Title

3D Geometric Deep Learning and its Engineering Applications

Author

Jiang, Chiyu

Publication Date

2020

3D Geometric Deep Learning and its Engineering Applications

by

Chiyu Jiang

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Mechanical Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Philip S. Marcus, Chair
Professor Philip Colella
Professor Tarek Zohdi

Spring 2020

3D Geometric Deep Learning and its Engineering Applications

Abstract

3D Geometric Deep Learning and its Engineering Applications

by

Chiyu Jiang

Doctor of Philosophy in Mechanical Engineering

University of California, Berkeley

Professor Philip S. Marcus, Chair

The physical world is spatially three dimensional, hence understanding the physical and semantic properties of the three dimensional world is crucial to designing engineering systems that operate in and interact with its three dimensional surroundings. Recent advances in Deep Learning has shown great promise in offering a general methodology for creating algorithms that learns to map sensory inputs to desired outcomes by training on a repository of data. Such advances have been particularly salient in fields such as computer vision, where algorithms have achieved near-human or super-human performance in a range of traditionally difficult problems such as image classification, object detection and segmentation. However very often three dimensional problems have been reduced to two dimensional ones for simplicity. In traditional computer vision, inherently three dimensional objects are instead represented by images of their projections onto the two dimensional plane. In computational physics, three dimensional simulations are often simplified as two dimensional counterparts as the computational cost of 3D simulations are often times intractable for many applications. Such simplifications leave out essential information about the actual spatial relationships between objects and can result in inaccurate or erroneous predictions.

The focus of this dissertation is to address the challenges in designing deep learning algorithms and architectures that interact with three dimensional data. Conventional data representation and neural architectures in computer vision do not directly extend to three dimensional counterparts for various reasons. First, data representation for 3D objects is varied and diverse. For instance, in computer graphics and computation physics, geometries are usually represented as simplicial complexes (point clouds, wire frames, triangular meshes, volumetric tetrahedral meshes etc.) for efficiency. In robotics, however, the data representation is mainly determined by the form of the raw sensory inputs to the system, such as point clouds resulting from Lidar scans or Kinect sensors, RGB-D images from depth-enabled cameras, multiview images from binocular or multi-camera setup of the system. Panoramic or fisheye images are also becoming increasingly prevalent in UAVs (Unmanned Aerial Vehicles) and self-driving cars. Second, three dimensional data tend to be orders of magnitudes greater that its two dimensional counter part. Using Cartesian grid repre-

sentation as an example, the storage complexity for 2D images is $\mathcal{O}(n^2)$ whereas for 3D objects that would be $\mathcal{O}(n^3)$. Therefore more efficient encoding and representation methods need to be brought forward to address such challenges. Last but not least, three dimensional data is much more costly to acquire (due to much higher costs of collection, and inaccessibility of such sensors), label (orienting and navigating 3D data on 2D screens requires engineering effort), store and process. In the sections that follow, I will discuss novel methodologies that address these aforementioned challenges, and present various useful applications in computer vision and computational physics that benefit from such methodologies.

In Chapter One, I will present an in-depth overview of these challenges that arise from 3D data, as well as a detailed discussion of existing disciplines from computational physics, climate science, to computer vision that can benefit from progress in 3D Geometric Deep Learning. In Chapter Two, I will present a novel and general methodology for differentiably rasterizing unstructured geometric representations in the form of simplicial complexes. This can serve as a geometry layer within deep neural networks that allows a natural extension of Convolutional Neural Network (CNN) based architectures to a vast collection of 3D representations. In Chapter Three, I will introduce a methodology for natively performing convolutions on the spherical manifold, which is the underlying geometric representation for signals from a range of disciplines, from climate science to panoramic vision. Our methodology is efficient to compute, and naturally prevents the distortion related problems that arise from directly using CNNs on equirectangular projections of spherical signals. Finally, in Chapter Four, I will present a novel continuous implicit 3D representation for large scenes and large physical systems that can allow us to leverage localized learned geometric priors for 3D reconstruction tasks. Moreover, a simple extension to this representation allows us to inject Partial Differential Equation (PDE) constraints within physical systems, in order to facilitate a physics-informed and physics-abiding deep learning architecture.

The formulation of a problem is often more essential than its solution, which may be merely a matter of mathematical or experimental skills.

Albert Einstein

Dedicated to

my wife, *Luna Huang*,
for accompanying me through this otherwise lonely journey for knowledge.

my loving parents, *Weiping Ma* and *Lixi Jiang*,
for having given me nothing but unconditional support, now and always.

Contents

List of Figures

List of Tables

Acknowledgments

Walking through Sather gate as a first graduate student, everyone thinks he can change the world. The same cannot be said for everyone that walks out through the gate. Though I can't say that I have changed the world significantly in a positive direction during my five wonderful years at UC Berkeley, I am thankful and fortunate to have maintained that innocence and curiosity to think that I am more empowered by my education at Berkeley to do so. I owe this time to the incredible group of people I am very fortunate to have worked with.

First and foremost, I owe my heartfelt appreciation to my Ph.D. advisor Dr. Philip Marcus, who has given me unwavering support, and has always believed in me as a researcher. Thank you Phil for leading me towards this incredible journey into 3D Geometric Deep Learning (starting from geometry morphing), supporting me to present my work at various venues, advising me on ways to incorporate physics into machine learning models, and for allowing me to pursue somewhat unconventional fields in computer science.

I am also extremely thankful for the advising and academic support from Prof Matthias Nießner of TUM in the field of 3D Geometric Deep Learning. Thank you for your help and support in launching my academic career in 3D Learning. I have learned a tremendous amount about the field, about finding interesting problems to investigate, and about working my way up the field as a junior researcher. Thank you for taking time to help me shape my numerous research ideas, line by line editing my manuscript, etc., and for connecting me to other scholars in the field.

Additionally, I want to express my deep appreciation for my wonderful collaborators in 3D Learning. A spent a very memorable summer as a reseach intern at Google, where I have had the opportunity to work with a very brilliant group of people across industry and academia. A very special thank-you to Prof. Tom Funkhouser, thank you for your unwavering support through every stage of my project at Google. Thanks to your mentorship, I learned a tremendous amount about being a thorough scientist and researcher. Thank you for going through all the little details of my data, my algorithm, every choice of word for my paper. Thank you for losing sleep during ICCV conference to help me with a visualization code. Thank you Avneesh and Ameesh for being wonderful and accomodating hosts for my unforgettable time at Google, and for working with me side by side through my project. Thank you to my wonderful intern friends Kyle Genova, Fangyin Wei and Yue Wang for the inspiring discussions and for the time and company.

A special shoutout to Jingwei Huang for being a wonderful friend, collaborator and peer through my academic journey in 3D Deep Learning. Thank you for the inspiring discussions, encouragements during my projects, for the all-nighters you pulled before paper deadlines, and for connecting me to some of the brightest minds in the field.

I would also like to thank the incredible team at NERSC, Lawrence Berkeley National Laboratory for supporting me and working with me over the course of the final two years of my graduate

student career. My deepest appreciation to Karthik Kashinath and Prabhat for seeing what's in me early in my graduate career, and for your continued support in my work to pursue the integration of science and deep learning. Thank you for supporting me as a graduate research assistant at LBNL, it meant a lot to me I wouldn't have made it through without the support. I would also like to extend my appreciation for all my wonderful friends and colleagues I have met and worked with at LBNL: Grzegorz Muszynski, Adrian Albert, Jinglong Wu, Ankur Mahesh, Mayur Mudigonda, and Ben Toms. I would also like to thank my other wonderful collaborators in the pursuit of PDE-constrained Deep Learning: Soheil Esmaeilzadeh, Kamyar Azizzadenesheli, Mustafa Mustafa. Last but not least, a special thank-you to Prof. Anima Anandkumar from Caltech for your undivided time and energy in advising my project. Thank you to Dr. Balu Nadiga for hosting me for my memorable week at Los Alamos National Laboratory, and for the interesting discussion on various scientific topics.

Coming back to Berkeley, I owe my thanks to my oral and dissertation committee members: Prof. Philip Colella, Prof. Reza Alam, and Prof. Kathy Yelick, for their detailed comments and valuable feedback on my research. I would like to thank my past and present labmates at the CFD Lab. Thank you to my seniors, Sahuck Oh, Meng Wang and Chung-Hsiang Jiang for your help and support during my very first years at Berkeley. Thank you to Dana Lansigan for being a great research partner. Thank you to my friends and peers, Haris Sheikh, Aidi Zhang, Andrew Sanville, Anzhu Sun, Jinge Wang, Haley Wohlever for your support and company.

Above all I am indebted to my loving family for their unconditional support. Thank you to my parents, Weiping Ma and Lixi Jiang for everything. Words cannot describe my appreciation for all that you have done for me. Thank you to my wife and best friend, Luna Huang, for being there for me through all my ups and downs, listening to my rants, listening to my research ideas, and for accompanying me through this journey (and many more to come).

Chapter 1

3D Geometric Deep Learning

Geometry plays a central role in a range of Engineering applications. Physical objects are geometrical, and more importantly, three dimensional in nature. Inferring semantic and physical properties about geometrical objects is a unified theme across various branches of science and engineering. For instance, aerodynamicists are primarily concerned with the aerodynamic properties of structures such as lift and drag. Structural and civil engineers are interested in the mechanical properties of solid 3D structures, or the interaction between fluid and solid structures. Molecular biologists study the relationship between 3D structures of proteins and its biophysical properties. Robotics engineers are interested in inferring semantic properties from structures to better design machines that interact with structures: If you see a rope, pull. If you see a snake, run. Climate scientists are concerned about the impacts of structures that are orders of magnitudes greater: extreme weather events, such as tropical cyclones and extreme weather events. Predictions made about such structures can impact the lives of many, with long lasting social economic consequences.

Conventionally, such predictions can be made through underlying domain-specific knowledge about these geometric objects and its physical surroundings. The aerodynamics of an airfoil can be inferred through Computational Fluid Dynamics (CFD) simulations, by first modeling the physical laws in the form of governing equations, digitizing and discretizing the geometry of interest in the form of a mesh, then solving for a solution using a numerical algorithm. Similarly, a range of scientific and engineering problems can be tacked in this manner: finite element simulation to structural-mechanical properties, molecular dynamics simulation for protein folding. Even planet scale systems such as the climate system can be simulated at scale through atmospheric modeling. Such an approach, inference via simulation, are successful in some case, but fails in others. Some problems could be too complex, such as modeling the global climate system. Unknown physical dynamics across various levels of the system accumulate error. Some problems require too much compute, at a level that is prohibitively expensive. Turbulence is a good example. The governing equations (Navier-Stokes) are well understood, but an accurate simulation of turbulence requires resolving the simulation at an extremely fine scale (the Komogorov scale) can be prohibitively expensive and impractical. For some problems, we simply do not know what the physical laws should be - such as inferring semantic properties of objects in 3D or 2D images. Many conventional methods that utilize hand-crafted image features perform poorly for high level tasks that require semantic understanding of scenes and images.

Machine Learning, in particular Deep Learning, has become a rapidly emerging and growing field, for its ability in solving a wide range of problems with very little domain specific knowledge about such problems to begin with. Deep Learning algorithms utilize artificial neural networks, a class of algorithms that contain learnable parameters that can be mathematically optimized to fit training data. This framework has been very successful at solving a range of engineering problems, from computer vision to natural language processing. This method is extremely general and flexible, and does not require much domain specific knowledge about underlying mechanisms. Such properties can be ideal for problems that are vague in nature, where providing examples are much easier than abstracting general rules and patterns. For instance, it is very easy to give examples of what is a cat, but extremely difficult to abstract a general rule for recognizing cats. Deep Learning as a general methodology shows incredible promise at tackling the aforementioned important yet difficult problems that involve performing inference on 3D geometric instances.

Extending existing methods in Deep Learning for 3D data is, however, far from being a straightforward process. First, data representation for 3D objects is varied and diverse. For instance, in computer graphics and computation physics, geometries are usually represented as simplicial complexes (point clouds, wire frames, triangular meshes, volumetric tetrahedral meshes etc.) for efficiency. In robotics, however, the data representation is mainly determined by the form of the raw sensory inputs to the system, such as point clouds resulting from Lidar scans or Kinect sensors, RGB-D images from depth-enabled cameras, multiview images from binocular or multi-camera setup of the system. Panoramic or fisheye images are also becoming increasingly prevalent in UAVs (Unmanned Aerial Vehicles) and self-driving cars. Second, three dimensional data tend to be orders of magnitudes greater that its two dimensional counter part. Using Cartesian grid representation as an example, the storage complexity for 2D images is $\mathcal{O}(n^2)$ whereas for 3D objects that would be $\mathcal{O}(n^3)$. Therefore more efficient encoding and representation methods need to be brought forward to address such challenges. Last but not least, three dimensional data is much more costly to acquire (due to much higher costs of collection, and inaccessibility of such sensors), label (orienting and navigating 3D data on 2D screens requires engineering effort), store and process.

New work in the 3D Learning literature have emerged to address these aforementioned challenges to 3D learning. Concerning the various modes of representation, different neural architectures have been designed to address the varied input and output structures. A host of deep neural architectures have been designed to process point cloud data [84, 82, 107, 83], generate point cloud data [30, 63], process mesh data [42, 36], generate mesh data [103, 102], process octree structured voxelized data [105], generate mesh structured data [98].

This dissertation attempts to address the following aspects in 3D Deep Learning:

In Chapter 2, I will present a novel framework for differentiably rasterizing arbitrary 2D or 3D geometries represented as a simplicial complex, which we call the Deep Differentiable Simplex Layer (DDSL). A simplicial complex is a superposition of arbitrary simplicies of arbitrary degrees. Many well known geometric representations are inherently simplicial complexes. For example, point clouds (0-simplex), wireframes (1-simplex), triangular meshes (2-simplex), and tetrahedral meshes (3-simplex). The generalizability of such a layer allows for a unified framework across different geometry representations. The differentiability of such a layer allows its natural use as a layer within an end-to-end trainable neural network. I will show various engineering applications of such a differentiable simplex layer, including its use in aerodynamic design optimization, and image segmentation.

In Chapter 3, I will present a novel framework that naturally extends CNNs to spherical manifolds. We generalize the convolution kernel to be a parameterized linear combination of differential operators, such that it can be efficiently computed and implemented on meshes. Using such a reparameterized convolution kernel, we can acquire a much more parameter efficient network architecture that enables state-of-the-art performance in classification and segmentation tasks for spherical signals. As an application for this approach, we use the network for performing semantic segmentation of indoor scenes from panoramic images. We also utilized this spherical network for a climate science application, where we achieved good performance in segmenting and detecting various extreme weather events (atmospheric rivers, tropical cyclones) in climate simulations.

