the labels and primitive type from its closest point in $\mathcal{R}^3$ on the retrieved shape. We train PARSENET on the training set of TraceParts using the losses proposed in the Section 4.2 and we also train SPFN using their proposed losses. All results are reported on the test set of TraceParts.

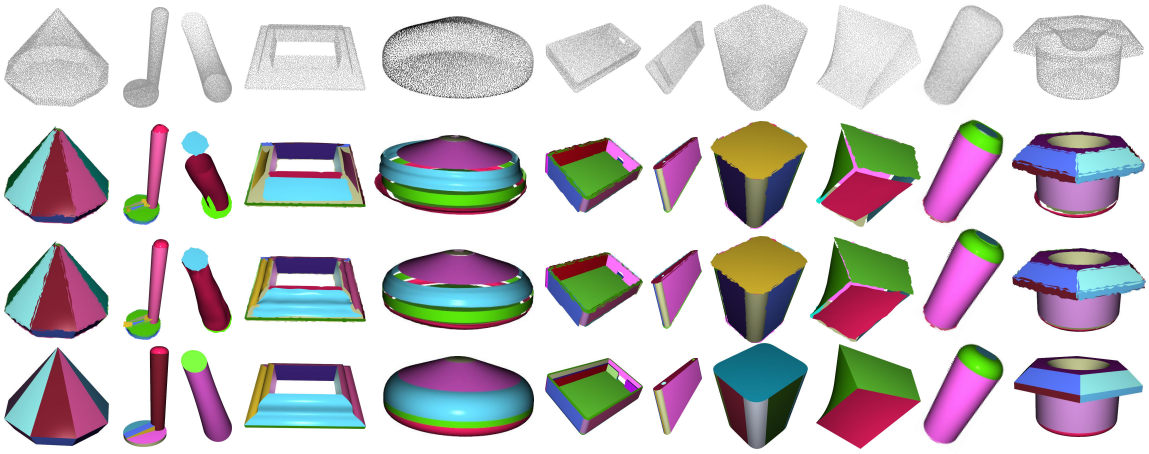


Figure B.3: Given the input point clouds with normals in the first row, we show surfaces produced by PARSENET without post-processing optimization (second row), and full PARSENET including optimization (third row). The last row shows the ground-truth surfaces from our ABCPARTSDATASET.

Results are shown in the Table B.4. The NN approach achieves a high segmentation mIOU of 81.92% and primitive type mIOU of 95%. Figure B.4 shows the NN results for a random set of shapes in the test set. It seems that the test and training sets often contain duplicate or near-duplicate shapes in the TraceParts dataset. Thus the performance of the NN can be attributed to the lack of shape diversity in this dataset. In comparison, our dataset is diverse, both in terms of shape variety and primitive types, and the NN baseline achieve much lower performance with segmentation mIOU of 54.10% and primitive type mIOU of 61.10%.

We further compare our PARSENET with SPFN with just points as input. PARSENET achieves 79.91% seg mIOU compared to 76.4% in SPFN. PARSENET achieves 97.39% label mIOU compared to 95.18% in SPFN. We also perform better when both points and normals are used as input to PARSENET and SPFN.

Finally, we compare reconstruction performance in the Table B.5. With just points as input to the network, PARSENET reduces the relative residual error by 9.35% with respect to SPFN. With both points and normals as input PARSENET reduces relative residual error by 15.17% with respect to SPFN.

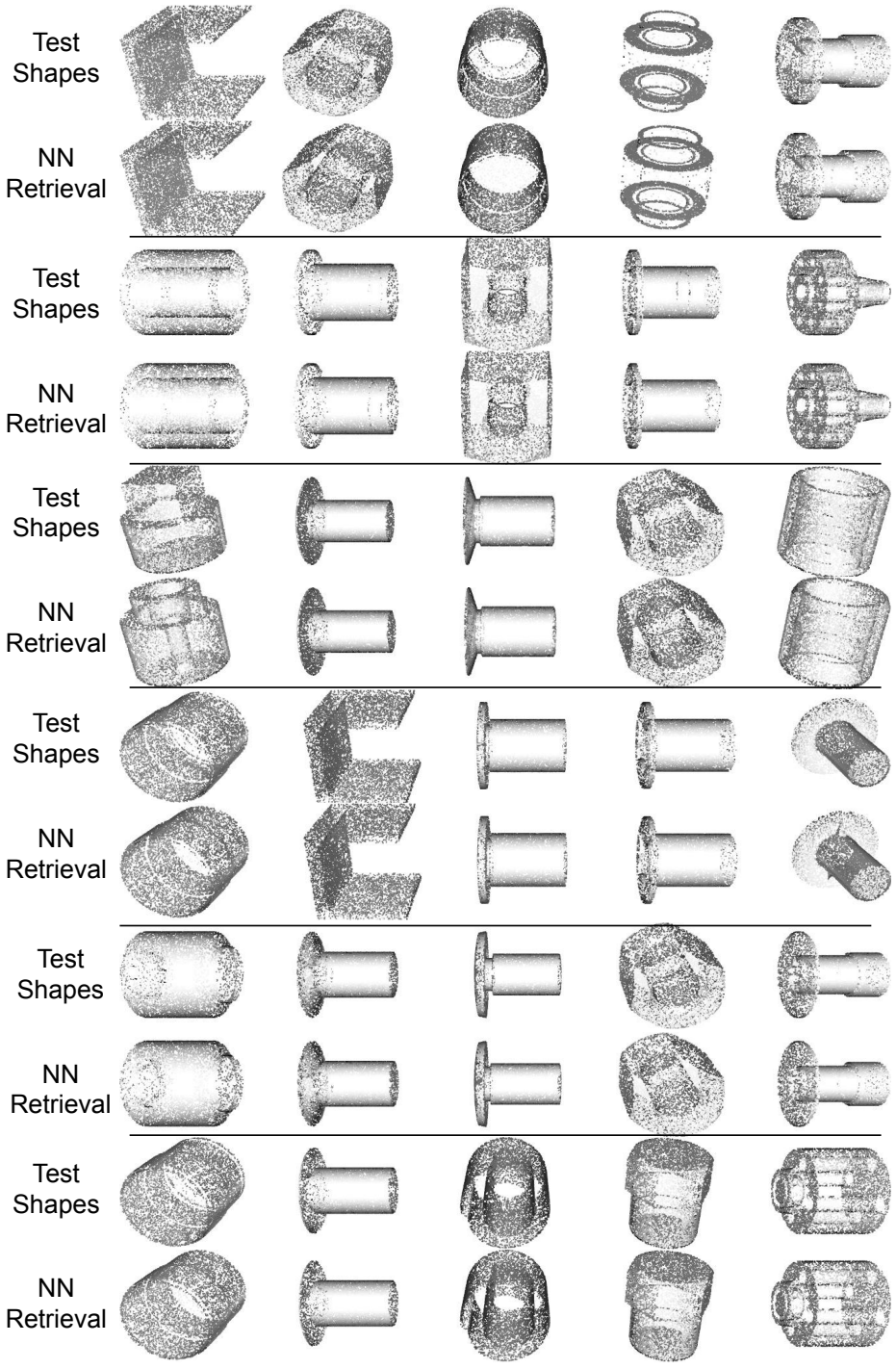


Figure B.4: **Nearest neighbor retrieval on the TracePart dataset** We randomly select 30 shapes from the test set of TraceParts dataset and show the NN retrieval, which reveals high training and testing set overlap. Shapes are an-isotropically scaled to unit length in each dimension. This is further validated quantitatively in Table B.4.

Method	Input	seg mIOU	label mIOU
NN	p	81.92	95.00
SPFN	p	76.4	95.18
SPFN	p + n	88.05	98.10
ParseNet	p	79.91	97.39
ParseNet	p + n	**88.57**	**98.26**

Table B.4: **Segmentation results on the TraceParts dataset.** We report segmentation and primitive type prediction performance of various methods.

Method	Input	res	P cover
NN	p	0.0138	91.90
SPFN	p	0.0139	91.70
SPFN	p + n	0.0112	**92.94**
ParseNet	p	0.0126	90.90
ParseNet	p + n	0**.0095**	92.72

Table B.5: **Reconstruction results on the TraceParts dataset.** We report residual loss and P cover metrics for various methods.

APPENDIX C

LEARNING POINT EMBEDDING FROM SHAPE REPOSITORIES FOR FEW SHOT SEMANTIC SEGMENTATION

C.1 Dataset

Our dataset is a subset of ShapeNetCore where we focus on 16 categories from ShapeNet part segmentation dataset. Note that the semantic segmentation dataset contains $16.6k$ shapes of these categories compared to $28k$ in the ShapeNet core. We first start by downloading collada file for shapes in ShapenetCore dataset from the 3D Warehouse website, but constraining to 16 categories mentioned above. Samples from the dataset are shown in the Figure C.2 (left). The Collada format stores the meshes in hierarchical structure, starting from the root node, recursively applying transformation until the leaf nodes that correspond to different parts of the 3D shape.

Note that we only use a small number of the segmentation labels provided in the ShapeNet segmentation benchmark for training in our few-shot segmentation experiments. We also make sure that the there is no overlap between any of our training set (embedding training, tag training, semantic segmentation training) and the evaluation set.

C.1.1 Generating segments from meshes.