In Chapter 4, I present an efficient 3D representation for large 3D scenes, which we call the Local Implicit Grid (LIG) representation. LIG is a new 3D shape representation designed for scalability and generality. The motivating idea is that most 3D surfaces share geometric details at some scale – i.e., at a scale smaller than an entire object and larger than a small patch. We train an autoencoder to learn an embedding of local crops of 3D shapes at that size. Then, we use the decoder as a component in a shape optimization that solves for a set of latent codes on a regular grid of overlapping crops such that an interpolation of the decoded local shapes matches a partial or noisy observation. We demonstrate the value of this proposed approach for 3D surface reconstruction from sparse point observations, showing significantly better results than alternative approaches.

Chapter 2

Deep Learning on Simplicial Complexes

2.1 Introduction

The simplicial complex (i.e., simplex mesh) is a flexible and general representation for non-uniform geometric signals. Various commonly-used geometric representations, including point clouds, wire-frames, polygons, triangular mesh, tetrahedral mesh etc., are examples of simplicial complexes. Leveraging deep learning architectures for such non-uniform geometric signals has been of increasing interest, and varied methodologies and architectures have been presented to deal with varied representations [8].

In this study, we propose a Deep Differentiable Simplex Layer (DDSL), which performs differentiable rasterization of arbitrary simplex mesh-based geometric signals. The DDSL is based upon simplex Non-Uniform Fourier Transform (NUFT) [45] for the forward-pass, which is highly generalizable across arbitrary topologies. Furthermore, we find the general differential form of the simplex NUFT, allowing for an efficient backward pass. Our work differs from previous work in the literature on differentiable rendering in two major ways. First, our network is generalizable across arbitrary simplex degrees and dimensions, making it a unified framework for a range of geometric representations. Second, while other differentiable renderers are specifically posed for projective-rendering by projecting 3D meshes to 2D grids, the DDSL is capable of in-situ rasterization in the original dimension. Building on the differentiable nature of the rasterizer, we explore two unique use cases. First, using the differentiablity of the DDSL, we can utilize Convolutional Neural Network (CNN) based deep learning models as surrogate models of physical properties for shape optimization, which is useful in a range of engineering disciplines. Secondly, using the DDSL as a neural network layer, we can formulate a differentiable rasterization loss that allows for end-to-end generation of shapes using a direct supervised approach, which can be useful in a range of computer vision problems.

As an example of the two use cases, we perform three experiments. First, to validate the effectiveness of gradient propagation through the layer, we illustrate with the toy problem of MNIST shape optimization, where we can use gradients propagated through the neural network and DDSL to manipulate and transform the input polygon mesh into a target digit (Sec. 4.3). Next, to further

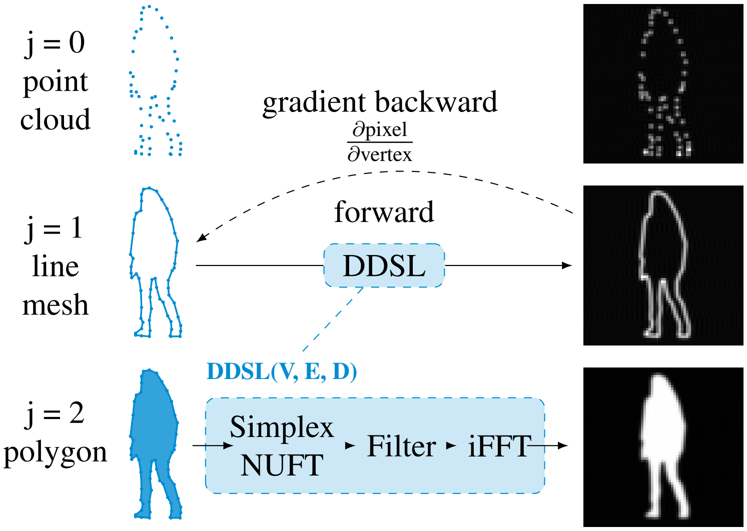


Figure 2.1: A schematic of the DDSL layer with 2D simplex meshes. The DDSL algorithm is general for handling simplex meshes of arbitrary dimensions and simplex degrees. The input to DDSL is a simplex mesh described by three matrices: float matrix V for vertex coordinates, uint matrix E for simplex connectivity, and float matrix D for per-simplex density (constant density of 1 in the example above). A raster image of arbitrary resolution can be produced. The gradient of per-pixel intensity with respect to each spatial coordinate in V can be computed analytically within the DDSL layer.

illustrate potential applications of neural shape optimization enabled by the DDSL, we investigate the classic engineering problem of airfoil optimization and show that the shape optimization pipeline effectively manipulates the input shape into a desired lift-drag ratio (Sec. 4.3). Finally, to illustrate the effectiveness of the differentiable rasterization loss, we train a polygon generating neural network end-to-end with direct supervision to generate polygonal segmentation masks for image segmentation (Sec. 2.4). With the novel rasterization loss, we surpass state-of-the-art in the polygon segmentation task, with a much simpler network architecture and training scheme.

In summary, we contribute the following:

- We propose the DDSL, which is a differentiable rasterizer for arbitrary simplex-mesh based geometries. Its differentiable nature allows for its effective integration in deep neural networks.
- We show that the DDSL effectively facilitates shape optimization for engineering applications such as aerodynamic optimzation of airfoils, using neural networks as surrogate models.
- We show that the DDSL can be used to produce a differentiable rasterization loss, which can be used to create direct supervision to facilitate end-to-end training of shape generators, with applications in polygonal segmentation mask generation.

- We develop and release code for effectively integrating the DDSL into deep neural networks*, with compelling computational performance benchmarks.

2.2 Related Work

We present a brief overview of geometric representations for deep learning, various related differentiable renderers, and related work in the space of our two exemplary applications.

Geometric Representations for Deep Learning In general, there are two classes of geometric representations, either in its native form of simplex meshes, or in a raster form which can be efficiently processed with grid-based network architectures such as CNNs. As simplex meshes come in various forms and dimensions (point clouds, meshes etc.), there is a vast body of literature for different geometric signals of different simplex degrees and dimensions. For example, PointNets have been specially designed for point clouds [84, 82], various algorithms perform convolutions natively on the mesh manifold, [46, 41, 7], the graph [27, 59, 113] etc.

Grid-based algorithms on the other hand require the rasterization of a simplex-mesh based geometric signal for further processing by CNNs. Examples of such include binary-voxel based algorithms [73, 109], Truncated Signed-Distance Function (TSDF) based algorithms [21, 115, 94, 23], multi-view image based algorithms [96, 52], and hybrids [50, 19]. Compared to deep learning methods that directly perform convolutions on the simplex mesh, grid-based methods are more generalizable across shape topologies and computationally easier to implement, since it leverages highly efficient tensor operators such as 2D/3D convolution kernels for rasterized data. However, conventional voxelization methods are not differentiable with respect to the input mesh, and differentiable rasterizers have been proposed to close the gap between simplex and grid representations.

Differentiable Rasterization in Deep Learning Recently, a series differentiable projective renderers have been proposed. [69] proposed an approximate differentiable rasterizer for inverse graphics. [53] proposed a deep neural renderer that uses linear approximations for the gradients of the pixel intensity with respect to the vertex positions. [64] introduced a differentiable ray-tracer for differentiability of additional rendering effects. Very recently, [67] proposed a differentiable rasterizer that approximates rendering derivatives with soft boundaries. Various studies in face mesh reconstruction applications [33, 101, 100, 87] and general mesh reconstruction tasks [51, 61] utilize some form of differentiable rasterization to facilitate gradient flows in neural networks.

Shape Optimization Shape optimization is essential in a broad range of engineering fields, including aerodynamic, mechanical, structural, and architectural designs. Traditionally, shape optimization algorithms couple gradient-based or gradient-free optimizers (e.g., genetic algorithms, simulated annealing) with physics simulators, e.g., Computational Fluid Dynamics (CFD) and multiphysics software for evaluation. For aerodynamic shape optimization, the adjoint method

*Code available: `https://github.com/maxjiang93/DDSL`

Notation	Description
d	Dimension of Euclidean space $\mathbb{R}^d$
j	Degree of simplex. Point $j = 0$, Line $j = 1$, Tri. $j = 2$, Tet. $j = 3$
n, N	Index of the n-th element among a total of N elements
Ω_n^j	Domain of n-th element of order j
$\boldsymbol{x}$	Cartesian space coordinate vector. $\boldsymbol{x} = (x, y, z)$
$\boldsymbol{k}$	Spectral domain coordinate vector. $\boldsymbol{k} = (u, v, w)$
p	Index of a point in a simplex element. $p \in \mathbb{N}, p \leq j + 1$
i	Imaginary number unit

Table 2.1: List of math symbols in our method.

has been used for gradient-based optimizations with sensitivities acquired from physics simulators [81, 43]. Recently, machine learning algorithms such as multilayer perceptrons have been used as surrogate models for the response surface to speed up evaluation and optimization [56, 70]. More recently, CNNs have been used for the evaluation of aerodynamic properties [116], and gradient-based optimization methods coupled with CNNs have been explored [39]. However, direct manipulation of input mesh has not been achieved due to the lack of in-situ differentiable rasterization of polygons and 3D meshes.

Image Segmentation with Polygon Masks Image segmentation is a central task in computer vision, and has been thoroughly studied. Much of the work in the image segmentation literature creates pixel-level masks [68, 88, 104, 38, 26, 65]. However, more recently, to address the need of assisting human annotators to create ground-truth segmentation labels, new network architectures such as PolygonRNN [10] and PolygonRNN++ [1] have been proposed for creating polygonal segmentation masks given ground-truth bounding boxes. Our work targets this application to explore a more effective and efficient polygon generating network using our DDSL-enabled rasterization loss.

2.3 Method

DDSL Overview

A schematic of the DDSL layer is presented in Fig. 2.1. The DDSL layer consists of three consecutive mathematical operations, first computing the Fourier transform of the simplicial complex by uniformly sampling it in the spectral domain, followed by a spectral filtering step by multiplying the spectral signal with a Gaussian filter to eliminate ringing effects. Lastly, we use the inverse Fourier Transform (iFFT) to acquire the physical raster image corresponding to the input. Since the forward and backward methods of the filtering step (an element-wise product) and iFFT are well known, we focus our analysis on the simplex NUFT, which we derive and detail below.

Mathematical Description

We represent discrete geometric signals as weighted simplicial complexes. We provide the following definitions for a j-simplex and a j-simplex mesh:

Definition 2.3.1 (j-simplex)**.** A *simplex* is the generalization of the two-dimensional triangle in other dimensions. The j-simplex determined by $j+1$ affinely independent points $v_0, \dots, v_j \in \mathbb{R}^n$ is

$$\begin{aligned} C &= \mathbf{conv}\{v_0, \dots, v_j\} \\ &= \{\theta_0 v_0 + \dots + \theta_j v_j \mid \boldsymbol{\theta} \succeq 0,\ \mathbf{1}^T\boldsymbol{\theta} = 1\} \end{aligned} \tag{2.1}$$

where $\mathbf{1}$ is the vector with all entries one.

Definition 2.3.2 (j-simplex mesh)**.** A simplicial complex consisting only of j-simplices is a homogeneous simplicial j-complex, or a *j-simplex mesh*.

Example 2.3.1 (Examples of simplices and simplex meshes)**.** A 0-simplex is a point, a 1-simplex is a line, a 2-simplex is a triangle, and a 3-simplex is a tetrahedron. The 0-, 1-, 2-, and 3-simplicial complexes are the point cloud and linear, triangular, and tetrahedral meshes, respectively.

Definition 2.3.3 (Functions over a j-simplex element and a j-simplex mesh)**.** The Piecewise-Constant Function (PCF) over a j-simplex mesh consisting of N simplices is the superposition of the density functions $f_n^j(\boldsymbol{x})$ for each j-simplex with domain Ω_n^j and signal density ρ_n:

$$f_n^j(\boldsymbol{x}) = \begin{cases} \rho_n, \boldsymbol{x} \in \Omega_n^j \\ 0, \boldsymbol{x} \notin \Omega_n^j \end{cases}, \quad f^j(\boldsymbol{x}) = \sum_{n=1}^{N} f_n^j(\boldsymbol{x}) \tag{2.2}$$

For the forward pass, we use the NUFT of a PCF over a j-simplex mesh.

Proposition 2.3.1 (Forward pass)**.** The NUFT of a PCF over a simplex in a mesh is

$$F_n^j(\boldsymbol{k}) = \rho_n i^j \gamma_n^j S \tag{2.3}$$

$$S := \sum_{t=1}^{j+1} \frac{e^{-i\sigma_t}}{\prod_{l=1, l\neq t}^{j+1} (\sigma_t - \sigma_i)}, \quad \sigma_t := \boldsymbol{k} \cdot \boldsymbol{x}_t \tag{2.4}$$

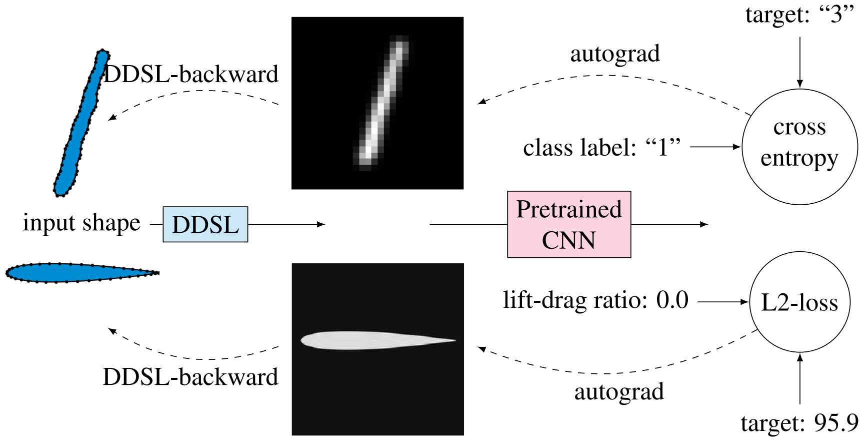


Figure 2.2: Schematic of deep learning model driven shape optimization pipeline.

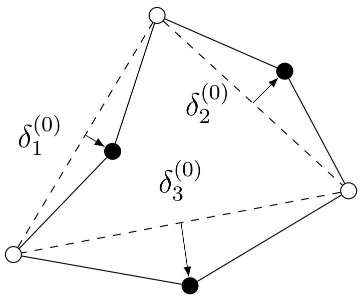


Figure 2.3: Schematic for the hierarchical polygon generation process in PolygonNet. New nodes in the next hierarchy are generated by offsetting edge center in normal direction by δ.

where γ_n^j is the content distortion factor, which is the ratio between the simplex content and the unit orthogonal simplex content. The simplex content C_n^j is computed using the Cayley-Menger determinant:

$$C_n^j = \sqrt{\frac{(-1)^{j+1}}{2^j (j!)^2} det(\hat{B}_n^j)} \tag{2.5}$$

$$\hat{B}_n^j := \begin{bmatrix} 0 & 1 & 1 & 1 & \dots \\ 1 & 0 & d_{12}^2 & d_{13}^2 & \dots \\ 1 & d_{21}^2 & 0 & d_{23}^2 & \dots \\ 1 & d_{31}^2 & d_{32}^2 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \end{bmatrix} \tag{2.6}$$

where each element d_{st}^2 of $\hat{B}_n^j$ is the squared distance between points s and t. The content of the unit orthogonal simplex C_I^j is $1/j!$, so the content distortion factor is

$$\gamma_n^j = \frac{C_n^j}{C_I^j} = j! C_n^j \tag{2.7}$$

From the linearity of the Fourier transform, the NUFT of a PCF over an entire j-simplex mesh is

$$F^j(\boldsymbol{k}) = \sum_{n=1}^{N} F_n^j(\boldsymbol{k}) = \sum_{n}^{N} \rho_n i^j \gamma_n^j S \tag{2.8}$$

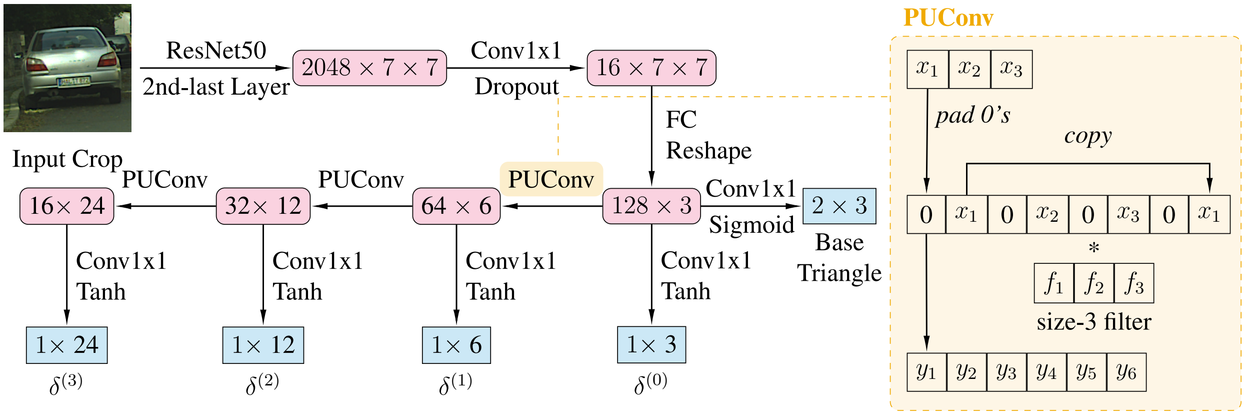


Figure 2.4: Schematic of the deep learning architecture for polygon segmentation (PolygonNet). All intermediate layers are followed by BatchNorm and ReLU. A Periodic Upsampling Convolution (PUConv) is used to generate vertex offsets (δ) at the consecutive level. For each level, we learn a learnable scale factor for all offsets.

For efficient computing, we use the auxiliary node method (AuxNode), which utilizes signed content.

Corollary 2.3.1 (AuxNode)**.** *To compute the Fourier transform of uniform signals in j-polytopes represented by its watertight $(j-1)$-simplex mesh using AuxNode, Eqn. (2.3) is modified as follows:*

$$F_n^j(\boldsymbol{k}) = i^j \sum_{n'=1}^{N'_n} s_{n'}\gamma_{n'}^j \left(\frac{(-1)^j}{\prod_{l=1}^j \sigma_l} + \sum_{t=1}^{j} \frac{e^{-i\sigma_t}}{\sigma_t \prod_{l=1,l\neq t}^{j}(\sigma_t - \sigma_l)} \right) \tag{2.9}$$

where $s_{n'}\gamma_{n'}^j$ is the signed content distortion factor for the n'th auxiliary j-simplex where $s_{n'} \in \{-1, 1\}$. For practical purposes, assume that the auxiliary j-simplex is in $\mathbb{R}^d$ where $d = j$. The signed content distortion factor is computed using the determinant of the Jacobian matrix for parameterizing the auxiliary simplex to a unit orthogonal simplex:

$$s_{n'}\gamma_{n'}^j = j!\det(J) = j!\det([\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_j]) \tag{2.10}$$

Proof. Refer to [45]. □

For the backward pass, we derive the analytic derivative of the NUFT with respect to the vertex coordinates of a j-simplex mesh. Following from the product rule, we require the derivatives of the content distortion factor γ_n^j and the summation term S to obtain the entire derivative of $F_n^j(\boldsymbol{k})$.

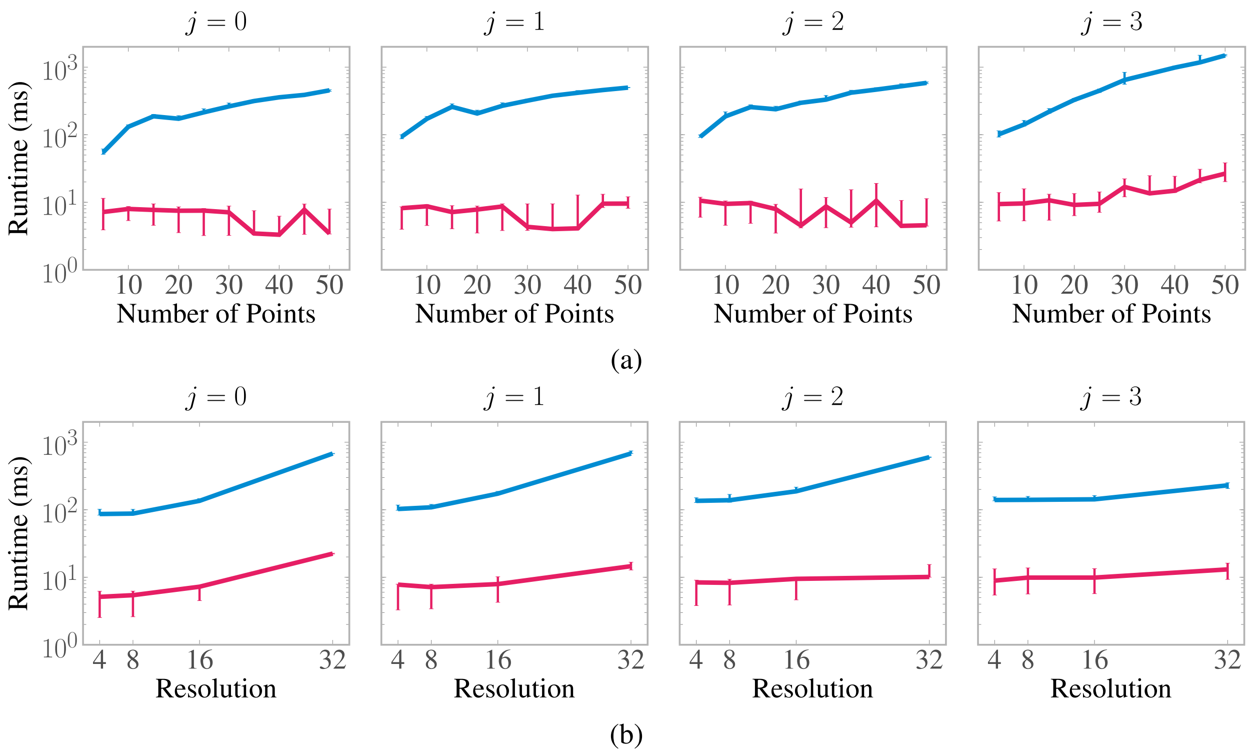


Figure 2.5: Comparison of the analytic (pink) and numeric (blue) derivative runtimes for the (a) mesh size and (b) resolution tests. All rasters are computed for a square cube, and resolution is per dimension.

Lemma 2.3.1 (Derivative of the content distortion factor)**.** *The derivative of γ_n^j with respect to vertex coordinate $\boldsymbol{x}_p$ is*

$$\frac{\partial \gamma_n^j}{\partial \boldsymbol{x}_p} = \frac{(-1)^{j+1}/2^j}{\gamma_n^j} \sum_{\substack{m=1 \\ m \neq p}}^{j+1} A_{pm} \boldsymbol{D}_{pm} \tag{2.11}$$

where $\boldsymbol{D}_{pm} = 2(\boldsymbol{x}_p - \boldsymbol{x}_m)$ and A_{pm} is the element in the $(p+1)$th row and $(m+1)$th column of $adj(\hat{B_n^j})$.

Lemma 2.3.2 (Derivative of the summation term)**.** *Let S_t be one term in the summation term S:*

$$S_t := \frac{e^{-i\sigma_t}}{\prod_{l=1, l \neq t}^{j+1} (\sigma_t - \sigma_l)} \tag{2.12}$$

The derivative of the summation term with respect to $\boldsymbol{x}_p$ is

$$\frac{\partial S}{\partial \boldsymbol{x}_p} = \left(-iS_p + \sum_{t=1, t \neq p}^{j+1} \frac{S_t + S_p}{\sigma_t - \sigma_p} \right) \boldsymbol{k} \tag{2.13}$$

where $\boldsymbol{k}$ is the spectral domain coordinate vector.

Proposition 2.3.2 (Backward pass)**.** Following from Lemmas 2.3.1 and 2.3.2, the derivative of $F_n^j(\boldsymbol{k})$ with respect to a point $\boldsymbol{x}_p$ in the simplex element n is

$$\frac{\partial F_n^j(\boldsymbol{k})}{\partial \boldsymbol{x}_p} = \rho_n i^j \left(\Lambda \boldsymbol{k} + \Gamma \sum_{\substack{m=1 \\ m \neq p}}^{j+1} A_{pm} \boldsymbol{D}_{pm} \right) \tag{2.14}$$

where A_{pm} is the element in the pth row and mth column of $adj(\hat{B}_n^j)$ starting at $p = 0$ and $m = 0$,

$$\Lambda := \gamma_n^j \left(-iS_p + \sum_{t=1, t\neq p}^{j+1} \frac{S_t + S_p}{\sigma_t - \sigma_p} \right) \tag{2.15}$$

$$\Gamma := \frac{(-1)^{j+1}/2^j}{\gamma_n^j} S \tag{2.16}$$

We provide a detailed derivation of Eqn. 2.14 as well as proofs of Lemmas 2.3.1 and 2.3.2 in Sec. A.1 of the Appendix.

Deep Learning Architectures and Pipelines

We present the a schematic of the deep learning model-driven shape optimization (Sec. 4.3) in Fig. 2.2, and a schematic of the polygon segmentation network (PolygonNet) in Figs. 2.3 and 2.4. A detailed description of the architectures is presented in Appendix A.2.

2.4 Experiments

Performance Benchmarking

We compare the runtime of our implementation of the backward pass over the DDSL with that of the numeric derivatives calculated using the finite difference method.

Experiment Setup We perform tests for the 0-, 1-, 2-, and 3-simplex meshes in 3-dimensional space and examine the effects of mesh size (number of points in the mesh) and image resolution. We test mesh sizes ranging from 5 to 50 points and resolutions ranging from 4 to 32, and we run each test 100 times to acquire a distribution of data. For each run, we randomly generate a 3-dimensional simplex mesh of varied simplex degrees, varied densities, with random gradient values on each raster pixel. We then calculate the analytic and numeric derivatives for the DDSL using our implementation of Eqn. 2.14 and the finite difference method, respectively, and time each calculation.

Analysis of complexity Since the analytic finite difference backward pass for computing the gradients using Eqn. 2.14 requires computing each pair of spectral coefficient and each vertex in a j-simplex, the computational complexity for the finite difference backward pass is the same as the forward pass, $\mathcal{O}((j+1)n_e m)$, for a mesh of n_e simplices and a raster of m degrees of freedom. Finite difference, on the other hand, requires n_v forward computations, each of complexity $\mathcal{O}((j+1)n_e m)$. Assuming $n_v \propto n_e$, the Finite Difference evaluation is of complexity $\mathcal{O}((j+1)n_e^2 m)$.

Results The results of our mesh size and resolution runtime tests are shown in Fig. 2.5. In both tests and for all j-simplices, our implementation of the analytic derivative consistently outperforms the numerical method for calculating the derivative by $10 \sim 100\times$ in the range we tested.

Shape Optimization

We demonstrate the utility of the DDSL through the task of shape optimization. Since many physical characteristics depend on shape, shape optimization is an important and challenging task across many fields of science and engineering. We show that the DDSL allows us to accomplish this shape optimization task due to the analytic nature of its derivative.

General Experiment Setup We pre-process each shape into a polygon of the shape's boundary. The polygons are rasterized using the DDSL. We train neural networks on the raster images, and we use the gradients out of these neural networks for the shape optimization task.

Using gradient descent, we optimize a shape to a prescribed target value, which can be a shape classification or a physical quantity. Since we implemented the DDSL as a differentiable neural network layer, we can obtain the gradient of the target value with respect to the original shape directly from the neural network. Rather than directly manipulating vertices, we further propagate this gradient to control points attached to the original shape for enhanced robustness. Each control point has 3 degrees of freedom: translation in the x and y directions, and rotation about the point. More details about the control points are given in Sec. A.1. We iterate the shape optimization process until the loss converges to zero.

MNIST We first demonstrate shape optimization using the DDSL with the MNIST dataset of handwritten digits. Rather than using the traditional pixel images, we use polygons of the digits as inputs. The polygon form of MNIST digits can be acquired by contouring the original images. The objective of this experiment is to optimize a digit in the MNIST dataset to a target digit.