The number of segments in meshes from Collada files can vary from 1-4000. These range from ones where all the parts are grouped together to others where parts are vastly over segmented. A possible way to control the number of segments is to select the depth of the tree that gives reasonable number of segments. Lower level in the hierarchy gives smaller number of segments as shown in Figure-3 (main paper). We select the depth of the tree such that the number of segments are at least k, where k is the number of semantic parts present

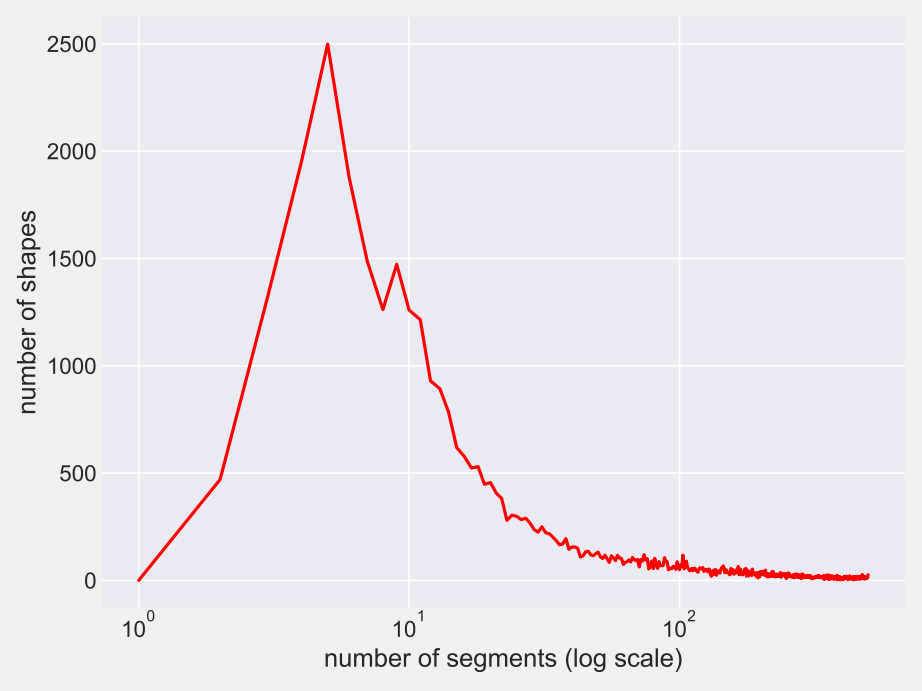


Figure C.1: **Distribution of number of segments.**

in the semantic part-segmentation dataset for that category. This is done to avoid favoring cases where semantically different parts are merged. We further, select the depth of the tree such that maximum number of segments is less than 500 to avoid large over-segmentation of shape and to keep high ratio of number of points vs number of segments. Figure C.1 shows the distribution of segments in our pruned dataset.

These meshes have inconsistent orientation, thus we preprocess these meshes to align in a canonical orientation of the Shapenent core dataset. The alignment is done by first sampling points from source and target meshes, then rotating the source point cloud along all the three-axis by from 0 to 180 degrees at the interval of 30 degrees and finally by selecting the orientation which gives least Chamfer distance between the source and the target shape points. The coarse search is sufficient to align most models. We preprocess the meshes by uniformly sampling 10k points from the surface using stratified sampling where sampling is weighted by the area of the segment, i.e. we sample more points from the segments with larger surface area in comparison to segments with smaller surface area.

Figure C.2: **Visualization of the meta data.** (Left) Parts of various objects shown in different colors. Notice that segmentations vary in their number and granularity across instances. (Right) A word cloud of the raw tags collected from the dataset. The font size is proportional to the square root of frequency in the dataset.

C.2 Visualization of Semantic Segmentation

Figure C.3 compares the segmentation models pretrained with Hierarchy meta data, trained from scratch, and autoencoder pretrained for training size 4 and 8.

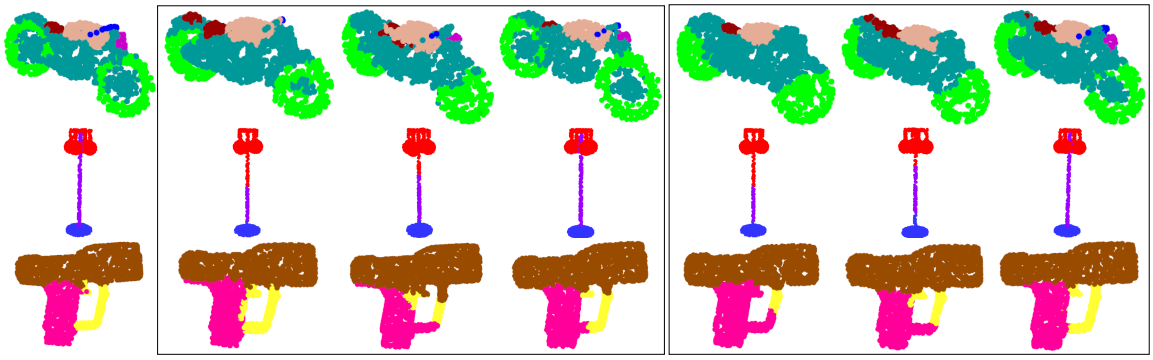


Figure C.3: **Segmentation results.** Visualization of segmentations produced by various models (scratch, autoencoder, hierarchy) when the number of training shapes is 4 (Left) and 8 (Right). The boundaries between parts are better delinated (as seen in the ground truth) by the models trained on hierarchy meta data.

APPENDIX D

SURFIT: LEARNING TO FIT SURFACES IMPROVES FEW SHOT LEARNING ON POINT CLOUDS

D.0.0.0.1 Mean-shift clustering details. (1) The mean-shift clustering adapts to the complexity of the shape by allowing different number of clusters based on a *bandwidth parameter*. This is computed for each shape by using the average distance of each point to its 100^{th} neighbor in the embedding space [147]. (2) We use *non-max suppression* to extract cluster centers. We start by extracting high density points that include at least one nearest neighbor within radius b. Then we remove all points that are within the same radius and repeat until no other high density points are left. All selected high density points in this way act as cluster centers.

D.0.0.0.2 Signed distance field approximation. We use approximate signed distance for ellipsoids:

$$S(p, s) = k_1(k_1 - 1)/k_2 \tag{D.1}$$

where s is an ellipsoid, p is the centered and re-oriented (to standard axis) coordinate of the point at which signed distance is calculated, $k_1 = \sqrt{\sum_i (\frac{p_i}{s_i})^2}$ and $k_2 = \sqrt{\sum_i (\frac{p_i}{s_i^2})^2}$, s_i is the length of the ellipsoid semi-axis in i^{th} direction.

D.0.0.0.3 Point sampling of ellipsoids. The parametric equation of an ellipsoid is the following:

$$\big(x, y, z\big) = \big(a\cos u \sin v, b \sin u \sin v, c \cos v\big), \tag{D.2}$$

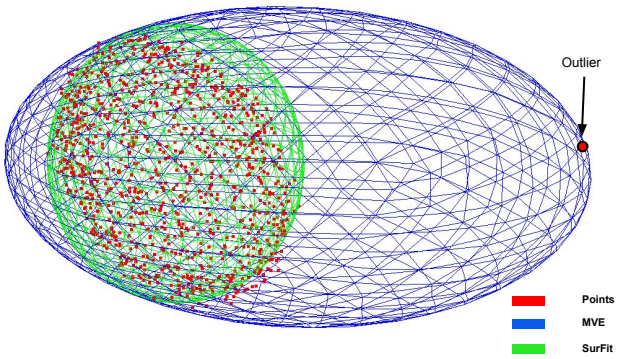


Figure D.1: **Robustness to outliers.** An example of outlier-robust fitting with our method in contrast to MVE (minimum volume ellipsoid) that is sensitive to outliers. Our fitting result shown in green closely fits the input points (red) while ignoring the outlier, whereas MVE approach (blue) is sensitive to outlier.

and its inverse parameterization is:

$$\big(v, u\big) = \big(\arccos(z/c), \mathrm{atan2}(y/b, x/a)\big), \tag{D.3}$$

where (u, v) are the parameters of the ellipsoid's 2D parametric domain, (x, y, z) the point coordinates, and (a, b, c) the lengths of semi-axis of the ellipsoid.

To sample the ellipsoid in a near-uniform manner, we start by creating a standard axis-aligned and origin centered mesh using the principal axis lengths predicted by SURFIT. Then we apply Poisson surface sampling to gather points on the surface in an approximately uniform manner. We then compute parameters (u, v) of the sampled points using Eq. D.3. Note that this is done outside the computation graph. We inject the computed parameters back to the computation graph using Eq. D.2 to compute point coordinates (x, y, z) again. These point coordinates are rotated and shifted based on the predicted axis and center respectively.

D.0.0.0.4 Robustness of ellipsoid fitting. Fig. D.1 shows the robustness of our approach to outliers in comparison to minimum volume ellipsoid (MVE) [56]. Our approach takes into account the membership of the point to a cluster. For this example, we use a simple

membership function $1/r^{\frac{1}{4}}$, where r is the distance of point from the center of the cluster and incorporates these per-point weights to estimate the parameters of ellipsoid in a closed form using SVD. not solve the MVE problem, rather provides an approximately fitting ellipsoid to a cluster of points.

APPENDIX E

MVDECOR: MULTI-VIEW DENSE CORRESPONDENCE LEARNING FOR FINE- GRAINED 3D SEGMENTATION

E.1 Supplementary Material

Here we further provide the following supplementary information and results:

- Training details of our approach and baselines
- RenderPeople annotation details
- Qualitative visualization on the RenderPeople dataset
- Experiment on the Shapenet dataset
- License of the datasets
- Discussion on societal impact and use of human dataset.

E.1.1 Training details

E.1.1.0.1 Training details for PartNet dataset. For pre-training and fine-tuning stages of our method we use the Adam optimizer with 0.001 learning rate. For pre-training we decay the learning rate by half when validation loss saturates. During pre-training, we use $4k$ matched pairs of points for a pair of views to compute our self supervision loss. During pre-training on the PartNet dataset, we train our model with batch size of 16 for $200k$ iterations. For fine-tuning, we use the batch size of 8 and exponential learning rate decay (factor=0.99) after every 40 iterations. For $k = 10$, $v = all$ setting, we train our model for $4k$ iterations, and for $k = 10$ and $v = 5$ setting, we train our model for $2k$ iterations.

E.1.1.0.2 Training details for RenderPeople dataset. For pre-training and fine-tuning stages of our method, we use the Adam optimizer with 0.001 learning rate. For pre-training, we decay the learning rate by half when validation loss saturates. During pre-training, we use $4k$ matched pairs of points for a pair of views to compute our self supervision loss. During pre-training for the RenderPeople dataset, we train our model with batch size of 16 for $100k$ iterations. For fine-tuning, we use the batch size of 8 and exponential learning rate decay (factor=0.99) after every 40 iterations. For the RenderPeople dataset for $k = 5$, $v = all$ setting we train our model for $2K$ iterations, and for $k = 5$ and $v = 3$ setting, we train our model for 400 iterations.