Airfoils We further illustrate the functionality of the DDSL with the more practical task of aerodynamic shape optimization. For this experiment, we optimize an airfoil to a prescribed lift-drag ratio, which is related to the efficiency of an aerodynamic body. We use the `airfoiltools.com` database of consisting of 1,636 airfoils of aircraft wings and turbine blades, along with pre-computed physical quantities such as drag and lift coefficients at different angles of attack and

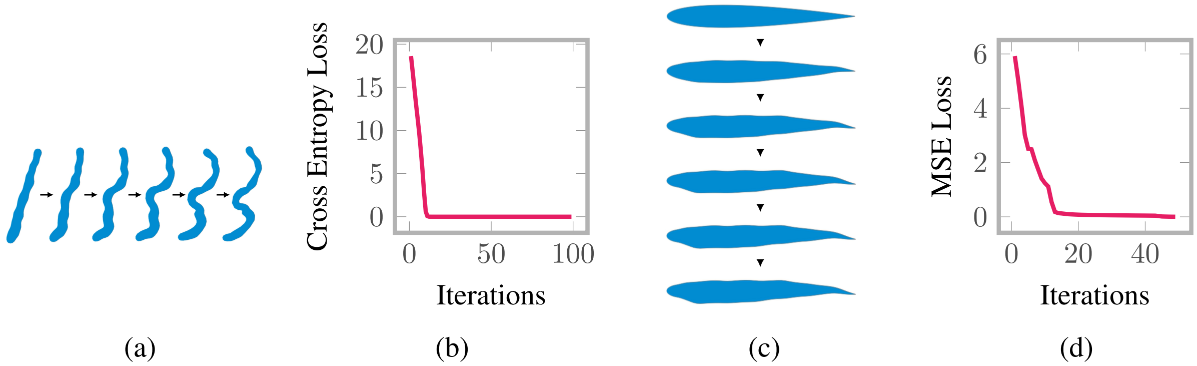


Figure 2.6: Optimization of (a), (b): a ‘1’ from the MNIST dataset to a ‘3’ by minimizing the cross-entropy between the input and the target class. (c), (d): the NACA 0012 airfoil (with an original lift-drag ratio of 0) to a lift-drag ratio of 95.9. The airfoil is set at an angle of attack of zero, and the Reynolds number is set to 1×10^6.

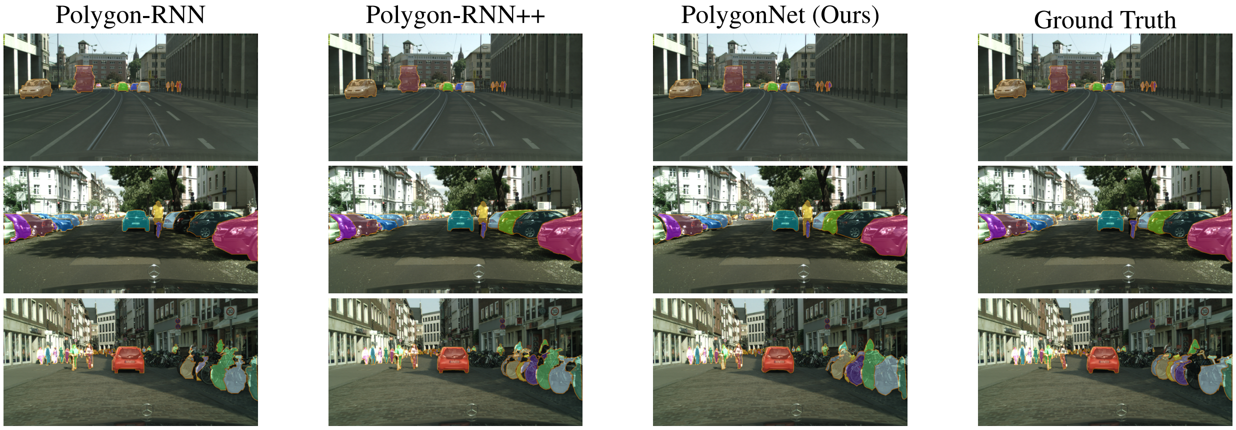


Figure 2.7: Visualization of image segmentation results. Ground-truth bounding boxes are given for all models to create image crops as inputs to the networks.

Reynolds numbers, acquired from CFD simulations. Airfoils are originally represented as polygons and rasterized using the DDSL. We then train a neural network to predict lift-drag ratios of airfoils at specific angles of attack and Reynolds numbers and use this neural network for the shape optimization task. When optimizing the airfoil shape, we specify the angle of attack of the airfoil and the Reynolds number of the flow.

Results We show some iterations of the shape optimization process for the MNIST and airfoil experiments as well as graphs showing the loss over each iteration in Figs. 4.4, respectively. The success of the DDSL in the shape optimization task is most intuitively clear in the MNIST experiment, where the original digit, ‘1,’ is transformed into a ‘3.’ In the airfoil experiment, the

Model	Bicycle	Bus	Person	Train	Truck	Motorcycle	Car	Rider	Mean
SquareBox [10]	35.41	53.44	26.36	39.34	54.75	39.47	46.04	26.09	40.11
Dilation10 [114]	46.80	48.35	49.37	44.18	35.71	26.97	61.49	38.21	43.89
DeepMask [79]	47.19	69.82	47.93	62.20	63.15	47.47	61.64	52.20	56.45
SharpMask [80]	52.08	73.02	53.63	64.06	65.49	51.92	65.17	56.32	60.21
Polygon-RNN [10]	52.13	69.53	63.94	53.74	68.03	52.07	71.17	60.58	61.40
Polygon-RNN++ [1]	**63.06**	81.38	**72.41**	64.28	**78.90**	62.01	**79.08**	**69.95**	71.38
PolygonNet (Ours)	62.26	**84.38**	68.62	**82.42**	76.57	**63.57**	78.08	64.10	**72.50**

Table 2.2: Comparison of Cityscape image segmentation IoU against baseline algorithms on test set.

lift-drag ratio increased, as desired. The optimized shape is an airfoil with its trailing edge deflected downwards, resembling an aircraft deploying its flaps at takeoff to increase lift. Both experiments exhibit a monotonic decrease in loss, which converges to zero, confirming that optimization was achieved.

Segmentation Mask Generation

To further illustrate applications of the DDSL layer in deep learning applications, we experiment on the task of image segmentation by generating polygonal masks. In contrast to conventional segmentation frameworks that output pixel masks, directly predicting polygons allows for a more efficient and flexible output structure, and has been shown to be effective in assisting human annotators in labeling new datasets [10, 1].

Experiment Setup For direct comparison with state-of-the-art, we follow the experiment setup of [10] and [1] for predicting polygonal masks. In contrast to the conventional setup of instance segmentation, we assume crops of input images given ground-truth bounding boxes, and we output the corresponding polygonal masks using our neural network. Following the two studies, we train and test our model on the Cityscapes dataset [18]. The Cityscapes dataset is one of the most comprehensive benchmarks for instance segmentation, containing 2975 training, 500 validation, and 1525 test images labeled with 8 semantic classes. We follow the two studies for an alternative split of the original dataset, since the original test images do not provide ground-truth instances. The new partitions consists of 40174 / 3448 / 8440 image crops of train/validation/test sets, each of size 224×224.

Training We use two losses for training the model, a multi-resolution rasterization loss, and a smoothness loss. The losses are defined as:

$$\mathcal{L}_{\text{mres}} = \sum_{i,res} ||D_{res}(G_\theta^{(i)}(x)) - D_{res}(y)||_1 \tag{2.17}$$

$$i \in \{0, 1, 2, 3\}, res \in \{224, 112, 56, 28\}$$

$$\mathcal{L}_{\text{smooth}} = \frac{1}{n}\sum_j^n (\frac{A_j(G_\theta^{(3)}(x))}{\pi} - 1)^2 \tag{2.18}$$

$$\mathcal{L} = \mathcal{L}_{\text{mres}} + \lambda\mathcal{L}_{\text{smooth}} \tag{2.19}$$

where D_{res} is DDSL rasterization at resolution res, $G_\theta^{(i)}$ is the polygon output from the polygon generator network parameterized by θ, up to level i, x and y are the input images and the ground-truth polygons, A_j is the j-th angel of the polygon, and λ is the smoothness penalty term. We train the model (see Fig. 2.4) end-to-end using the loss defined above. We weight the loss of each class inversely proportional to the label frequencies in the training set. See more details in Appendix A.2.

Results We evaluate our model against state-of-the-art models and detail the results in Table 2.2, where we evaluate runtime on a single Titan X (Pascal) GPU. We provide a visual comparison in Fig. 2.7. Our model surpasses state of the art for class-averaged IoU. In particular, the simplicity of our network architecture is highlighted in Table 2.3. While Polygon-RNN++ was unable to propagate gradients through IoU scores, it uses IoU as a reward to an additional reinforcement learning model, which adds additional complexities to the overall architecture. It also uses additional graph neural network to upsample and finetune the polygons. Due to the differentiable rasterization loss, our model uses a single CNN-based polygon generator. In comparison to Polygon-RNN++, our model achieves a 100x speed-up with a quarter of the total model parameters.

Model	# Params	Runtime (s)
Polygon-RNN	58M	2.0332 ± 0.0168
Polygon-RNN++	100M	2.3241 ± 0.0181
PolygonNet (Ours)	**24M**	$\mathbf{0.0287} \pm 0.0022$

Table 2.3: Comparison of network parameters and evaluation time for a batch of 16 image crops.

2.5 Conclusion

We propose the DDSL as a differentiable simplex layer for neural networks. We present a unifying framework for differentiable rasterization of arbitrary geometrical signals represented on a simplicial complex. We further show two geometric applications of this method: we can effectively propagate gradients across the DDSL for shape optimization, and we can utilize the DDSL to construct a differentiable rasterization loss that allows for a simple, yet effective, polygon generating network that surpasses state of the art in segmentation IoU as well as runtime and parameter efficiency.

Chapter 3

Deep Learning on the Spherical Manifold

3.1 Introduction

A wide range of machine learning problems in computer vision and related areas require processing signals in the spherical domain; for instance, omnidirectional RGBD images from commercially available panorama cameras, such as Matterport [13], panaramic videos coupled with LIDAR scans from self-driving cars [31], or planetary signals in scientific domains such as climate science [86]. Unfortunately, naively mapping spherical signals to planar domains results in unwanted distortions. Specifically, projection artifacts near polar regions and handling of boundaries makes learning with 2D convolutional neural networks (CNNs) particularly challenging and inefficient. Very recent work, such as [16] and [29], propose network architectures that operate natively in the spherical domain, and are invariant to rotations in the $\mathcal{SO}(3)$ group. Such invariances are desirable in a set of problems – e.g., machine learning problems of molecules – where gravitational effects are negligible and orientation is arbitrary. However, for yet a different class of problems at large, assumed orientation information is crucial to the predictive capability of the network. A good example of such problems is the MNIST digit recognition problem, where orientation plays the dominant role in distinguishing digits “6” and “9”. Other examples include omnidirectional images, where images are naturally oriented by gravity; and planetary signals, where planets are naturally oriented by their axis of rotation.

In this work, we present a new convolutional kernel for CNNs on arbitrary manifolds and topologies, discretized by an unstructured grid (i.e., mesh), and focus on its applications towards the spherical domain approximated by an icosahedral spherical mesh. We propose and evaluate the use of a new parameterization scheme for CNN convolution kernels, which we call Parameterized Differential Operators (PDOs), which is easy to implement on unstructured grids. This parameterization scheme utilizes only 4 parameters for each kernel, and achieves significantly better performance than competing methods, with a much smaller parameter budget. In particular, we illustrate its use towards various machine learning problems in computer vision and climate science.

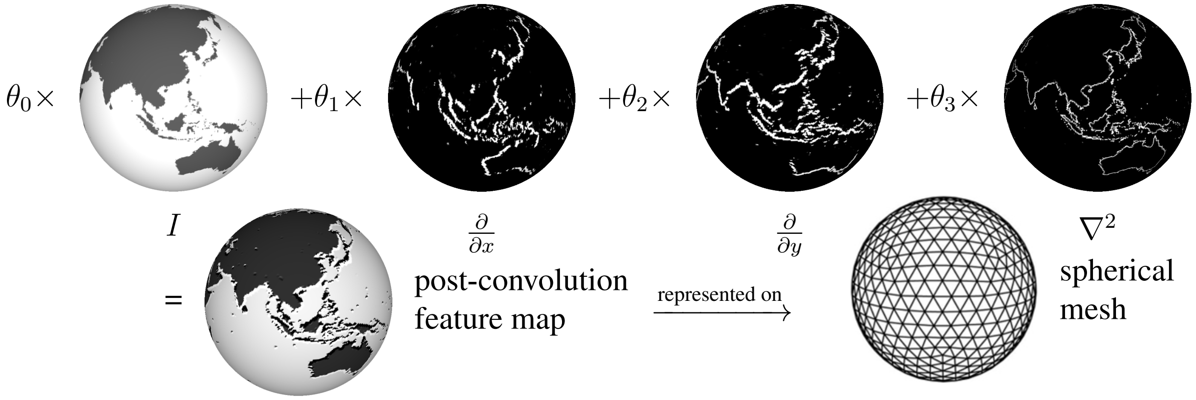


Figure 3.1: Illustration for the MeshConv operator using parameterized differential operators to replace conventional learnable convolutional kernels.

In summary, our contributions are as follows:

- We present a general approach for orientable CNNs on unstructured grids using parameterized differential operators.
- We show that our spherical model achieves significantly higher parameter efficiency compared to state-of-the-art network architectures for 3D classification tasks and spherical image semantic segmentation.

We organize the structure of the paper as follows. We first provide an overview of related studies in the literature in Sec.3.2; we then introduce details of our methodology in Sec.3.3, followed by an empirical assessment of the effectiveness of our model in Sec.3.4. Finally, we evaluate the design choices of our kernel parameterization scheme in Sec.3.5.

3.2 Background

Spherical CNNs A first and foremost concern for processing spherical signals is regarding distortions introduced by the curved spherical surface. [95] process equirectangular images with regular convolutions with increased kernel sizes near polar regions where greater distortions are introduced by the planar mapping. [17] and [117] use a constant kernel that samples points on the tangent plane of the spherical image to reduce distortions. A slightly different line of literature explores rotational-equivariant implementations of spherical CNNs. [16] proposed spherical convolutions with intermediate feature maps in $\mathcal{SO}(3)$ that are rotational-equivariant. [29] used spherical harmonic basis to achieve similar results.

Reparameterized Convolutional Kernel Related to our approach in using parameterized differential operators, several works utilize the diffusion kernel for efficient Machine Learning and

CNNs. [60] was among the first to suggest the use of diffusion kernel on graphs. [3] propose Diffusion-Convolutional Neural Networks (DCNN) for efficient convolution on graph structured data. [7] introduce a generalization of classic CNNs to non-Euclidean domains by using a set of oriented anisotropic diffusion kernels. [89] explore the reparameterization of convolutional kernels using parabolic and hyperbolic differential basis with regular grid images.

Non-Euclidean Convolutions Related to our work on performing convolutions on manifolds represented by an unstructured grid (i.e., mesh), works in geometric deep learning address similar problems [8]. Other methods perform graph convolution by parameterizing the convolution kernels in the spectral domain, thus converting the convolution step into a spectral dot product [9, 27, 58, 113]. [72] perform convolutions directly on manifolds using cross-correlation based on geodesic distances and [71] use an optimal surface parameterization method (seamless toric covers) to parameterize genus-zero shapes into 2D signals for analysis using conventional planar CNNs.

Image Semantic Segmentation Image semantic segmentation is a classical problem in computer vision, and there has been an impressive body of literature studying semantic segmentation of planar images ([88, 5, 68, 44, 104]). [92] study semantic segmentation of equirectangular omnidirectional images, but in the context of image inpainting, where only a partial view is given as input. [2, 13] provide benchmarks for semantic segmentation of 360 panorama images. In the 3D learning literature, researchers have looked at 3D semantic segmentation on point clouds or voxels [25, 84, 106, 99, 24]. Our method also targets the application domain of image segmentation by providing a more efficient convolutional operator for spherical domains, for instance, focusing on panoramic images [13].

3.3 Method

Parameterized Differential Operators

We present a novel scheme for efficiently performing convolutions on manifolds approximated by a given underlying mesh, using what we name as Parameterized Differential Operators. To this end, we reparameterize the learnable convolution kernel as a linear combination of differential operators. Such reparameterization provides two distinct advantages: first, we can drastically reduce the number of parameters per given convolution kernel, allowing for an efficient and lean learning space; second, as opposed to the cross-correlation type convolution on mesh surfaces [72], which requires large amounts of geodesic computations and interpolations, first and second order differential operators can be efficiently estimated using only the one-ring neighborhood.

In order to illustrate the concept of PDOs, we draw comparisons to the conventional 3×3 convolution kernel in the regular grid domain. The 3×3 kernel parameterized by parameters $\boldsymbol{\theta}$: $\mathcal{G}_{\boldsymbol{\theta}}^{3\times3}$ can be written as a linear combination of basis kernels which can be viewed as delta functions at constant offsets:

$$\mathcal{G}_{\boldsymbol{\theta}}^{3\times 3}(x,y) = \sum_{i=-1}^{1}\sum_{j=-1}^{1} \delta(x-i, y-j) \tag{3.1}$$

Due to the linearity of the cross-correlation operator ($*$), the output feature map can be expressed as a linear combination of the input function cross-correlated with different basis functions. Defining the linear operator Δ_{ij} to be the cross-correlation with a basis delta function, we have:

$$\Delta_{ij}\mathcal{F}(x,y) := \mathcal{F}(x,y) * \delta(x-i, y-j) \tag{3.2}$$

$$\mathcal{F}(x,y) * \mathcal{G}_{\boldsymbol{\theta}}^{3\times 3}(x,y) = \sum_{i=-1}^{1}\sum_{j=-1}^{1} \theta_{ij}\Delta_{ij}\mathcal{F}(x,y) \tag{3.3}$$

In our formulation of PDOs, we replace the cross-correlation linear operators Δ_{ij} with differential operators of varying orders. Similar to the linear operators resulting from cross-correlation with basis functions, differential operators are linear, and approximate local features. In contrast to cross-correlations on manifolds, differential operators on meshes can be efficiently computed using Finite Element basis, or derived by Discrete Exterior Calculus. In the actual implementation below, we choose the identity (I, 0th order differential, same as Δ_{00}), derivatives in two orthogonal spatial dimensions (∇_x, ∇_y, 1st order differential), and the Laplacian operator (∇^2, 2nd order differential):

$$\mathcal{F}(x,y) * \mathcal{G}_{\boldsymbol{\theta}}^{diff} = \theta_0 I\mathcal{F} + \theta_1\nabla_x\mathcal{F} + \theta_2\nabla_y\mathcal{F} + \theta_3\nabla^2\mathcal{F} \tag{3.4}$$

The identity (I) of the input function is trivial to obtain. The first derivative (∇_x, ∇_y) can be obtained by first computing the per-face gradients, and then using area-weighted average to obtain per-vertex gradient. The dot product between the per-vertex gradient value and the corresponding x and y vector fields are then computed to acquire $\nabla_x\mathcal{F}$ and $\nabla_y\mathcal{F}$. For the sphere, we choose the east-west and north-south directions to be the x and y components, since the poles naturally orient the spherical signal. The Laplacian operator on the mesh can be discretized using the contangent formula:

$$\nabla^2\mathcal{F} \approx \frac{1}{2\mathcal{A}_i}\sum_{j\in\mathcal{N}(i)} (\cot\alpha_{ij} + \cot\beta_{ij})(\mathcal{F}_i - \mathcal{F}_j), \tag{3.5}$$

where $\mathcal{N}_i$ is the nodes in the neighboring one-ring of i, $\mathcal{A}_i$ is the area of the dual face corresponding to node i, and α_{ij} and β_{ij} are the two angles opposing edge ij. With this parameterization of the convolution kernel, the parameters can be similarly optimized via backpropagation using standard stochastic optimization routines.

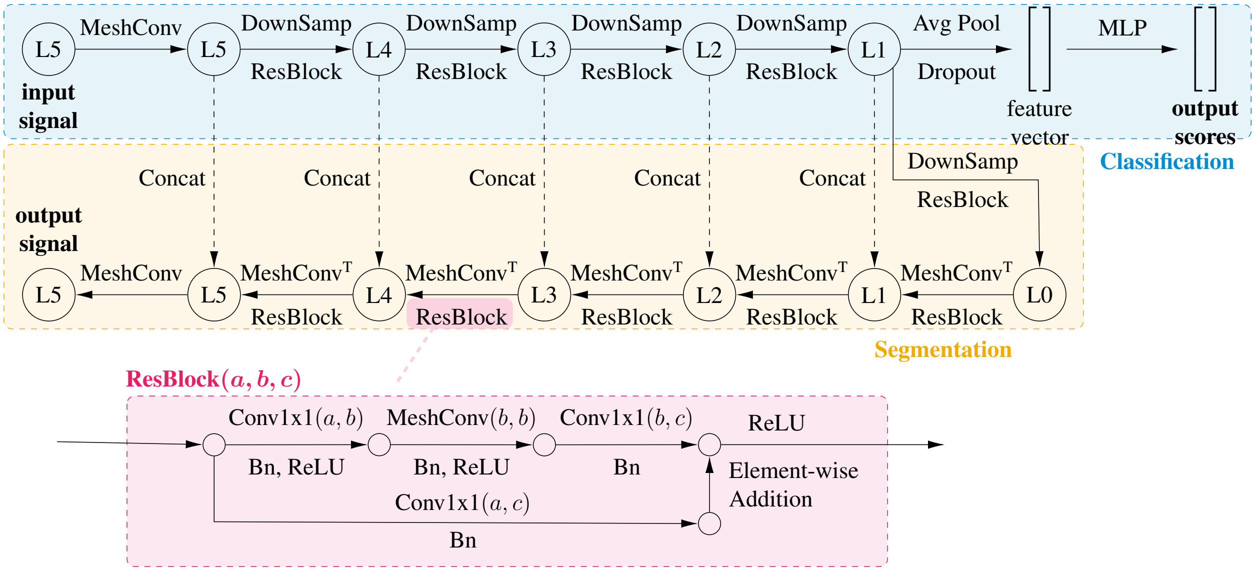


Figure 3.2: Schematics for model architecture for classification and semantic segmentation tasks, at a level-5 input resolution. Ln stands for spherical mesh of level-n as defined in Sec. 3.3. MeshConv is implemented according to Eqn. 3.4. MeshConv^T first pads unknown values at the next level with 0, followed by a regular MeshConv. DownSamp samples the values at the nodes in the next mesh level. A ResBlock with bottleneck layers, consisting of Conv1x1 (1-by-1 convolutions) and MeshConv layers is detailed above. In the decoder, ResBlock is after each MeshConv^T and Concat.

Icosahedral Spherical Mesh

The icosahedral spherical mesh [6] is among the most uniform and accurate discretizations of the sphere. A spherical mesh can be obtained by progressively subdividing each face of the unit icosahedron into four equal triangles and reprojecting each node to unit distance from the origin. Apart from the uniformity and accuracy of the icosahedral sphere, the subdivision scheme for the triangles provides a natural coarsening and refinement scheme for the grid that allows for easy implementations of pooling and unpooling routines associated with CNN architectures. See Fig. 3.1 for a schematic of the level-3 icosahedral spherical mesh.

For the ease of discussion, we adopt the following naming convention for mesh resolution: starting with the unit icosahedron as the level-0 mesh, each progressive mesh resolution is one level above the previous. Hence for a level-l mesh:

$$n_f = 20 \cdot 4^l; n_e = 30 \cdot 4^l; n_v = n_e - n_f + 2 \tag{3.6}$$

where n_f, n_e, n_v stands for the number of faces, edges, and vertices of the spherical mesh.

Model	Accuracy(%)	Number of Parameters
S2CNN [16]	96.00	58k
SphereNet [17]	94.41	196k
Ours	**99.23**	62k

Table 3.1: Results on the MNIST dataset for validating the use of Parameterized Differential Operators. Our model achieves state-of-the-art performance with comparable number of training parameters.

Model Architecture Design

A detailed schematic for the neural architectures in this study is presented in Fig. 3.2. The schematic includes architectures for both the classification and regression network, which share a common encoder architecture. The segmentation network consists of an additional decoder which features transpose convolutions and skip layers, inspired by the U-Net architecture [88]. Minor adjustments are made for different tasks, mainly surrounding adjusting the number of input and output layers to process signals at varied resolutions. A detailed breakdown for model architectures, as well as training details for each task in the Experiment section (Sec. 3.4), is provided in the appendix (Appendix Sec. B.1).

3.4 Experiments

Spherical MNIST

To validate the use of parameterized differential operators to replace conventional convolution operators, we implemented such neural networks towards solving the classic computer vision benchmark task: the MNIST digit recognition problem [62].

Experiment Setup We follow [16] by projecting the pixelated digits onto the surface of the unit sphere. We further move the digits to the equator to prevent coordinate singularity at the poles. We benchmark our model against two other implementations of spherical CNNs: a rotational-invariant model by [16] and an orientable model by [17]. All models are trained and tested with non-rotated digits to illustrate the performance gain from orientation information.

Results and Discussion Our model outperforms its counterparts by a significant margin, achieving the best performance among comparable algorithms, with comparable number of parameters. We attribute the success in our model to the gain in orientation information, which is indispensable for many vision tasks. In contrast, S2CNN [16] is rotational-invariant, and thus has difficulties distinguishing digits “6” and “9”.

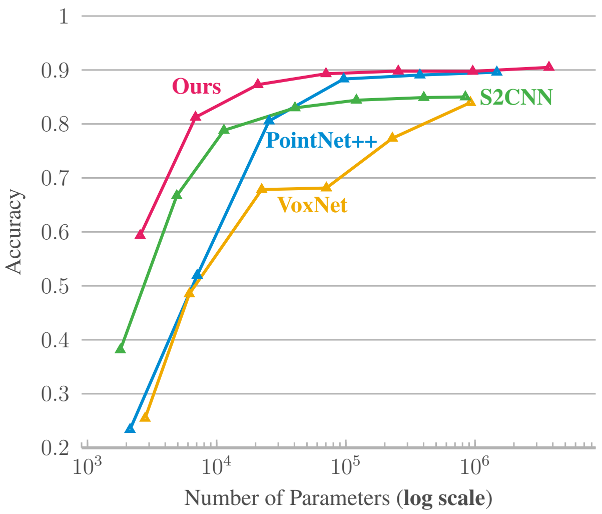


Figure 3.3: Parameter efficiency study on ModelNet40, benchmarked against representative 3D learning models consuming different input data representations: PointNet++ using point clouds as input, VoxNet consuming binary-voxel inputs, S2CNN consuming the same input structure as our model (spherical signal). The abscissa is drawn based on log scale.

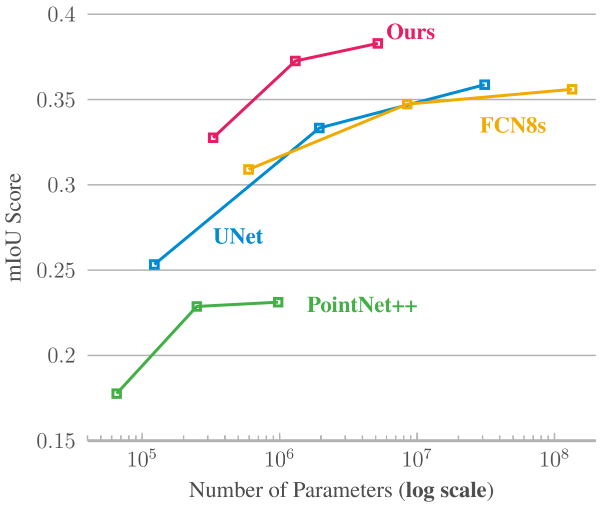


Figure 3.4: Parameter efficiency study on 2D3DS semantic segmentation task. Our spherical segmentation model outperforms the planar and point-based counterparts by a significant margin across all parameter regimes.

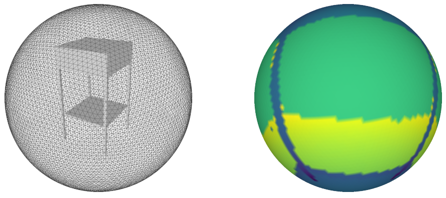
3D Object Classification

We use the ModelNet40 benchmark [110], a 40-class 3D classification problem, to illustrate the applicability of our spherical method to a wider set of problems in 3D learning. For this study, we look into two aspects of our model: peak performance and parameter efficiency.

Experiment Setup To use our spherical CNN model for the object classification task, we preprocess the 3D geometries into spherical signals. We follow [16] for preprocessing the 3D CAD models. First, we normalize and translate each mesh to the coordinate origin. We then encapsulate each mesh with a bounding level-5 unit sphere and perform ray-tracing from each point to the origin. We record the distance from the spherical surface to the mesh, as well as the $\sin, \cos$ of the incident angle. The data is further augmented with the 3 channels corresponding to the convex hull of the input mesh, forming a total of 6 input channels. An illustration of the data preprocessing process is presented in Fig. 3.5. For peak performance, we compare the best performance achievable by our model with other 3D learning algorithms. For the parameter efficiency study, we progressively reduce the number of feature layers in all models without changing the overall model architecture. Then, we evaluate the models after convergence in 250 epochs. We benchmark our

Model	Input	Accu. (%)
3DShapeNets [110]	voxels	84.7
VoxNet [73]	voxels	85.9
PointNet [84]	points	89.2
PointNet++ [82]	points	91.9
DGCNN [106]	points	**92.2**
S2CNN [16]	spherical	85.0
SphericalCNN [29]	spherical	88.9
Ours	spherical	90.5

Table 3.2: Results on ModelNet40 dataset. Our method compares favorably with state-of-the-art, and achieves best performance among networks utilizing spherical input signals.



(a) Original CAD model and spherical mesh.

(b) Resulting surface distance signal.

Figure 3.5: Illustration of spherical signal rendering process for a given 3D CAD model.

results against PointNet++ [84], VoxNet [85], and S2CNN[*][16].