E.1.1.0.3 DeepLabv3+. We use the DeepLabV3+ as our 2D CNN backbone for learning pixel level features. We modify the last layer of DeepLabV3+. In the original version, the (64×64) feature map is directly $4\times$ upsampled to a (256×256) feature map using bilinear interpolation, since the input image has a size of $(256 \times 256 \times 3)$. We instead gradually upsample the (64×64) feature map to (256×256) resolution in two upsampling stages to preserve fine-grained details in the following way: $\texttt{Up}(2) \rightarrow \texttt{BN}(256) \rightarrow \texttt{Relu} \rightarrow \texttt{Conv2D}(256, 128, 3) \rightarrow \texttt{Up}(2) \rightarrow \texttt{BN}(256) \rightarrow \texttt{Relu} \rightarrow \texttt{Conv2D}(128, 64, 3)$. We also use bilinear up-sampling. $\texttt{Conv2D}(i, o, k)$ is a 2D convolution layer with i input channels, o output channels and k kernel size, $\texttt{Relu}$ is rectified linear unit, $\texttt{Up}(x)$ is bilinear up-sampling by a factor of x and $\texttt{BN}$ is a batch normalization layer.

E.1.1.0.4 DenseCL. We keep all the hyper parameters same as proposed in the original work. When depth map and normal maps are also input to the network, the spatial augmentations applied to the RGB image are also applied to the normal and depth maps. We do not augment normal and depth maps in any other way. The models are trained until convergence. Once the DenseCL baseline is pre-trained using their proposed approach on our dataset, we use the backbone ResNet weights to initialize our DeepLabv3+ architecture as described above and add a 2D convolution layer as a segmentation head.

E.1.1.0.5 PointContrast. To implement our 3D baseline, we use a 3D ResNet with U-Net based architecture with 42 layers as proposed in the original paper [189]. We use a voxel size of 0.01. We use a batch size of 16 and $10k$ pairs of matched points to compute their self supervision loss. The implementation of the loss is done using the source code provided by the authors. We use the SGD optimizer with learning rate 0.1 with 0.9 momentum and 0.0001 weight decay. We train this model for $100k$ iterations. The validation loss saturates after $100k$ iterations.

E.1.2 RenderPeople dataset

We label Renderpeople shapes using the labeling tool from [230]. We start by rendering multiple RGB images of the textured mesh such that maximum surface area can be covered. Then we label each rendered image and back-project the pixel labels to the surface. We label 13 different parts as shown in Figure E.1.

In Figure E.2, we provide additional qualitative results on the RenderPeople dataset.

E.1.3 Experiment on Shapenet dataset

The main focus of our work is fine-grained semantic segmentation. We also experiment with the Shapenet Semantic Segmentation dataset [28] for the task of few-shot semantic segmentation, which consists of $16,881$ labeled point clouds across 16 shape categories, with a total of 50 part categories. We transfer the point labels to triangles of a mesh using nearest neighbor queries to train our models. The evaluation is done by transferring the predicted triangle labels back to original point cloud. We use the same architecture and training strategy for this dataset as used for other datasets. We report our results in Table E.1. Note that instance mIOU is highly influenced by the shape categories with large number of testing shapes *e.g.* Chair, Table. Class mIOU, on the other hand gives equal importance to all categories, hence it is a more robust evaluation metric. We evaluate the performance of work by Wang *et al.* [232] all all shape categories from this dataset and average the performance over 5 random runs.

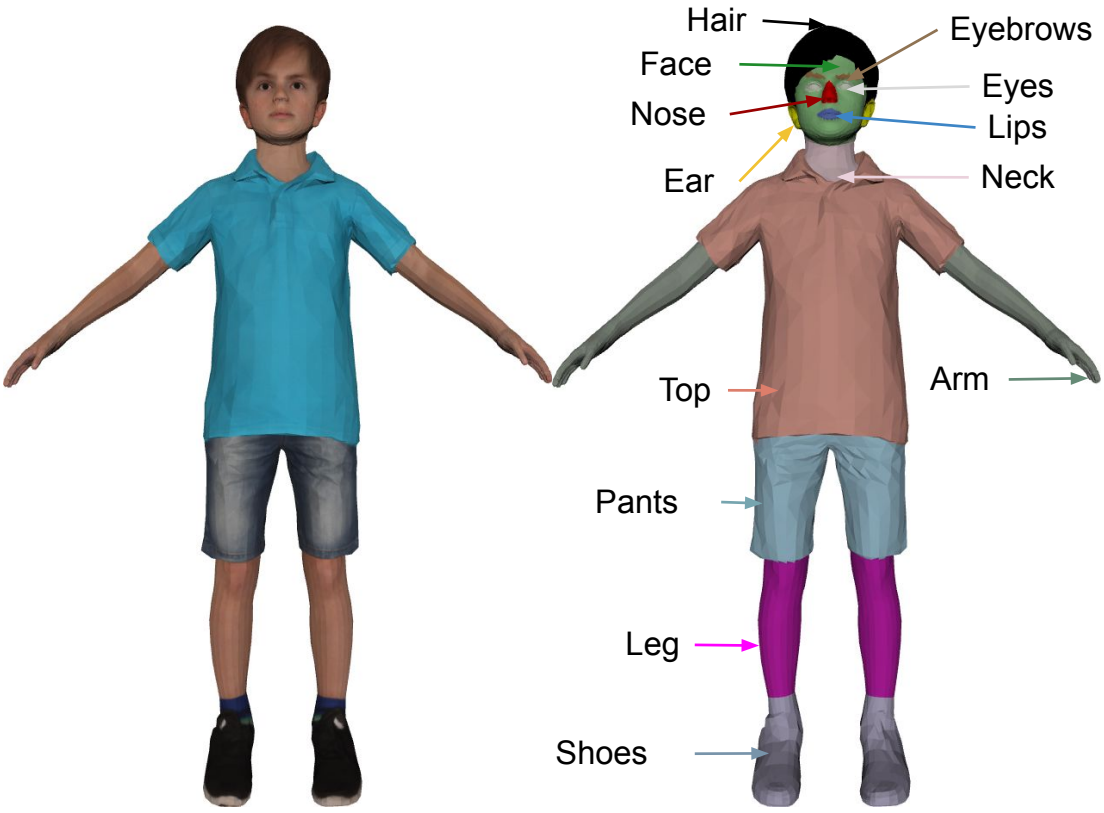


Figure E.1: Semantic labels of a shape from the RenderPeople dataset.

Figure E.2: **Visualization of predicted semantic labels on Renderpeople dataset in the few-shot setting when $k = 5$ fully labeled shapes are used for fine-tuning.** We visualize the predictions of all baselines. To visualize the details of predicted segmentations in the facial region, we provide an inset figure.

Methods	instance avg. mIOU	class avg. mIOU
SO-Net [136]	64.0	-
PointCapsNet [265]	67.0	-
MortonNet [222]	-	-
JointSSL [11]	71.9	-
Multi-task [87]	68.2	-
Deformation [232]	68.9	66.2
PointContrast [189]	74.0	-
ACD [64]	**75.7**	74.1
2D Scratch	72.7	74.7
(2D+3D) Ours	74.3	**75.8**

Table E.1: **Comparison with state-of-the-art few-shot part segmentation methods on ShapeNet.** Performance is evaluated using *instance-averaged* and *class-averaged* mIOU while using 1% of the training data.

E.1.4 Dataset licenses

The PartNet dataset [159] is a collection of labeled shapes from ShapeNet [28]. The license can be found on the website of ShapeNet. We obtained the license for using the Renderpeople dataset [5] through an agreement with Renderpeople. We carefully inspected the datasets we use and did not find identifiable information or offensive content.

To run the comparison with baseline methods, we use the source code provided by the authors of DenseCL [1], PointContrast [3].

E.1.5 Ethics discussions

E.1.5.1 Potential negative societal impacts

We present a method for labeling detailed parts of 3D models given a provided training set of shapes. Like many other learning-based methods, our results can be biased by training datasets. For purpose of deploying the method for human shapes, one would need to carefully inspect and de-bias the dataset to depict the target distribution of a wide range of possible body shapes, clothing, skin tones, or at the intersection of race and gender.

E.1.5.2 Personal data and human subjects

Our paper uses human 3D models from Renderpeople for training and evaluation. The data collection and ethics approvals were taken care of by the dataset provider. We carefully inspected the dataset and did not find identifiable information or offensive content. More information about the dataset can be found on the websites of data provider.

BIBLIOGRAPHY

[1] Densecl. https://github.com/WXinlong/DenseCL.

[2] Ellipsoid SDF. `https://www.iquilezles.org/www/articles/ellipsoids/ellipsoids.htm`. Accessed: 2020-11-15.

[3] Pointcontrast. https://github.com/facebookresearch/PointContrast.

[4] Pytorch. `https://pytorch.org`.

[5] Renderpeople. https://renderpeople.com/.

[6] Trimble 3D Warehouse. `https://3dwarehouse.sketchup.com/`.

[7] Achlioptas, Panos, Diamanti, Olga, Mitliagkas, Ioannis, and Guibas, Leonidas J. Learning Representations and Generative Models For 3D Point Clouds. In *International Conference on Machine Learning* (2018).

[8] Ahmed, Eman, Saint, Alexandre, Shabayek, Abd El Rahman, Cherenkova, Kseniya, Das, Rig, Gusev, Gleb, Aouada, Djamila, and Ottersten, Björn E. Deep learning advances on different 3D data representations: A survey. *Computing Research Repository (CoRR) abs/1808.01462* (2019).

[9] Ahmed Asif Fuad, Kazi, Martin, Pierre-Etienne, Giot, Romain, Bourqui, Romain, Benois-Pineau, Jenny, and Zemmari, Akka. Features Understanding in 3D CNNs for Actions Recognition in Video. In *Tenth International Conference on Image Processing Theory, Tools and Applications, IPTA 2020* (Paris, France, Oct. 2020).

[10] Allen, Brett, Curless, Brian, Curless, Brian, and Popović, Zoran. The space of human body shapes: Reconstruction and parameterization from range scans. *ACM Trans. Graph. 22*, 3 (2003).

[11] Alliegro, Antonio, Boscaini, Davide, and Tommasi, Tatiana. Joint supervised and self-supervised learning for 3d real-world challenges, 2020.

[12] Andreas, Jacob, Rohrbach, Marcus, Darrell, Trevor, and Klein, Dan. Neural Module Networks. In *Proc. CVPR* (2016).

[13] Anguelov, Dragomir, Srinivasan, Praveen, Koller, Daphne, Thrun, Sebastian, Rodgers, Jim, and Davis, James. Scape: Shape completion and animation of people. *ACM Trans. Graph. 24*, 3 (2005).