Results and Discussion Fig. 3.3 shows a comparison of model performance versus number of parameters. Our model achieves the best performance across all parameter ranges. In the low-parameter range, our model is able to achieve approximately 60% accuracy for the 40-class 3D classification task with a mere $2000+$ parameters. Table 3.2 shows a comparison of peak performance between models. At peak performance, our model is on-par with comparable state-of-the-art models, and achieves the best performance among models consuming spherical input signals.

Omnidirectional Image Segmentation

We illustrate the semantic segmentation capability of our network on the omnidirectional image segmentation task. We use the Stanford 2D3DS dataset [2] for this task. The 2D3DS dataset consists of 1,413 equirectangular images with RGB+depth channels, as well as semantic labels across 13 different classes. The panoramic images are taken in 6 different areas, and the dataset is officially split for a 3-fold cross validation. While we are unable to find reported results on the semantic segmentation of these omnidirectional images, we benchmark our spherical segmentation algorithm against classic 2D image semantic segmentation networks as well as a 3D point-based model, trained and evaluated on the same data.

[*] We use the author's open-source implementations: PointNet++, VoxNet, S2CNN. We run PointNet++ with standard input of 1000 points with xyz coordinates.

Experiment Setup First, we preprocess the data into a spherical signal by sampling the original rectangular images at the latitude-longitudes of the spherical mesh vertex positions. Input RGB-D channels are interpolated using bilinear interpolation, while semantic labels are acquired using nearest-neighbor interpolation. We input and output spherical signals at the level-5 resolution. We use the official 3-fold cross validation to train and evaluate our results. We benchmark our semantic segmentation results against two classic semantic segmentation networks: the U-Net [88] and FCN8s [68]. We also compared our results with a 3D point-based method, PointNet++ [82] using (x, y, z,r,g,b) inputs reconstructed from panoramic RGBD images. We evaluate the network performance under two standard metrics: mean Intersection-over-Union (mIoU), and pixel-accuracy. Similar to Sec. 3.4, we evaluate the models under two settings: peak performance and a parameter efficiency study by varying model parameters. We progressively decimate the number of feature layers uniformly for all models to study the effect of model complexity on performance.

Results and Discussion Fig. 3.4 compares our model against state-of-the-art baselines. Our spherical segmentation outperforms the planar baselines for all parameter ranges, and more significantly so compared to the 3D PointNet++. We attribute PointNet++'s performance to the small sample size of the training data. Fig. 3.6 shows a visualization of our semantic segmentation performance compared to the ground truth and the planar baselines.

Climate Pattern Segmentation

To further illustrate the capabilities of our model, we evaluate our model on the climate pattern segmentation task. We follow [75] for preprocessing the data and acquiring the ground-truth labels for this task. This task involves the segmentation of Atmospheric Rivers (AR) and Tropical Cyclones (TC) in global climate model simulations. Following [75], we analyze outputs from a 20-year run of the *Community Atmospheric Model v5 (CAM5)* [76]. We benchmark our performance against [75] for the climate segmentation task to highlight our model performance. We preprocess the data to level-5 resolution.

Results and Discussion Segmentation accuracy is presented in Table 3.3. Our model achieves better segmentation accuracy as compared to the baseline models. The baseline model [75] trains and tests on random crops of the global data, whereas our model inputs the entire global data and predicts at the same output resolution as the input. Processing full global data allows the network to acquire better holistic understanding of the information, resulting in better overall performance.

3.5 Ablation Study

We further perform an ablation study for justifying the choice of differential operators for our convolution kernel (as in Eqn. 3.4). We use the ModelNet40 classification problem as a toy example and use a 250k parameter model for evaluation. We choose various combinations of

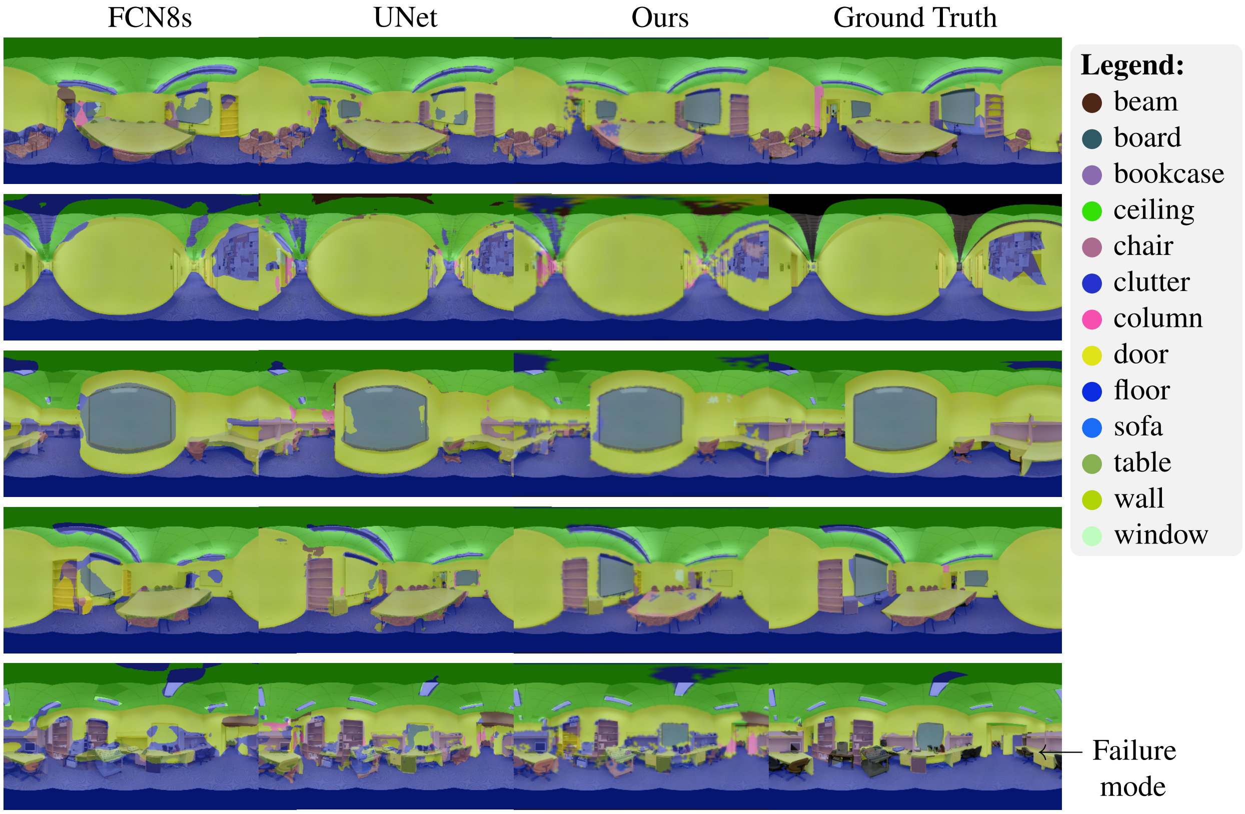


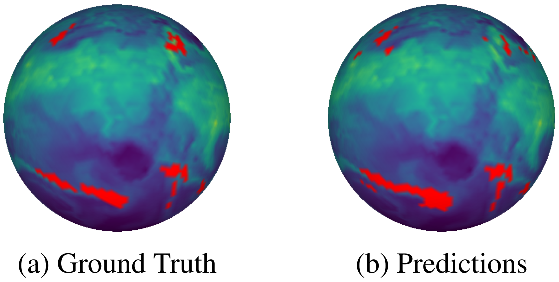
Figure 3.6: Visualization of semantic segmentation results on test set. Our results are generated on a level-5 spherical mesh and mapped to the equirectangular grid for visualization. Model underperforms in complex environments, and fails to predict ceiling lights due to incomplete RGB inputs.

Model	Background (%)	TC (%)	AR (%)	Mean (%)
[75]	97	74	65	78.67
Ours	97	**94**	**93**	**94.67**

Table 3.3: We achieve better accuracy compared to our baseline for climate pattern segmentation.

Convolution kernel	Accuracy
$I + \frac{\partial}{\partial y} + \nabla^2$	0.8748
$I + \frac{\partial}{\partial x} + \nabla^2$	0.8809
$I + \nabla^2$	0.8801
$I + \frac{\partial}{\partial x} + \frac{\partial}{\partial y}$	0.8894
$I + \frac{\partial}{\partial x} + \frac{\partial}{\partial y} + \nabla^2$	**0.8979**

Table 3.4: Results for the ablation study. The choice of kernel that includes all differential operator components achieve the best accuracy, validating our choice of kernel in Eqn.3.4.


(a) Ground Truth

(b) Predictions

Figure 3.7: Visualization of segmentation for Atmospheric River (AR). Plotted in the background is Integrated Vapor Transport (IVT), whereas red masks indicates the existance of AR.

differential operators, and record the final classification accuracy. Results for the ablation study is presented in Table 3.4. Our choice of differential operator combinations in Eqn. 3.4 achieves the best performance among other choices, and the network performance improves with increased differential operators, thus allowing for more degrees of freedom for the kernel.

3.6 Conclusion

We have presented a novel method for performing convolution on unstructured grids using parameterized differential operators as convolution kernels. Our results demonstrate its applicability to machine learning problems with spherical signals and show significant improvements in terms of overall performance and parameter efficiency. We believe that these advances are particularly relevant with the widespread of omnidirectional signals, for instance, as captured by real-world 3D or LIDAR panorama sensors.

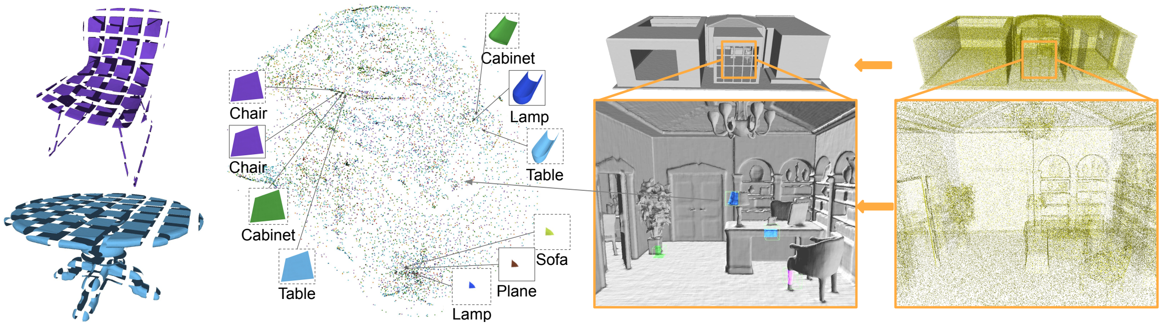
Chapter 4

Deep Learning on Implicit 3D Representations

4.1 Introduction

Geometric representation for scenes has been central to various tasks in computer vision and graphics, including geometric reconstruction, compression, and higher-level tasks such as scene understanding, object detection and segmentation. An effective representation should generalize well across a wide range of semantic categories, scale efficiently to large scenes, exhibit a rich expressive capacity for representing sharp features and complex topologies, and at the same time leverage learned geometric priors acquired from data.

In the last years, several works have proposed new network architectures to allow conventional geometric representations such as point clouds [84, 30, 112], meshes [103, 34], and voxel grids



(a) Training parts (ShapeNet). (b) t-SNE of part embeddings. (c) Reconstructing scenes w/ Local Implicit Grids

Figure 4.1: We learn an embedding of parts from objects in ShapeNet [11] using a part autoencoder with an implicit decoder. We show that this representation of parts is generalizable across object categories, and easily scalable to large scenes. By localizing implicit functions in a grid, we are able to reconstruct entire scenes from points via optimization of the latent grid.

[22, 109] to leverage data priors. More recently, a neural implicit representation [14, 74, 77] has been proposed as an alternative to these approaches for its expressive capacity for representing fine geometric details. However, the aforementioned works focus on learning representations for whole objects within one or a few categories, and they have not been studied in the context of generalizing to other categories, or scaling to large scenes.

In this paper we propose a learned 3D shape representation that generalizes and scales to arbitrary scenes. Our key observation is that although different shapes across different categories and scenes have vastly different geometric forms and topologies on a global scale, they share similar features at a certain local scale. For instance, sofa seats and car windshields have a similar curved parts, tabletops and airplane wings both have thin sharp edges, etc.. While no two shapes are the same at the macro scale, and all shapes on a micro-scale can be locally approximated by an angled plane, there exists an intermediate scale (a "part scale"), where a meaningful shared abstraction for all geometries can be learned by a single deep neural network. We aim to learn shape priors at that scale and then leverage them in a scalable and general 3D reconstruction algorithm.

To this end, we propose the Local Implicit Grid (LIG) representation, a regular grid of overlapping part-sized local regions, each encoded with an implicit feature vector. We learn to encode/decode geometric parts of objects at a part scale by training an implicit function autoencoder on 13 object categories from ShapeNet [11]. Then, armed with the pretrained decoder, we propose a mechanism to optimize for the Latent Implicit Grid representation that matches a partial or noisy scene observation. Our representation includes a novel overlapping latent grid mechanism for confidence-weighted interpolation of learned local features for seamlessly representing large scenes. We illustrate the effectiveness of this approach by targeting the challenging application of scene reconstruction from sparse point samples, where we are able to faithfully reconstruct entire scenes given only sparse point samples and shape features learned from ShapeNet objects. Such an approach requires no training on scene level data, where data is costly to acquire. We achieve significant improvement both visually and quantitatively in comparison to state-of-the-art reconstruction algorithms for the scene reconstruction from point samples task (Poisson Surface Reconstruction [54, 55], or PSR).

In summary, the main contributions of this work are:

- We propose the Local Implicit Grid representation for geometry, where we learn and leverage geometric features on a part level, and associated methods such as the overlapping latent grid mechanism and latent grid optimization methods for representing and reconstructing scenes at high fidelity.

- We illustrate the significantly improved generalizability of our part-based approach in comparison to related methods that learn priors for entire objects – i.e., we can reconstruct shapes from novel object classes after training only on chairs, or construct entire scenes after training only on ShapeNet parts.

- We apply our novel shape representation approach towards the challenging task of scene reconstruction from sparse point samples, and show significant improvement over the state-of-the-art approach (For Matterport reconstruction from $100/m^2$ input points, an F-Score of 0.8894 versus 0.4550).

4.2 Related Work

Geometric representation for objects

In computer vision and graphics, geometric representations such as simplicial complexes (point clouds, line meshes, triangular meshes, tetrahedral meshes) have long been used for representing geometries for its flexibility and compactness. In recent years, various neural architectures have been proposed for analyzing or generating such representations. For instance for [84, 107] have been proposed for analyzing point cloud representations, and [30, 112] for generating point clouds. [72, 36, 49, 42] have been proposed for analyzing signals on meshes, and [103, 34, 20] for generating mesh representations. [48] proposed a general framework for analyzing arbitrary simplicial complex based geometric signals. Naturally paired with 3D Convolutional Neural Networks (CNNs), voxel grids have also been extensively used as a 3D representation [110, 23, 15].