[14] Armeni, I., Sener, O., Zamir, A. R., Jiang, H., Brilakis, I., Fischer, M., and Savarese, S. 3d semantic parsing of large-scale indoor spaces. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 1534–1543.

[15] Balog, Matej, Gaunt, Alexander L, Brockschmidt, Marc, Nowozin, Sebastian, and Tarlow, Daniel. DeepCoder: Learning to Write Programs. In *Proc. ICLR* (2017).

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

[17] Ben-Shabat, Yizhak, and Gould, Stephen. Deepfit: 3d surface fitting via neural network weighted least squares. *arXiv preprint arXiv:2003.10826* (2020).

[18] Biederman, Irving. Recognition-by-Components: A Theory of Human Image Understanding. *Psychological Review 94*, 2 (1987).

[19] Binford, I. Visual perception by computer. In *IEEE Conference of Systems and Control* (1971).

[20] Binford, Thomas O, Levitt, Tod S, and Mann, Wallace B. Bayesian inference in model-based machine vision. In *Proceedings of the Third Conference on Uncertainty in Artificial Intelligence* (1987), pp. 86–97.

[21] Bogo, Federica, Romero, Javier, Loper, Matthew, and Black, Michael J. FAUST: Dataset and evaluation for 3D mesh registration. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (Piscataway, NJ, USA, June 2014), IEEE.

[22] Bourdev, Lubomir, Maji, Subhransu, Brox, Thomas, and Malik, Jitendra. Detecting people using mutually consistent poselet activations. In *Proc. ECCV* (2010).

[23] Bronstein, Alexander, Bronstein, Michael, Guibas, Leonidas, and Ovsjanikov, Maks. Shape google: Geometric words and expressions for invariant shape retrieval. *ACM Trans. Graph. 30* (01 2011), 1.

[24] Buchele, Suzanne F., and Crawford, Richard H. Three-dimensional halfspace constructive solid geometry tree construction from implicit boundary representations. In *Proceedings of the Eighth ACM Symposium on Solid Modeling and Applications* (2003).

[25] Carr, Jonathan C, Beatson, Richard K, Cherrie, Jon B, Mitchell, Tim J, Fright, W Richard, McCallum, Bruce C, and Evans, Tim R. Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), pp. 67–76.

[26] Cazals, F., and Pouget, M. Estimating differential quantities using polynomial fitting of osculating jets. *Comput. Aided Geom. Des. 22*, 2 (feb 2005), 121–146.

[27] Chang, Angel X., Funkhouser, Thomas A., Guibas, Leonidas J., Hanrahan, Pat, Huang, Qi-Xing, Li, Zimo, Savarese, Silvio, Savva, Manolis, Song, Shuran, Su, Hao, Xiao, Jianxiong, Yi, Li, and Yu, Fisher. Shapenet: An information-rich 3d model repository. *CoRR abs/1512.03012* (2015).

[28] Chang, Angel X., Funkhouser, Thomas A., Guibas, Leonidas J., Hanrahan, Pat, Huang, Qi-Xing, Li, Zimo, Savarese, Silvio, Savva, Manolis, Song, Shuran, Su, Hao, Xiao, Jianxiong, Yi, Li, and Yu, Fisher. Shapenet: An information-rich 3d model repository. *CoRR abs/1512.03012* (2015).

[29] Chatfield, K., Simonyan, K., Vedaldi, A., and Zisserman, A. Return of the devil in the details: Delving deep into convolutional nets. In *Proc. BMVC* (2014).

[30] Chaudhuri, Siddhartha, and Koltun, Vladlen. Data-driven suggestions for creativity support in 3d modeling. *ACM Trans. Graph. 29*, 6 (dec 2010).

[31] Chen, Liang-Chieh, Zhu, Yukun, Papandreou, George, Schroff, Florian, and Adam, Hartwig. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV* (2018).

[32] Chen, Ting, Kornblith, Simon, Norouzi, Mohammad, and Hinton, Geoffrey. A simple framework for contrastive learning of visual representations, 2020.

[33] Chen, Xiaobai, Golovinskiy, Aleksey, and Funkhouser, Thomas. A benchmark for 3d mesh segmentation. *ACM Trans. Graph. 28*, 3 (July 2009).

[34] Chen, Zhiqin, Tagliasacchi, Andrea, and Zhang, Hao. Bsp-net: Generating compact meshes via binary space partitioning. *CVPR* (2020).

[35] Chen, Zhiqin, Yin, Kangxue, Fisher, Matthew, Chaudhuri, Siddhartha, and Zhang, Hao. BAE-NET: branched autoencoder for shape co-segmentation. *CoRR abs/1903.11228* (2019).

[36] Chen, Zhiqin, Yin, Kangxue, Fisher, Matthew, Chaudhuri, Siddhartha, and Zhang, Hao. BAE-NET: branched autoencoder for shape co-segmentation. In *Proceedings of the IEEE International Conference on Computer Vision* (2019), pp. 8490–8499.

[37] Cheng, Jingchun, Tsai, Yi-Hsuan, Hung, Wei-Chih, Wang, Shengjin, and Yang, Ming-Hsuan. Fast and accurate online video object segmentation via tracking parts. *CoRR abs/1806.02323* (2018).

[38] Choy, Christopher, Gwak, JunYoung, and Savarese, Silvio. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *CVPR* (2019).

[39] Choy, Christopher B, Gwak, JunYoung, Savarese, Silvio, and Chandraker, Manmohan. Universal correspondence network. In *Advances in Neural Information Processing Systems 30*. 2016.

[40] Chung, Junyoung, Gulcehre, Caglar, Cho, KyungHyun, and Bengio, Yoshua. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[41] Cuturi, Marco, Teboul, Olivier, and Vert, Jean-Philippe. Differentiable ranking and sorting using optimal transport. In *Advances in Neural Information Processing Systems 32* (2019).

[42] Dai, A., Chang, A. X., Savva, M., Halber, M., Funkhouser, T., and Nießner, M. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 2432–2443.

[43] Deng, B., Genova, K., Yazdani, S., Bouaziz, S., Hinton, G., and Tagliasacchi, A. Cvxnet: Learnable convex decomposition. In *CVPR* (2020), pp. 31–41.

[44] Deng, Boyang, Genova, Kyle, Yazdani, Soroosh, Bouaziz, Sofien, Hinton, Geoffrey, and Tagliasacchi, Andrea. Cvxnet: Learnable convex decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2020).

[45] Deng, J., Dong, W., Socher, R., Li, L., Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition* (2009), pp. 248–255.

[46] Denil, Misha, Gómez Colmenarejo, Sergio, Cabi, Serkan, Saxton, David, and De Freitas, Nando. Programmable Agents. *arXiv preprint arXiv:1706.06383* (2017).

[47] Deprelle, Theo, Groueix, Thibault, Fisher, Matthew, Kim, Vladimir, Russell, Bryan, and Aubry, Mathieu. Learning elementary structures for 3d shape generation and matching. In *Proc. NeurIPS* (2019), vol. 32.

[48] Dijkstra, E. W. Recursive programming. *Numer. Math. 2*, 1 (1960).

[49] Doersch, Carl, Gupta, Abhinav, and Efros, Alexei A. Unsupervised visual representation learning by context prediction. *CoRR abs/1505.05192* (2015).

[50] Du, Tao, Inala, Jeevana Priya, Pu, Yewen, Spielberg, Andrew, Schulz, Adriana, Rus, Daniela, Solar-Lezama, Armando, and Matusik, Wojciech. Inversecsg: Automatic conversion of 3d models to csg trees. *ACM Trans. Graph. 37*, 6 (Dec. 2018).

[51] Duda, Richard O., and Hart, Peter E. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM 15*, 1 (Jan. 1972), 11–15.

[52] Eck, Matthias, and Hoppe, Hugues. Automatic reconstruction of B-spline surfaces of arbitrary topological type. In *SIGGRAPH* (1996).

[53] Ellis, Kevin, Ritchie, Daniel, Solar-Lezama, Armando, and Tenenbaum, Joshua B. Learning to Infer Graphics Programs from Hand-Drawn Images. *arXiv preprint arXiv:1707.09627* (2017).

[54] Eslami, S. M. Ali, Heess, Nicolas, Weber, Theophane, Tassa, Yuval, Szepesvari, David, Kavukcuoglu, Koray, and Hinton, Geoffrey. Attend, Infer, Repeat: Fast Scene Understanding with Generative Models. In *Proc. NIPS* (2016).

[55] et al., Kundu. Virtual multi-view fusion for 3d semantic segmentation. In *Proc. ECCV* (2020).

[56] Fang, Shu-Cherng, and Puthenpura, Sarat. *Linear Optimization and Extensions: Theory and Algorithms*. Prentice-Hall, Inc., USA, 1993.

[57] Farin, Gerald. *Curves and Surfaces for CAGD*, 5th ed. Morgan Kaufmann, 2002.

[58] Fayolle, Pierre-Alain, and Pasko, Alexander. An evolutionary approach to the extraction of object construction trees from 3d point clouds. *Computer-Aided Design 74* (2016), 1 – 17.

[59] Felzenszwalb, Pedro F, and Huttenlocher, Daniel P. Pictorial structures for object recognition. *IJCV 61*, 1 (2005), 55–79.

[60] Fischler, Martin A, and Elschlager, Robert A. The representation and matching of pictorial structures. *IEEE Transactions on computers 100*, 1 (1973), 67–92.

[61] Foley, James D., van Dam, Andries, Feiner, Steven K., and Hughes, John F. *Computer Graphics: Principles and Practice*, 2nd ed. Addison-Wesley Longman Publishing Co., Inc., USA, 1990.

[62] Fowlkes, E. B., and Mallows, C. L. A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association 78*, 383 (1983), 553–569.

[63] Gadelha, Matheus, Gori, Giorgio, Ceylan, Duygu, Mech, Radomir, Carr, Nathan, Boubekeur, Tamy, Wang, Rui, and Maji, Subhransu. Learning generative models of shape handles. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2020).

[64] Gadelha, Matheus, RoyChowdhury, Aruni, Sharma, Gopal, Kalogerakis, Evangelos, Cao, Liangliang, Learned-Miller, Erik, Wang, Rui, and Maji, Subhransu. Label-efficient learning on point clouds using approximate convex decompositions. In *European Conference on Computer Vision (ECCV)* (2020).

[65] Gadelha, Matheus, Wang, Rui, and Maji, Subhransu. Multiresolution Tree Networks for 3D Point Cloud Processing. In *ECCV* (2018).

[66] Gal, Ran, Sorkine, Olga, Mitra, Niloy J., and Cohen-Or, Daniel. iwires: An analyze-and-edit approach to shape manipulation. *ACM Transactions on Graphics (Siggraph) 28*, 3 (2009), #33, 1–10.

[67] Gao, Jun, Tang, Chengcheng, Ganapathi-Subramanian, Vignesh, Huang, Jiahui, Su, Hao, and Guibas, Leonidas J. Deepspline: Data-driven reconstruction of parametric curves and surfaces. *CoRR abs/1901.03781* (2019).

[68] Gao, Jun, Tang, Chengcheng, Ganapathi-Subramanian, Vignesh, Huang, Jiahui, Su, Hao, and Guibas, Leonidas J. DeepSpline: Data-driven reconstruction of parametric curves and surfaces. *Computing Research Repository (CoRR) abs/1901.03781* (2019).

[69] Genova, Kyle, Cole, Forrester, Sud, Avneesh, Sarna, Aaron, and Funkhouser, Thomas. Local deep implicit functions for 3d shape. In *CVPR* (June 2020).

[70] Genova, Kyle, Cole, Forrester, Vlasic, Daniel, Sarna, Aaron, Freeman, William T, and Funkhouser, Thomas. Learning shape templates with structured implicit functions. In *International Conference on Computer Vision* (2019).

[71] Gershman, Sam, and Goodman, Noah D. Amortized inference in probabilistic reasoning. In *Proceedings of the Thirty-Sixth Annual Conference of the Cognitive Science Society* (2014).

[72] Gidaris, Spyros, Singh, Praveer, and Komodakis, Nikos. Unsupervised representation learning by predicting image rotations. *CoRR abs/1803.07728* (2018).

[73] Ginzburg, Dvir, and Raviv, Dan. Cyclic functional mapping: Self-supervised correspondence between non-isometric deformable shapes, 2019.

[74] Gori, Giorgio, Sheffer, Alla, Vining, Nicholas, Rosales, Enrique, Carr, Nathan, and Ju, Tao. Flowrep: Descriptive curve networks for free-form design shapes. *ACM Transaction on Graphics 36*, 4 (2017).

[75] Goyal, Ankit, Law, Hei, Liu, Bowei, Newell, Alejandro, and Deng, Jia. Revisiting point cloud shape classification with a simple and effective baseline. *arXiv preprint arXiv:2106.05304* (2021).

[76] Grill, Jean-Bastien, Strub, Florian, Altché, Florent, Tallec, Corentin, Richemond, Pierre, Buchatskaya, Elena, Doersch, Carl, Avila Pires, Bernardo, Guo, Zhaohan, Gheshlaghi Azar, Mohammad, Piot, Bilal, kavukcuoglu, koray, Munos, Remi, and Valko, Michal. Bootstrap your own latent - a new approach to self-supervised learning. In *Advances in Neural Information Processing Systems* (2020), H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., pp. 21271–21284.

[77] Grill, Jean-Bastien, Strub, Florian, Altché, Florent, Tallec, Corentin, Richemond, Pierre H., Buchatskaya, Elena, Doersch, Carl, Pires, Bernardo Avila, Guo, Zhaohan Daniel, Azar, Mohammad Gheshlaghi, Piot, Bilal, Kavukcuoglu, Koray, Munos, Rémi, and Valko, Michal. Bootstrap your own latent: A new approach to self-supervised learning, 2020.

[78] Groueix, Thibault, Fisher, Matthew, Kim, Vladimir G., Russell, Bryan, and Aubry, Mathieu. AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (2018).

[79] Hackel, Timo, Wegner, Jan D., and Schindler, Konrad. Contour detection in unstructured 3d point clouds. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 1610–1618.

[80] Halimi, Oshri, Litany, Or, Rodolà, Emanuele, Bronstein, Alex, and Kimmel, Ron. Self-supervised learning of dense shape correspondence, 2018.

[81] Hallinan, Peter W., Gordon, Gaile G., Yuille, A. L., Giblin, Peter, and Mumford, David. *Two- and Three-Dimensional Patterns of the Face*. A. K. Peters, Ltd., USA, 1999.

[82] Hamdi, Abdullah, Giancola, Silvio, and Ghanem, Bernard. Mvtn: Multi-view transformation network for 3d shape recognition, 2021.

[83] Hamza, Karim, and Saitou, Kazuhiro. Optimization of constructive solid geometry via a tree-based multi-objective genetic algorithm. In *Genetic and Evolutionary Computation – GECCO 2004* (Berlin, Heidelberg, 2004), Kalyanmoy Deb, Ed., Springer Berlin Heidelberg, pp. 981–992.

[84] Hamza, Karim, and Saitou, Kazuhiro. Optimization of constructive solid geometry via a tree-based multi-objective genetic algorithm. In *Genetic and Evolutionary Computation* (2004).

[85] Hanocka, Rana, Hertz, A., Fish, N., Giryes, Raja, Fleishman, S., and Cohen-Or, D. Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG) 38* (2019), 1 – 12.

[86] Hanocka, Rana, Hertz, Amir, Fish, Noa, Giryes, Raja, Fleishman, Shachar, and Cohen-Or, Daniel. Meshcnn: A network with an edge. *ACM Transactions on Graphics (TOG) 38*, 4 (2019), 90.

[87] Hassani, Kaveh, and Haley, Mike. Unsupervised multi-task feature learning on point clouds. In *Proceedings of the IEEE International Conference on Computer Vision* (2019), pp. 8160–8171.

[88] He, Kaiming, Chen, Xinlei, Xie, Saining, Li, Yanghao, Dollár, Piotr, and Girshick, Ross B. Masked autoencoders are scalable vision learners. *CoRR abs/2111.06377* (2021).

[89] He, Kaiming, Fan, Haoqi, Wu, Yuxin, Xie, Saining, and Girshick, Ross. Momentum contrast for unsupervised visual representation learning. *arXiv preprint arXiv:1911.05722* (2019).

[90] He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *CVPR* (2016).

[91] Hermans, Alexander, Beyer, Lucas, and Leibe, Bastian. In defense of the triplet loss for person re-identification. *CoRR abs/1703.07737* (2017).

[92] Hinton, G. E., and Salakhutdinov, R. R. Reducing the dimensionality of data with neural networks. *Science 313*, 5786 (2006), 504–507.

[93] Hoffer, Elad, and Ailon, Nir. Deep metric learning using triplet network. *Computing Research Repository (CoRR) abs/1412.6622* (2014).

[94] Hoffman, Donald D, and Richards, Whitman. Parts of recognition.

[95] Hopcroft, John E, Motwani, Rajeev, and D, Ullman Jeffrey. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 2001.

[96] Hoppe, Hugues, DeRose, Tony, Duchamp, Tom, Halstead, Mark, Jin, Hubert, McDonald, John, Schweitzer, Jean, and Stuetzle, Werner. Piecewise smooth surface reconstruction. In *Proc. SIGGRAPH* (1994).

[97] Horn, B.K.P. Extended gaussian images. *Proceedings of the IEEE 72*, 12 (1984), 1671–1686.

[98] Hou, Ji, Graham, Benjamin, Nießner, Matthias, and Xie, Saining. Exploring data-efficient 3d scene understanding with contrastive scene contexts. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), pp. 15587–15597.

[99] Hou, Ji, Xie, Saining, Graham, Benjamin, Dai, Angela, and Nießner, Matthias. Pri3d: Can 3d priors help 2d representation learning? *arXiv preprint arXiv:2104.11225* (2021).

[100] Hu, Ronghang, Andreas, Jacob, Rohrbach, Marcus, Darrell, Trevor, and Saenko, Kate. Learning to reason: End-to-end module networks for visual question answering. In *Proc. ICCV* (2017).

[101] Huang, Haibin, Kalogerakis, Evangelos, Chaudhuri, Siddhartha, Ceylan, Duygu, Kim, Vladimir G., and Yumer, Ersin. Learning local shape descriptors from part correspondences with multiview convolutional networks. *ACM Transactions on Graphics 37*, 1 (2018).

[102] Huang, Haibin, Kalogerakis, Evangelos, and Marlin, Benjamin. Analysis and synthesis of 3d shape families via deep-learned generative models of surfaces. *Computer Graphics Forum 34*, 5 (2015).

[103] Huang, Haibin, Kalogerakis, Evangelos, Yumer, Ersin, and Mech, Radomir. Shape Synthesis from Sketches via Procedural Models and Convolutional Networks. *IEEE Trans. Vis. & Comp. Graphics 23*, 8 (2017).

[104] Huang, Hui, Wu, Shihao, Gong, Minglun, Cohen-Or, Daniel, Ascher, Uri, and Zhang, Hao (Richard). Edge-aware point set resampling. *ACM Trans. Graph. 32*, 1 (feb 2013).

[105] Huang, Jiahui, Gao, Jun, Ganapathi-Subramanian, Vignesh, Su, Hao, Liu, Yin, Tang, Chengcheng, and Guibas, Leonidas J. Deepprimitive: Image decomposition by layered primitive detection. *Computational Visual Media 4*, 4 (Dec 2018), 385–397.

[106] Huang, Jingwei, Zhang, Yanfeng, and Sun, Mingwei. Primitivenet: Primitive instance segmentation with local primitive embedding under adversarial metric. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (October 2021), pp. 15343–15353.

[107] Huang, Qi-Xing, Su, Hao, and Guibas, Leonidas. Fine-grained semi-supervised labeling of large shape collections. *ACM Trans. Graph. 32*, 6 (2013).

[108] Ionescu, C., Vantzos, O., and Sminchisescu, C. Matrix backpropagation for deep networks with structured layers. In *2015 IEEE International Conference on Computer Vision (ICCV)* (2015), pp. 2965–2973.

[109] Izadinia, Hamid, and Seitz, Steven M. Scene recomposition by learning-based icp. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2020).

[110] Jain, Arjun, Thormählen, Thorsten, Ritschel, Tobias, and Seidel, Hans-Peter. Exploring shape variations by 3d-model decomposition and part-based recombination. *Comput. Graph. Forum 31*, 2pt3 (May 2012), 631–640.

[111] Ji, Shuiwang, Xu, Wei, Yang, Ming, and Yu, Kai. 3d convolutional neural networks for human action recognition. vol. 35, IEEE Computer Society, pp. 221–231.