More recently, alternative representations have been proposed in the context of shape generation. Most related to our method are [74, 77, 14], where the implicit surfaces of geometries are represented as spatial functions using fully-connected neural networks. Continuous spatial coordinates are fed as input features to the network which directly produces the values of the implicit functions, however these methods encode the entire shape using a global latent code. [91] used such implicit networks to represent neural features instead of occupancies that can be combined with a differentiable ray marching algorithm to produce neural renderings of objects. Rather than learning a single global implicit network to represent the entire shape, [90] learns a continuous per-pixel occupancy and color representation using implicit networks. Other novel geometric representations in the context of shape reconstruction include Structured Implicit Functions that serves as learned local shape templates [32], and CvxNet [28] which represents space as a convex combination of half-planes that are localized in space. These methods represent entire shapes using a single global latent vector, which can be decoded into continuous outputs with the associated implicit networks.

Localized geometric representations

Though using a single global latent code to represent entire geometries and scenes is appealing for its simplicity, it fails to capture localized details, and scales poorly to large scenes with increased complexities. [111] proposes to address the localization problem in the context of image to 3D reconstruction by first estimating a camera pose for the images followed by the projection of local 2D features to be concatenated with global latents for decoding. However, the scalability of such hybrid representations beyond single objects has yet to be shown. Similar to our approach, [108] uses a local patch based representation. However it is not trained on any data, hence is not able to leverage any shape priors from 3d datasets. [78] combines shape patches extracted directly from a set of examples, which limits the shape expressibility. Similar to our spatial partitioning of geometries into part grids, [97] uses PCA-based decomposition to learn a reduced representation of geometric parts within TSDF grids of a fixed scale for the application of real-time geometry compression. These methods do not support scalable reconstruction with learned deep implicit functions.

Scene-level geometry reconstruction

Most deep learning studies have investigated object reconstruction, with input either as an RGB/D image [15, 103, 74, 14, 30, 28, 32] or 3D points [77, 66, 47], and yet few have considered learning to reconstruct full scenes. Scene level geometry reconstruction is a much more challenging task in comparison to single objects. [94] performs semantic scene completion within the frustum of a single depth image. [23] uses a 3D convolutional network with a coarse-to-fine inference strategy to directly regress gridded Truncated Signed Distance Function (TSDF) outputs from incomplete input TSDF. [4] tackles the scene reconstruction problem by CAD model retrieval, which produces attractive surfaces, at the expense of geometric inaccuracies. However, all of the methods require training on reliable and high-quality scene data. Though several real and synthetic scene datasets exist, such as SunCG [93], SceneNet [35], Matterport3D [12] and ScanNet [25], they are domain-specific and acquiring data for new scenes can be costly. In contrast to methods above that require training on scene dataset, our method naturally generalizes shape priors learned from object datasets and does not require additional training on scenes.

4.3 Methods

Method overview

We present a schematic overview of our method in Figure 4.1. We first learn an embedding of shape parts at a fixed scale from objects in a synthetic dataset using *part autoencoders* (see Sec. 4.3). We show two interesting properties of such a latent embedding: (1) objects that originated from different categories share similar part geometries, validating the generalizability of such learned representations, and (2) parts that are similar in shape are close in the latent space. In order to scale to scenes of arbitrary sizes, we introduce an overlapping gridded representation that can layout these local representations in a scene (Sec. 4.3). Using such part embeddings that can be continuously decoded spatially using a local implicit network, we are able to faithfully reconstruct geometries from only sparse oriented point samples by searching for a corresponding latent code using gradient descent-based optimization to match given observations (Sec. 4.3), thus efficiently leveraging geometric priors learned from parts from the ShapeNet dataset.

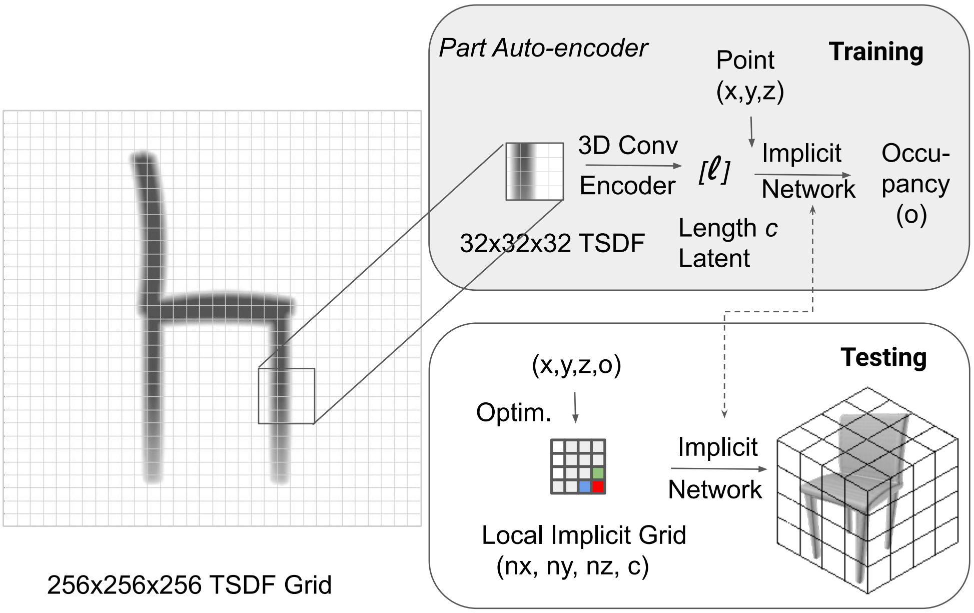


Figure 4.2: A schematic of the part autoencoder. At train time, crops of the TSDF grid from the ShapeNet dataset are used to train a part autoencoder, with a 3D CNN encoder and implicit network decoder. Interior and exterior points are sampled to supervise the network during training. At inference time, the pre-trained implicit network is attached to a Local Implicit Grid, and the corresponding latent values are optimized via gradient descent on observed interior/exterior points.

Learning a latent embedding for parts

Data Our part embedding model is learned from a collection of 20 million object parts culled from 3D-R^{2}N^2 [15], a 13-class subset of ShapeNet. As preprocessing, we normalize watertight meshes (generated with tools from [74]) into a $[0, 1]$ unit cube, leaving a margin of 0.1 at each side. To maintain the fidelity of the parts, we compute a signed distance function (SDF) at a grid resolution of 256^3. Starting from the origin and with a stride of 16, all 32^3 patches that have at least one point within $3/255$ of the shape surface are extracted as parts for training.

Part Autoencoder We use a 3D CNN decorated with residual blocks for encoding such local TSDF grids, and a reduced IM-NET [14] decoder for reconstructing the part. An IM-NET decoder is a simple fully connected neural network with internal skip connections that takes in a latent code concatenated with a 3D point coordinate, and outputs the corresponding implicit function value at the point. We train the network using point samples with binary in/out labels so that the network learns a continuous decision boundary of the binary classifier as the encoded surface.

Since decoding a part is a much more simplified task than decoding an entire shape, we reduce the number of feature channels in each hidden layer of IM-NET by 4 fold, obtaining a leaner and more efficient decoder. To acquire a compact latent representation of parts, we further reduce the number of latent channels for each part to 32. We train the part autoencoder with 2048 random point samples that we sample from the SDF grid on-the-fly during training, where we sample points farther from the boundary with Gaussian-decaying probabilities. The sign of the sample points is interpolated from the sign of the original SDF grid. Furthermore, we truncate the input SDF grids to a value of $3/255$ and renormalize the grid to $[0, 1]$ for stronger gradients near the boundary.

We train the part autoencoder with binary cross entropy loss on the point samples, with an additional latent regularization loss to constrain the latent space of the learned embeddings. The loss is given as:

$$\begin{aligned}\mathcal{L}(\theta_e, \theta_d) =& \frac{1}{|\mathcal{P}||\mathcal{B}|}\sum_{i\in\mathcal{P}}\sum_{j\in\mathcal{B}}\mathcal{L}_c(D_{\theta_d}(\boldsymbol{x}_{i,j}, E_{\theta_e}(g_i)), \text{sign}(\boldsymbol{x}_{i,j})) \\ &+ \lambda||E_{\theta_e}(g_i)||_2\end{aligned} \tag{4.1}$$

where $\mathcal{P}$ is the set of all training parts in a given mini-batch, $\mathcal{B}$ is the set of point samples sampled per part, $\mathcal{L}_c(\cdot, \cdot)$ is the binary cross-entropy loss with logits, E_{θ_e} is the convolutional encoder parameterized by trainable parameters θ_e, D_{θ_d} is the implicit decoder parameterized by trainable parameters θ_d, and g_i is the input tsdf grid for the i-th part, sign$(\cdot)$ takes the sign of the corresponding point $x_{i,j}$.

Local implicit grids

In order to use the learned part representations for representing entire objects and scenes, we lay out a sparse latent grid structure, where within each local grid cell the surface is continuously decoded from the local latent codes within the cell. In world coordinates, when querying for the implicit function value at location $\boldsymbol{x}$ against a single voxel grid cell centered at $\boldsymbol{x_i}$, the implicit value is decoded as:

$$f(\boldsymbol{x}, \boldsymbol{c}_i) = D_{\theta_d}(\boldsymbol{c}_i, \frac{2}{s}(\boldsymbol{x} - \boldsymbol{x}_i)) \tag{4.2}$$

where $\boldsymbol{c}_i$ is the latent code corresponding to the part in cell i, and s is the part scale. The coordinates are first being transformed into normalized local coordinates within the cell to $[-1, 1]$, before being queried against the decoder.

Though directly partitioning space into a voxel grid with latent channels within each cell gives decent performance, there will be discontinuities across voxel boundaries. Hence we propose the overlapping latent grid scheme, where each grid cell for a part overlaps with its neighboring cells by half the part scale. When querying for the implicit function value at an arbitrary position $\boldsymbol{x}$ against overlapping latent grids, the value is computed as a trilinear interpolation of independent queries to all cells that overlap at this position, which is 4 in 2 dimensions and 8 in 3 dimensions:

$$f(\boldsymbol{x}, \{\boldsymbol{c}_j | j \in \mathcal{N}\}) = \sum_{j\in\mathcal{N}} w_j D_{\theta_d}(\boldsymbol{c}_j, \frac{2}{s}(\boldsymbol{x} - \boldsymbol{x}_j)) \tag{4.3}$$

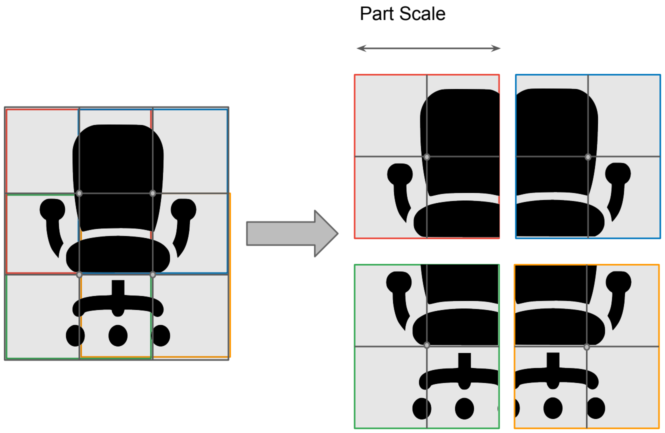


Figure 4.3: 2D schematic for representing geometries with overlapping latent grids. The implicit value at any point is an bilinear/trilinear interpolation of implicit values acquired by querying 4/8 (2D/3D) neighbors with respect to each cell center.

where $\mathcal{N}_j$ is the set of all neighboring cells of point $\boldsymbol{x}$, and w_j is the trilinear interpolation weight corresponding to cell j. Under such an interpolation scheme, the overall function represented by the implicit grid is guaranteed to be C^0 continuous. Higher-order continuity could be similarly acquired with higher degrees of polynomial interpolations, though we do not explore it in the scope of this study. For additional efficiency, since most grid cells do not have any points that fall into them, we use a sparse data structure for storing latent grid values, optimization, and decoding for the reconstructed surface, where empty space is assumed to be exterior space.

Geometric encoding via latent optimization

At inference time, when presented with a sparse point cloud of interior/exterior samples as input, we decompose space into a coarse grid and then perform optimization for the latent vectors associated with the grid cells in order to minimize the cost function for classifying sampled interior/exterior points. The initial values within the latent grid is initialized as random normal with a standard deviation of 10^{-2}. If we denote the set of effective latent grid cells as $\mathcal{G}$, the corresponding latent code in each grid cell c_j, and the set of all sampled interior/exterior input points as $\mathcal{B}$, we optimize the latent codes for the minimal classification loss on the sampled points:

$$\operatorname*{arg\,min}_{\boldsymbol{c}\in\mathcal{G}} \sum_{i\in\mathcal{B}} \sum_{j\in\mathcal{N}_i} \mathcal{L}_c(f(\boldsymbol{x_i}, \{\boldsymbol{c}_j | j \in \mathcal{N}\}), \text{sign}(\boldsymbol{x}_i)) + \lambda||\boldsymbol{c}_j||_2 \tag{4.4}$$

How do we acquire the signed point samples for performing this latent grid optimization? For autoencoding a geometry with a latent grid, the signed point samples are densely sampled

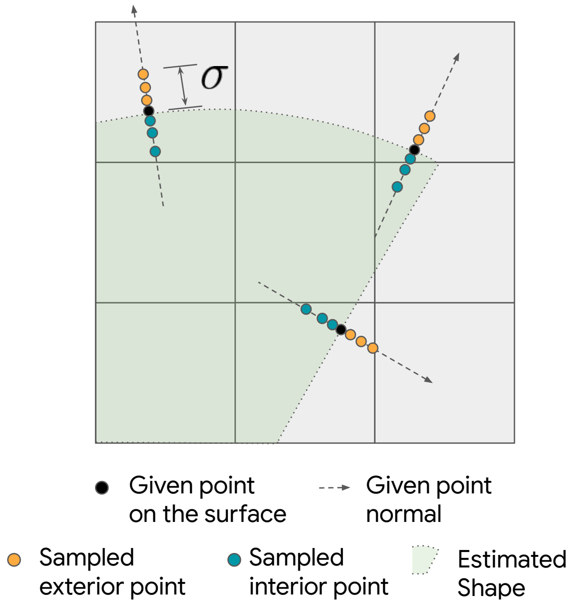


Figure 4.4: Schematic for reconstructing shapes based on sparse oriented point samples. Given original point samples on the surface with normals, we randomly sample k samples along both sides of each normal vector and assign signs for these samples accordingly. The points are sampled with a Gaussian falloff probability, with a given standard deviation σ. The latent codes within the overlapping latent grids are updated via optimization for minimizing classification loss as in Eqn. 4.4. The surface of the shape is reconstructed by densely querying the latent grid and extracting the zero-contour of the output logits.

near the surface of the given shape to be encoded. However, for the application of recovering surface geometry from sparse oriented point samples, we randomly sample interior and exterior points for each point sample along the given normal direction, with a Gaussian falloff probability parameterized by a standard deviation of σ. See Fig. 4.4 for details.