[112] Jiang, Chiyu, Sud, Avneesh, Makadia, Ameesh, Huang, Jingwei, Nießner, Matthias, Funkhouser, Thomas, et al. Local implicit grid representations for 3d scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 6001–6010.

[113] Johnson, Justin, Hariharan, Bharath, Van Der Maaten, Laurens, Hoffman, Judy, Fei-Fei, Li, Zitnick, C Lawrence, and Girshick, Ross. Inferring and Executing Programs for Visual Reasoning. In *Proc. ICCV* (2017).

[114] Joulin, Armand, and Mikolov, Tomas. Inferring Algorithmic Patterns with Stack-Augmented Recurrent Nets. In *Proc. NIPS* (2015).

[115] Kaiser, Adrien, Ybanez Zepeda, Jose Alonso, and Boubekeur, Tamy. A survey of simple geometric primitives detection methods for captured 3D data. *Computer Graphics Forum 38*, 1 (2019), 167–196.

[116] Kaiser, Lukasz, and Sutskever, Ilya. Neural GPUs Learn Algorithms. In *Proc. ICLR* (2016).

[117] Kalogerakis, Evangelos, Averkiou, Melinos, Maji, Subhransu, and Chaudhuri, Siddhartha. 3D shape segmentation with projective convolutional networks. In *Proc. CVPR* (2017).

[118] Kalogerakis, Evangelos, Nowrouzezahrai, Derek, Simari, Patricio, and Singh, Karan. Extracting lines of curvature from noisy point clouds. *Computer-Aided Design 41*, 4 (2009), 282–292. Point-based Computational Techniques.

[119] Kawana, Yuki, Mukuta, Yusuke, and Harada, Tatsuya. Neural star domain as primitive representation. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Red Hook, NY, USA, 2020), NIPS'20, Curran Associates Inc.

[120] Kingma, Diederik P., and Ba, Jimmy. Adam: A method for stochastic optimization. *CoRR abs/1412.6980* (2014).

[121] Klokov, Roman, and Lempitsky, Victor. Escape from cells: Deep Kd-Networks for the recognition of 3D point cloud models. In *Proc. ICCV* (2017).

[122] Knopp, Jan, Prasad, Mukta, Willems, Geert, Timofte, Radu, and Van Gool, Luc. Hough transform and 3d surf for robust three dimensional classification. vol. 6316, pp. 589–602.

[123] Knuttt, Donald E. On the translation of languages from left to right.

[124] Koch, Sebastian, Matveev, Albert, Jiang, Zhongshi, Williams, Francis, Artemov, Alexey, Burnaev, Evgeny, Alexa, Marc, Zorin, Denis, and Panozzo, Daniele. Abc: A big cad model dataset for geometric deep learning. In *CVPR* (2019).

[125] Koch, Sebastian, Matveev, Albert, Jiang, Zhongshi, Williams, Francis, Artemov, Alexey, Burnaev, Evgeny, Alexa, Marc, Zorin, Denis, and Panozzo, Daniele. ABC: A big cad model dataset for geometric deep learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2019).

[126] Kong, Shu, and Fowlkes, Charless. Recurrent pixel embedding for instance grouping. In *2018 Conference on Computer Vision and Pattern Recognition (CVPR)* (2018).

[127] Kong, Shu, and Fowlkes, Charless C. Recurrent pixel embedding for instance grouping. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 9018–9028.

[128] Krishnamurthy, Venkat, and Levoy, Marc. Fitting smooth surfaces to dense polygon meshes. In *SIGGRAPH* (1996).

[129] Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems* (2012), F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, Eds., vol. 25, Curran Associates, Inc.

[130] Kulkarni, Tejas D, Whitney, Will, Kohli, Pushmeet, and Tenenbaum, Joshua B. Deep convolutional inverse graphics network. In *Proc. NIPS* (2015).

[131] Laidlaw, David H., Trumbore, W. Benjamin, and Hughes, John F. Constructive solid geometry for polyhedral objects. In *Proc. SIGGRAPH* (1986).

[132] Larsson, Gustav, Maire, Michael, and Shakhnarovich, Gregory. Colorization as a proxy task for visual understanding. In *CVPR* (2017).

[133] Le, E., Sung, M., Ceylan, D., Mech, R., Boubekeur, T., and Mitra, N. J. Cpfn: Cascaded primitive fitting networks for high-resolution point clouds. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)* (Los Alamitos, CA, USA, oct 2021), IEEE Computer Society, pp. 7438–7446.

[134] Le, T., and Duan, Y. Pointgrid: A deep network for 3d shape understanding. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pp. 9204–9214.

[135] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. vol. 86, pp. 2278–2324.

[136] Li, Jiaxin, Chen, Ben M, and Hee Lee, Gim. So-net: Self-organizing network for point cloud analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2018), pp. 9397–9406.

[137] Li, Jiaxin, Chen, Ben M, and Lee, Gim Hee. So-net: Self-organizing network for point cloud analysis. *arXiv preprint arXiv:1803.04249* (2018).

[138] Li, L., Sung, Minhyuk, Dubrovina, Anastasia, Yi, L., and Guibas, L. Supervised fitting of geometric primitives to 3d point clouds. *CVPR* (2019), 2647–2655.

[139] Li, Lingxiao, Sung, Minhyuk, Dubrovina, Anastasia, Yi, Li, and Guibas, Leonidas J. Supervised fitting of geometric primitives to 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2019).

[140] Liang, Chen, Berant, Jonathan, Le, Quoc, Forbus, Kenneth D., and Lao, Ni. Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision. In *Proc. ACL* (2017).

[141] Lien, Jyh-Ming, and Amato, Nancy M. Approximate convex decomposition of polyhedra. In *Proceedings of the 2007 ACM Symposium on Solid and Physical Modeling* (2007), SPM '07.

[142] Liu, Dong, Li, Yue, Lin, Jianping, Li, Houqiang, and Wu, Feng. Deep learning-based video coding: A review and a case study. *ACM Comput. Surv. 53*, 1 (feb 2020).

[143] Liu, Yunchao, Wu, Jiajun, Wu, Zheng, Ritchie, Daniel, Freeman, William T., and Tenenbaum, Joshua B. Learning to describe scenes with programs. In *International Conference on Learning Representations* (2019).

[144] Loizou, Marios, Averkiou, Melinos, and Kalogerakis, Evangelos. Learning part boundaries from 3d point clouds. In *Computer Graphics Forum* (2020), vol. 39, Wiley Online Library, pp. 183–195.

[145] Lun, Zhaoliang, Kalogerakis, Evangelos, and Sheffer, Alla. Elements of style: Learning perceptual shape style similarity. *ACM Transactions on Graphics 34*, 4 (2015).

[146] Luo, Wenjie, Schwing, Alexander G., and Urtasun, Raquel. Efficient deep learning for stereo matching. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 5695–5703.

[147] Läuter, H. Silverman, b. w.: Density estimation for statistics and data analysis. chapman & hall, london – new york 1986, 175 pp. *Biometrical Journal 30*, 7 (1988), 876–877.

[148] Macedo, Ives, Gois, Joao Paulo, and Velho, Luiz. Hermite radial basis functions implicits. In *Computer Graphics Forum* (2011), vol. 30, Wiley Online Library, pp. 27–42.

[149] Magnus, Jan R., and Neudecker, Heinz. *Matrix Differential Calculus with Applications in Statistics and Econometrics*, second ed. John Wiley, 1999.

[150] Mardia, Kanti V, and Jupp, Peter E. *Directional statistics*, vol. 494. John Wiley & Sons, 2009.

[151] Marr, David, and Nishihara, Herbert Keith. Representation and recognition of the spatial organization of three-dimensional shapes. *Proceedings of the Royal Society of London. Series B. Biological Sciences 200*, 1140 (1978), 269–294.

[152] Martinovic, Andelo, and Van Gool, Luc. Bayesian Grammar Learning for Inverse Procedural Modeling. In *Proc. CVPR* (2013).

[153] Maturana, Daniel, and Scherer, Sebastian. 3D convolutional neural networks for landing zone detection from LiDAR. In *Proc. ICRA* (2015).

[154] Mehra, Ravish, Zhou, Qingnan, Long, Jeremy, Sheffer, Alla, Gooch, Amy, and Mitra, Niloy J. Abstraction of man-made shapes. *ACM Transactions on Graphics 28*, 5 (2009), #137, 1–10.

[155] Mescheder, Lars, Oechsle, Michael, Niemeyer, Michael, Nowozin, Sebastian, and Geiger, Andreas. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (2019).

[156] Milano, F., Loquercio, Antonio, Rosinol, Antoni, Scaramuzza, D., and Carlone, L. Primal-dual mesh convolutional neural networks. *ArXiv abs/2010.12455* (2020).

[157] Misra, Ishan, Zitnick, C. Lawrence, and Hebert, Martial. Unsupervised learning using sequential verification for action recognition. *CoRR abs/1603.08561* (2016).

[158] Mo, Kaichun, Zhu, Shilin, Chang, Angel, Yi, Li, Tripathi, Subarna, Guibas, Leonidas, and Su, Hao. PartNet: A large-scale benchmark for fine-grained and hierarchical part-level 3D object understanding.

[159] Mo, Kaichun, Zhu, Shilin, Chang, Angel X., Yi, Li, Tripathi, Subarna, Guibas, Leonidas J., and Su, Hao. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In *Computer Vision and Pattern Recognition (CVPR)* (2019).

[160] Mo, Kaichun, Zhu, Shilin, Chang, Angel X., Yi, Li, Tripathi, Subarna, Guibas, Leonidas J., and Su, Hao. PartNet: A large-scale benchmark for fine-grained and hierarchical part-level 3D object understanding. In *Computer Vision and Pattern Recognition (CVPR)* (2019).

[161] Muralikrishnan, Sanjeev, Kim, Vladimir G., and Chaudhuri, Siddhartha. Tags2Parts: Discovering semantic regions from shape tags. In *Proc. CVPR* (2018), IEEE.

[162] Neelakantan, Arvind, Le, Quoc V, and Sutskever, Ilya. Neural Programmer: Inducing Latent Programs with Gradient Descent. In *Proc. ICLR* (2016).

[163] Ng, Andrew Y., Harada, Daishi, and Russell, Stuart J. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proc. ICML* (1999).

[164] Nishida, Gen, Garcia-Dorado, Ignacio, G. Aliaga, Daniel, Benes, Bedrich, and Bousseau, Adrien. Interactive Sketching of Urban Procedural Models. *ACM Transactions on Graphics 35*, 4 (2016).

[165] Noroozi, Mehdi, and Favaro, Paolo. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision* (2016), Springer, pp. 69–84.

[166] Oeztireli, A. C., Guennebaud, G., and Gross, M. Feature Preserving Point Set Surfaces based on Non-Linear Kernel Regression. *Computer Graphics Forum* (2009).

[167] Ohtake, Yutaka, Belyaev, Alexander, and Seidel, Hans-Peter. Ridge-valley lines on meshes via implicit surface fitting. *ACM Trans. Graph. 23*, 3 (aug 2004), 609–612.

[168] Oord, Aaron van den, Li, Yazhe, and Vinyals, Oriol. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748* (2018).

[169] Paschalidou, D., Ulusoy, A. O., and Geiger, A. Superquadrics revisited: Learning 3d shape parsing beyond cuboids. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 10336–10345.

[170] Paschalidou, Despoina, Gool, Luc Van, and Geiger, Andreas. Learning unsupervised hierarchical part decomposition of 3d objects from a single rgb image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2020).

[171] Paschalidou, Despoina, Katharopoulos, Angelos, Geiger, Andreas, and Fidler, Sanja. Neural parts: Learning expressive 3d shape abstractions with invertible neural networks. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (2021).

[172] Paschalidou, Despoina, Ulusoy, Ali Osman, and Geiger, Andreas. Superquadrics revisited: Learning 3d shape parsing beyond cuboids. *CoRR abs/1904.09970* (2019).

[173] Paschalidou, Despoina, Ulusoy, Ali Osman, and Geiger, Andreas. Superquadrics revisited: Learning 3d shape parsing beyond cuboids. In *Computer Vision and Pattern Recognition (CVPR)* (2019).

[174] Paschalidou, Despoina, van Gool, Luc, and Geiger, Andreas. Learning unsupervised hierarchical part decomposition of 3d objects from a single rgb image. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (2020).

[175] Piegl, Les, and Tiller, Wayne. *The NURBS Book*, 2nd ed. Springer-Verlag, Berlin, Heidelberg, 1997.

[176] Poulenard, Adrien, and Ovsjanikov, M. Multi-directional geodesic neural networks via equivariant convolution. *ACM Transactions on Graphics (TOG) 37* (2018), 1 – 14.

[177] Powell, M. J. D. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal 7*, 2 (1964), 155.

[178] Qi, Charles R, Su, Hao, Mo, Kaichun, and Guibas, Leonidas J. PointNet: Deep learning on point sets for 3D classification and segmentation. In *Proc. CVPR* (2017).

[179] Qi, Charles R., Yi, Li, Su, Hao, and Guibas, Leonidas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *Proc. NIPS* (2017).

[180] Radford, Alec, Metz, Luke, and Chintala, Soumith. Unsupervised representation learning with deep convolutional generative adversarial networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (2016), Yoshua Bengio and Yann LeCun, Eds.

[181] Reed, Scott, and de Freitas, Nando. Neural Programmer-Interpreters. In *Proc. ICLR* (2016).

[182] Ren, Shaoqing, He, Kaiming, Girshick, Ross, and Sun, Jian. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Proc. NIPS* (2015).

[183] Riegler, G., Ulusoy, A. O., and Geiger, A. Octnet: Learning deep 3d representations at high resolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 6620–6629.

[184] Riegler, Gernot, Ulusoys, Ali Osman, and Geiger, Andreas. Octnet: Learning deep 3D representations at high resolutions. In *Proc. CVPR* (2017).

[185] Ritchie, Daniel, Mildenhall, Ben, Goodman, Noah D, and Hanrahan, Pat. Controlling Procedural Modeling Programs with Stochastically-ordered Sequential Monte Carlo. *ACM Transactions on Graphics 34*, 4 (2015).

[186] Ritchie, Daniel, Thomas, Anna, Hanrahan, Pat, and Goodman, Noah D. Neurally-Guided Procedural Models: Amortized Inference for Procedural Graphics Programs using Neural Networks. In *Proc. NIPS* (2016).

[187] Romaszko, Lukasz, Williams, Christopher K. I., Moreno, Pol, and Kohli, Pushmeet. Vision-as-inverse-graphics: Obtaining a rich 3d explanation of a scene from a single image. In *ICCV workshops* (2017).

[188] Rooshenas, Amirmohammad, Zhang, Dongxu, Sharma, Gopal, and McCallum, Andrew. Search-guided, lightly-supervised training of structured prediction energy networks. In *Advances in Neural Information Processing Systems* (2019), vol. 32.

[189] Saining Xie, Jiatao Gu, Demi Guo Charles R. Qi Leonidas Guibas Or Litany. Pointcontrast: Unsupervised pre-training for 3d point cloud understanding. In *ECCV* (2020).

[190] Sarlin, Paul-Edouard, DeTone, Daniel, Malisiewicz, Tomasz, and Rabinovich, Andrew. SuperGlue: Learning feature matching with graph neural networks. In *CVPR* (2020).

[191] Sauder, Jonathan, and Sievers, Bjarne. Self-supervised deep learning on point clouds by reconstructing space. In *Advances in Neural Information Processing Systems* (2019), pp. 12942–12952.

[192] Schnabel, R., Wahl, R., and Klein, R. Efficient ransac for point-cloud shape detection. *Computer Graphics Forum 26*, 2 (2007), 214–226.

[193] Schnabel, Ruwen, Wahl, Roland, and Klein, Reinhard. Efficient RANSAC for point-cloud shape detection. *Computer Graphics Forum 26* (06 2007), 214–226.

[194] Schneider, Philip J., and Eberly, David. *Geometric Tools for Computer Graphics*. Elsevier Science Inc., USA, 2002.

[195] Schroff, Florian, Kalenichenko, Dmitry, and Philbin, James. Facenet: A unified embedding for face recognition and clustering. *CoRR abs/1503.03832* (2015).

[196] Schulman, John, Heess, Nicolas, Weber, Theophane, and Abbeel, Pieter. Gradient estimation using stochastic computation graphs. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2* (Cambridge, MA, USA, 2015), MIT Press, p. 3528–3536.

[197] Schult, Jonas, Engelmann, Francis, Kontogianni, Theodora, and Leibe, B. Dualconvmesh-net: Joint geodesic and euclidean convolutions on 3d meshes. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), 8609–8619.

[198] Shapiro, Vadim, and Vossler, Donald L. Construction and optimization of csg representations. *Comput. Aided Des. 23*, 1 (1991).

[199] Shapiro, Vadim, and Vossler, Donald L. Separation for boundary to csg conversion. *ACM Trans. Graph. 12*, 1 (1993).

[200] Sharma, G., Goyal, R., Liu, D., Kalogerakis, E., and Maji, S. Csgnet: Neural shape parser for constructive solid geometry. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pp. 5515–5523.

[201] Sharma, G., Goyal, R., Liu, D., Kalogerakis, E., and Maji, S. Neural shape parsers for constructive solid geometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020), 1–1.

[202] Sharma, Gopal, Goyal, Rishabh, Liu, Difan, Kalogerakis, Evangelos, and Maji, Subhransu. Csgnet: Neural shape parser for constructive solid geometry. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2018), 5515–5523.

[203] Sharma, Gopal, Kalogerakis, Evangelos, and Maji, Subhransu. Learning point embeddings from shape repositories for few-shot segmentation. In *International Conference on 3D Vision (3DV)* (2019).

[204] Sharma, Gopal, Liu, Difan, Maji, Subhransu, Kalogerakis, Evangelos, Chaudhuri, Siddhartha, and Měch, Radomír. Parsenet: A parametric surface fitting network for 3d point clouds. In *Computer Vision – ECCV 2020* (Cham, 2020), Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, Eds., Springer International Publishing, pp. 261–276.

[205] Sharma, Gopal, Yin, Kangxue, Litany, Or, Kalogerakis, Evangelos, Maji, Subhransu, and Fidler, Sanja. Mvdecor: Multi-view dense correspondence learning for fine-grained 3d segmentation, 2022.

[206] Silverman, B. W. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, London, 1986.

[207] Simonyan, Karen, and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *CoRR* (2014).

[208] Smirnov, Dmitriy, Fisher, Matthew, Kim, Vladimir G., Zhang, Richard, and Solomon, Justin. Deep parametric shape predictions using distance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2020).

[209] Sorkine, Olga, and Alexa, Marc. As-rigid-as-possible surface modeling. In *Symposium on Geometry Processing* (2007), pp. 109–116.

[210] Stava, Ondrej, Pirk, Sören, Kratt, Julian, Chen, Baoquan, Měch, R, Deussen, Oliver, and Benes, Bedrich. Inverse Procedural Modelling of Trees. *Computer Graphics Forum 33*, 6 (2014).

[211] Strehl, Alexander, and Ghosh, Joydeep. Cluster ensembles — a knowledge reuse framework for combining multiple partitions. *J. Mach. Learn. Res. 3*, null (Mar. 2003), 583–617.

[212] Su, Hang, Maji, Subhransu, Kalogerakis, Evangelos, and Learned-Miller, Erik G. Multi-view convolutional neural networks for 3d shape recognition. In *Proc. ICCV* (2015).

[213] Su, Hao, Qi, Charles, Mo, Kaichun, and Guibas, Leonidas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *CVPR* (2017).

[214] Su, Jong-Chyi, Gadelha, Matheus, Wang, Rui, and Maji, Subhransu. A deeper look at 3d shape classifiers. In *Proceedings of the European Conference on Computer Vision Workshops (ECCV)* (2018).

[215] Sun, Chunyu, Zou, Qianfang, Tong, Xin, and Liu, Yang. Learning adaptive hierarchical cuboid abstractions of 3D shape collections. *ACM Transactions on Graphics (SIGGRAPH Asia) 38*, 6 (2019).

[216] Sutton, Richard S., and Barto, Andrew G. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

[217] Sutton, Richard S., McAllester, David, Singh, Satinder, and Mansour, Yishay. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Proc. NIPS* (1999).

[218] Talton, Jerry, Yang, Lingfeng, Kumar, Ranjitha, Lim, Maxine, Goodman, Noah, and Měch, Radomír. Learning Design Patterns with Bayesian Grammar Induction. In *Proc. UIST* (2012).

[219] Tatarchenko, Maxim, Park, Jaesik, Koltun, Vladlen, and Zhou., Qian-Yi. Tangent convolutions for dense prediction in 3D. *CVPR* (2018).

[220] Tchapmi, L., Choy, C., Armeni, I., Gwak, J., and Savarese, S. Segcloud: Semantic segmentation of 3d point clouds. In *2017 International Conference on 3D Vision (3DV)* (2017), pp. 537–547.

[221] Teboul, Olivier, Kokkinos, Iasonas, Simon, Loïc, Koutsourakis, Panagiotis, and Paragios, Nikos. Shape Grammar Parsing via Reinforcement Learning. In *Proc. CVPR* (2011).

[222] Thabet, Ali, Alwassel, Humam, and Ghanem, Bernard. MortonNet: Self-Supervised Learning of Local Features in 3D Point Clouds. *arXiv* (Mar 2019).

[223] Thiery, Jean-Marc, Guy, Emilie, and Boubekeur, Tamy. Sphere-meshes: Shape approximation using spherical quadric error metrics. *ACM Transaction on Graphics (Proc. SIGGRAPH Asia 2013) 32*, 6 (2013), Art. No. 178.

[224] Tian, Yonglong, Luo, Andrew, Sun, Xingyuan, Ellis, Kevin, Freeman, William T., Tenenbaum, Joshua B., and Wu, Jiajun. Learning to infer and execute 3d shape programs. In *ICLR* (2019).

[225] Tulsiani, Shubham, Su, Hao, Guibas, Leonidas J., Efros, Alexei A., and Malik, Jitendra. Learning Shape Abstractions by Assembling Volumetric Primitives. In *Proc. CVPR* (2017).

[226] Tulsiani, Shubham, Su, Hao, Guibas, Leonidas J., Efros, Alexei A., and Malik, Jitendra. Learning shape abstractions by assembling volumetric primitives. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (July 2017).

[227] van der Maaten, Laurens. Accelerating t-sne using tree-based algorithms. *Journal of Machine Learning Research 15*, 93 (2014), 3221–3245.

[228] Vanegas, Carlos A, Garcia-Dorado, Ignacio, Aliaga, Daniel G, Benes, Bedrich, and Waddell, Paul. Inverse Design of Urban Procedural Models. *ACM Transactions on Graphics 31*, 6 (2012).

[229] Vondrick, Carl, Shrivastava, Abhinav, Fathi, Alireza, Guadarrama, Sergio, and Murphy, Kevin. Tracking emerges by colorizing videos. *CoRR abs/1806.09594* (2018).

[230] Wada, Kentaro. labelme: Image Polygonal Annotation with Python. `https://github.com/wkentaro/labelme`, 2016.

[231] Wang, Hanchen, Liu, Qi, Yue, Xiangyu, Lasenby, Joan, and Kusner, Matthew J. Unsupervised point cloud pre-training via occlusion completion. In *International Conference on Computer Vision, ICCV* (2021).

[232] Wang, Lingjing, Li, Xiang, and Fang, Yi. Few-shot learning of part-specific probability space for 3d shape segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 4504–4513.

[233] Wang, Peng-Shuai, Liu, Yang, Guo, Yu-Xiao, Sun, Chun-Yu, and Tong, Xin. O-CNN: Octree-based convolutional neural networks for 3D shape analysis. *ACM Trans. Graph. 36*, 4 (2017).

[234] Wang, Peng-Shuai, Sun, Chun-Yu, Liu, Yang, and Tong, Xin. Adaptive o-cnn: A patch-based deep representation of 3d shapes. *ACM Trans. Graph. 37*, 6 (2018).

[235] Wang, Peng-Shuai, Yang, Yu-Qi, Zou, Qian-Fang, Wu, Zhirong, Liu, Yang, and Tong, Xin. Unsupervised 3d learning for shape analysis via multiresolution instance discrimination, 2021.

[236] Wang, Xiaogang, Xu, Yuelang, Xu, Kai, Tagliasacchi, Andrea, Zhou, Bin, Mahdavi-Amiri, Ali, and Zhang, Hao. Pie-net: Parametric inference of point cloud edges. In *Advances in Neural Information Processing Systems* (2020), H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., pp. 20167–20178.

[237] Wang, Xiaolong, and Gupta, Abhinav. Unsupervised learning of visual representations using videos. *CoRR abs/1505.00687* (2015).

[238] Wang, Xinlong, Zhang, Rufeng, Shen, Chunhua, Kong, Tao, and Li, Lei. Dense contrastive learning for self-supervised visual pre-training. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)* (2021).

[239] Wang, Yue, Sun, Yongbin, Liu, Ziwei, Sarma, Sanjay E., Bronstein, Michael M., and Solomon, Justin M. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics 38*, 5 (Oct. 2019).

[240] Wang, Yue, Sun, Yongbin, Liu, Ziwei, Sarma, Sanjay E, Bronstein, Michael M, and Solomon, Justin M. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG) 38*, 5 (2019), 1–12.

[241] Weber, Christopher, Hahmann, Stefanie, and Hagen, Hans. Sharp feature detection in point clouds. In *2010 Shape Modeling International Conference* (2010), pp. 175–186.

[242] Weiss, Daniel. Geometry-based structural optimization on cad specification trees, phd dissertation, eth zurich, 2009.

[243] Wikipedia contributors. Collada, 2019. [Online; accessed 5-August-2019].

[244] Wikipedia contributors. Lowest common ancestor, 2019. [Online; accessed 22-March-2019].

[245] Williams, Ronald J. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning 8*, 3-4 (1992), 229–256.

[246] Wu, Jiajun, and Tenenbaum, Joshua B. Neural Scene De-rendering. In *Proc. CVPR* (2017).

[247] Wu, Q, Xu, K, and Wang, J. Constructing 3D CSG Models from 3D Raw Point Clouds. *Computer Graphics Forum 37*, 5 (2018), 221–232.

[248] Wu, Zhirong, Xiong, Yuanjun, Stella, X Yu, and Lin, Dahua. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018).

[249] Xian, Chuhua, Lin, Hongwei, and Gao, Shuming. Automatic cage generation by improved obbs for mesh deformation. *The Visual Computer 28*, 1 (2012), 21–33.

[250] Yang, Guandao, Huang, Xun, Hao, Zekun, Liu, Ming-Yu, Belongie, Serge, and Hariharan, Bharath. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE International Conference on Computer Vision* (2019), pp. 4541–4550.

[251] Yang, Yaoqing, Feng, Chen, Shen, Yiru, and Tian, Dong. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 206–215.

[252] Yang, Yi, and Ramanan, Deva. Articulated pose estimation with flexible mixtures-of-parts. In *Proc. CVPR* (2011).

[253] Yi, Li, Guibas, Leonidas, Hertzmann, Aaron, Kim, Vladimir G., Su, Hao, and Yumer, Ersin. Learning hierarchical shape segmentation and labeling from online repositories. *ACM Trans. Graph. 36* (July 2017).

[254] Yi, Li, Kim, Vladimir G., Ceylan, Duygu, Shen, I-Chao, Yan, Mengyan, Su, Hao, Lu, Cewu, Huang, Qixing, Sheffer, Alla, and Guibas, Leonidas. A scalable active framework for region annotation in 3d shape collections. *ACM Trans. Graph. 35*, 6 (2016).

[255] Yu, Fenggen, Chen, Zhiqin, Li, Manyi, Sanghi, Aditya, Shayani, Hooman, Mahdavi-Amiri, Ali, and Zhang, Hao. Capri-net: Learning compact CAD shapes with adaptive primitive assembly. *CoRR abs/2104.05652* (2021).

[256] Yu, Lequan, Li, Xianzhi, Fu, Chi-Wing, Cohen-Or, Daniel, and Heng, Pheng-Ann. Ec-net: an edge-aware point set consolidation network. In *ECCV* (2018).

[257] Yuille, A., and Kersten, D. Vision as Bayesian inference: analysis by synthesis? *Trends in Cognitive Sciences* (2006), 301–308.

[258] Yumer, Mehmet Ersin, and Kara, Levent Burak. Surface creation on unstructured point sets using neural networks. *Computer-Aided Design 44*, 7 (2012), 644 – 656.

[259] Zaremba, Wojciech, Mikolov, Tomas, Joulin, Armand, and Fergus, Rob. Learning Simple Algorithms from Examples. In *Proc. ICML* (2016).

[260] Zaremba, Wojciech, and Sutskever, Ilya. Learning to Execute. *arXiv preprint arXiv:1410.4615* (2014).

[261] Zhang, Zaiwei, Girdhar, Rohit, Joulin, Armand, and Misra, Ishan. Self-supervised pretraining of 3d features on any point-cloud. *arXiv preprint arXiv:2101.02691* (2021).

[262] Zhang, Zaiwei, Girdhar, Rohit, Joulin, Armand, and Misra, Ishan. Self-supervised pretraining of 3d features on any point-cloud.

[263] Zhang, Zhengyou. Microsoft kinect sensor and its effect. *IEEE MultiMedia 19*, 2 (Apr. 2012), 4–10.

[264] Zhao, H., Jiang, L., Fu, C., and Jia, J. Pointweb: Enhancing local neighborhood features for point cloud processing. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 5560–5568.

[265] Zhao, Yongheng, Birdal, Tolga, Deng, Haowen, and Tombari, Federico. 3D point capsule networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 1009–1018.

[266] Zhou, Yang, Yin, Kangxue, Huang, Hui, Zhang, Hao, Gong, Minglun, and Cohen-Or, Daniel. Generalized cylinder decomposition. *ACM Trans. Graph. 34*, 6 (2015).

[267] Zhu, Chenyang, Xu, Kai, Chaudhuri, Siddhartha, Yi, Li, Guibas, Leonidas J., and Zhang, Hao. Cosegnet: Deep co-segmentation of 3d shapes with group consistency loss. *CoRR abs/1903.10297* (2019).

[268] Zou, Chuhang, Yumer, Ersin, Yang, Jimei, Ceylan, Duygu, and Hoiem, Derek. 3D-PRNN: Generating Shape Primitives with Recurrent Neural Networks. In *Proc. ICCV* (2017).

[269] Zou, Chuhang, Yumer, Ersin, Yang, Jimei, Ceylan, Duygu, and Hoiem, Derek. 3d-prnn: Generating shape primitives with recurrent neural networks. In *The IEEE International Conference on Computer Vision (ICCV)* (2017).