As our method requires optimizing over the learned latent space, it is reasonable to wonder if alternate models such as a variational autoencoder [57] or autodecoder [77] would be a more appropriate choice, as both formulations incorporate a latent distribution prior. However, [77] observed the stochastic nature of the VAE made training difficult. Also, the autodecoder is fundamentally unable to scale to large numbers of parts at training as it requires fast storage and random access to all latent embeddings during training. These concerns motivated our decision to adopt an autoencoder formulation with a regularization loss to constrain the latent space.

4.4 Experiments

We ran a series of experiments to test the proposed LIG method. We focus on two properties of our method: the generalization of our learned part representation, and the scalability of our learned

Category	IM-NET			Ours		
	CD (↓)	Normal (↑)	F-Score (↑)	CD (↓)	Normal (↑)	F-Score (↑)
chair	0.1813	0.819	0.5054	**0.0988**	**0.917**	**0.7095**
airplane	0.6975	0.5503	0.1509	0.1503	0.8174	0.5641
bench	0.2294	0.7189	0.4327	0.0543	0.9048	0.8569
cabinet	0.3428	0.7004	0.2302	0.1182	0.9479	0.7331
car	0.3543	0.6456	0.238	0.1518	0.8251	0.4723
display	0.6013	0.5738	0.1302	0.1703	0.9259	0.5514
lamp	0.8356	0.5923	0.119	0.1141	0.8819	0.624
loudspeaker	0.3772	0.7017	0.2462	0.1385	0.9369	0.7114
rifle	0.9017	0.4002	0.0798	0.1126	0.8242	0.6932
sofa	0.199	0.8116	0.484	0.0767	0.9435	0.8218
table	0.4253	0.681	0.2424	0.0659	0.9363	0.8443
telephone	0.6232	0.5471	0.12	0.0371	0.9842	0.9621
vessel	0.5914	0.5743	0.1473	0.1782	0.8474	0.4667
mean*	0.435	0.6663	0.2738	**0.1140**	**0.8980**	**0.6918**

Table 4.1: Shape autoencoding for autoencoders trained on only chairs and evaluated on all 13 categories. The mean corresponds to class-averaged mean of all out-of-training object categories.

shape representation to large scenes. Our target application is reconstructing scenes from a sparse set of oriented point samples, a challenging task that requires learned part priors for detailed and accurate reconstruction.

Metrics In all of our experiments, we evaluate geometric reconstruction quality with Chamfer Distance (CD), Normal Alignment (Normal), and F-Score. For Chamfer Distance and Normal Alignment, we base our implementation on [74] with small differences. For object-level autoencoding experiments, we follow [30, 74] and normalize the unit distance to be 1/10 of the maximal edge length of the current object's bounding box. We estimate CD and Normal Alignment using 100,000 randomly sampled points on the ground truth and reconstructed meshes. For the two scene-level experiments, we randomly sample 2 million points on each mesh when estimating CD and Normal Alignment. When evaluating scene reconstructions, we use world coordinate scales (meters) for computing CD, since data is provided in a physically-meaningful scale. Additionally, in all experiments, we compute the F-Score at a threshold of τ, as F-Score is a metric less sensitive to outliers. F-Score is the harmonic mean of recall (percentage of reconstruction to target distances under τ) and precision (vice versa). For object reconstruction (Sec. 4.3) we use $\tau = 0.1$ and for scene reconstruction, we use $\tau = 0.025$ (i.e., 2.5cm).

Metrics	CD(↓)	Normal(↑)	F-Score(↑)
IM-NET	0.1825	0.8267	0.6472
Ours	**0.007**	**0.9451**	**0.9853**

Table 4.2: Qualitative comparison of scene representational performance for IM-NET versus our method.

Generalization of learned part representation

Task In order to investigate the generalization of the learned embedding by reducing the scale of the learned shape from object scale to part scale, we construct an investigative experiment of training the models to learn a shape autoencoder on a single category of objects (in this case, chairs in the training set of ShapeNet), and reconstructing examples from the all 13 object categories, including the other 12 unseen categories.

Baseline As our main objective is to explore the gain in generalizability from learning an embedding of part scales, we benchmark our method against the original IM-NET decoder with a similar 3D convolution based encoder as the encoder part of our part autoencoder. To implement autoencoding for our method, we train our autoencoder on all the parts we extract from the training split of the chair category in ShapeNet. We then "encode" the geometries of the unseen shapes using the latent optimization method that is described in Sec. 4.3.

Results Discussion We quantitatively and qualitatively compare reconstruction performances in Table 4.1 and Figure 4.5, respectively. Given an IM-NET that is trained to learn a latent representation of objects (in this scenario, chairs), the learned representation does not generalize to classes beyond the source class. Visually, IM-NET achieves good reconstructions on the source class as well as related classes (e.g., sofa), but performs poorly on semantically different classes (e.g., airplane). In contrast, the part representation learned by our local implicit networks is transferable across drastically different object categories.

Scalability of scene representational power

Task As a second experiment, we investigate the increased representational power and scalability that we gain from learning a part-based shape embedding. The definition of the task is: given one scene, what is the best reconstruction performance we can get from either representation for memorizing and overfitting to the scene.

Baseline Similar to the previous experiment, we compare directly with IM-NET for representational capacity towards a scene, as it is the decoder backbone that our method is based on, to investigate the improvement in scalability that we are able to gain by distributing geometric information in spatially localized grid cells versus a single global representation. For this task, as the

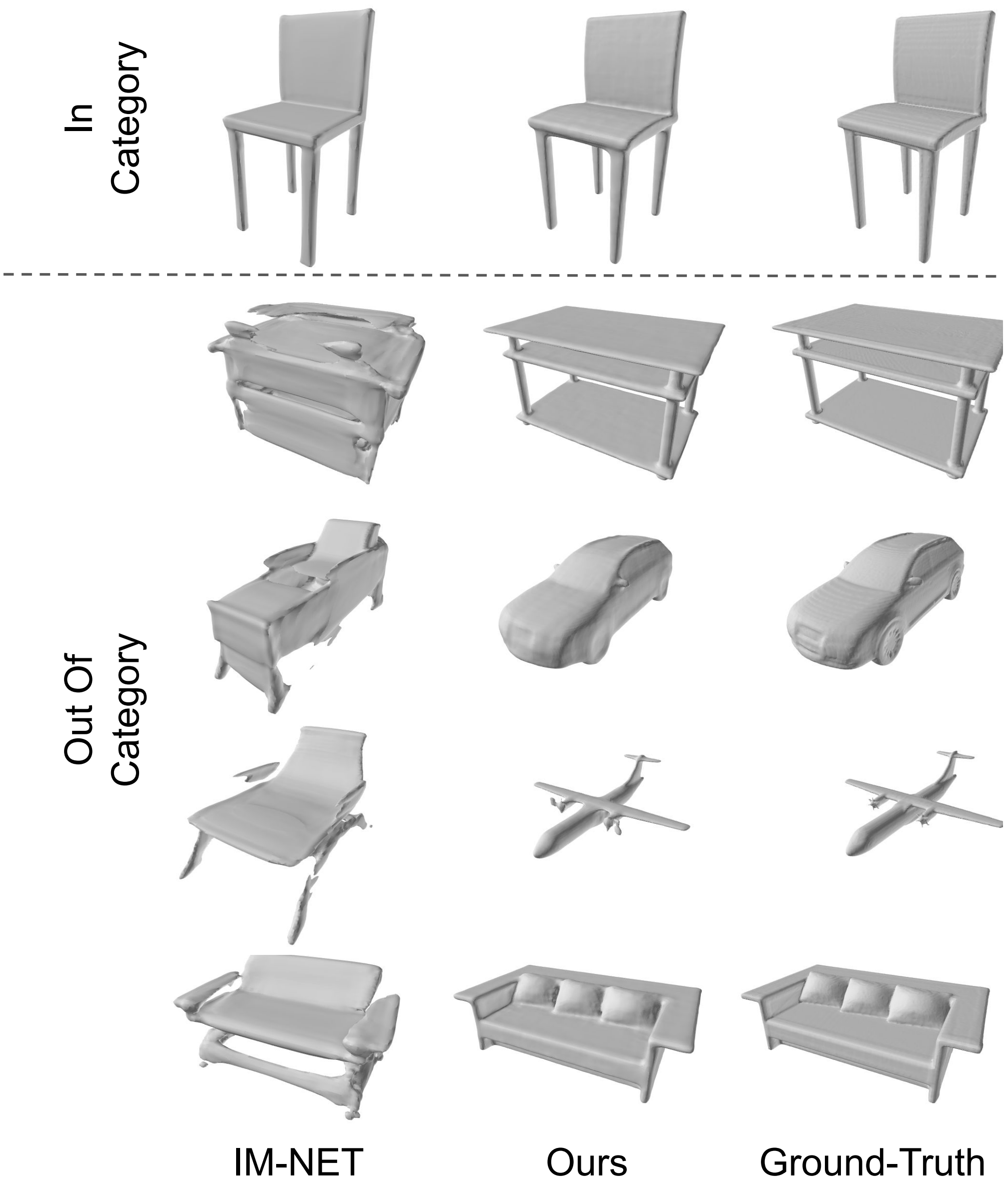


Figure 4.5: Qualitative comparison of autoencoded shape from in-category (chair) and out-of-category shapes. IM-NET trained to learn embeddings of one object category does not transfer well to unseen categories, while the part embedding learned by our local implicit networks is much more transferable across unseen categories.

objective is to encode one scene, we use the encoderless version of IM-NET, where during training time, the decoder only receives spatial coordinates of point samples (not concatenated with a latent code) that are paired with the signs of these points. For our method, we use latent optimization

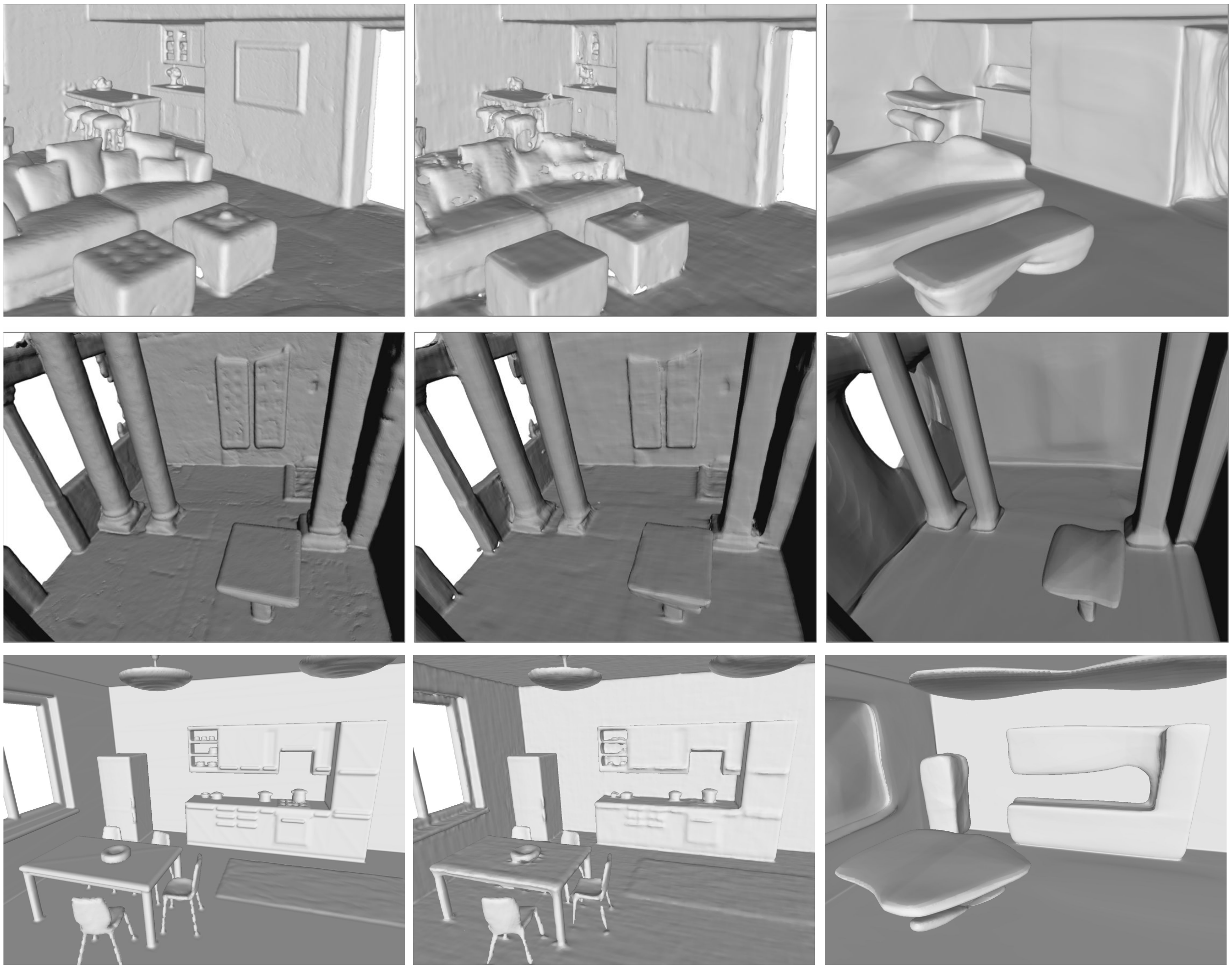


Figure 4.6: Qualitative comparison of the scene representational performance: Left to right: Ground truth scene, our reconstruction using sampling density 500 points/m^2, and IM-NET. First two rows from Matterport, last row from SceneNet.

against the pretrained decoder for encoding the scenes, using 100k surface point samples from the scene, with a sampling factor of $k = 10$ per point along the normal direction.

Data We evaluate the representational qualities of the two methods on the meshes from the validation set of the Matterport 3D [12] scene dataset. We perform the evaluations at the region level of the dataset, requiring the models to encode one region at a time. Additionally, we provide one example from SceneNet for visual comparison in Fig. 4.6.

Results Discussion The quantitative (Table 4.2) and qualitative (Fig. 4.6) results are presented. While IM-NET is able to reconstruct the general structure of indoor scenes such as smooth walls and floors, it fails to capture fine details of objects due to the difficulty of scaling a single implicit

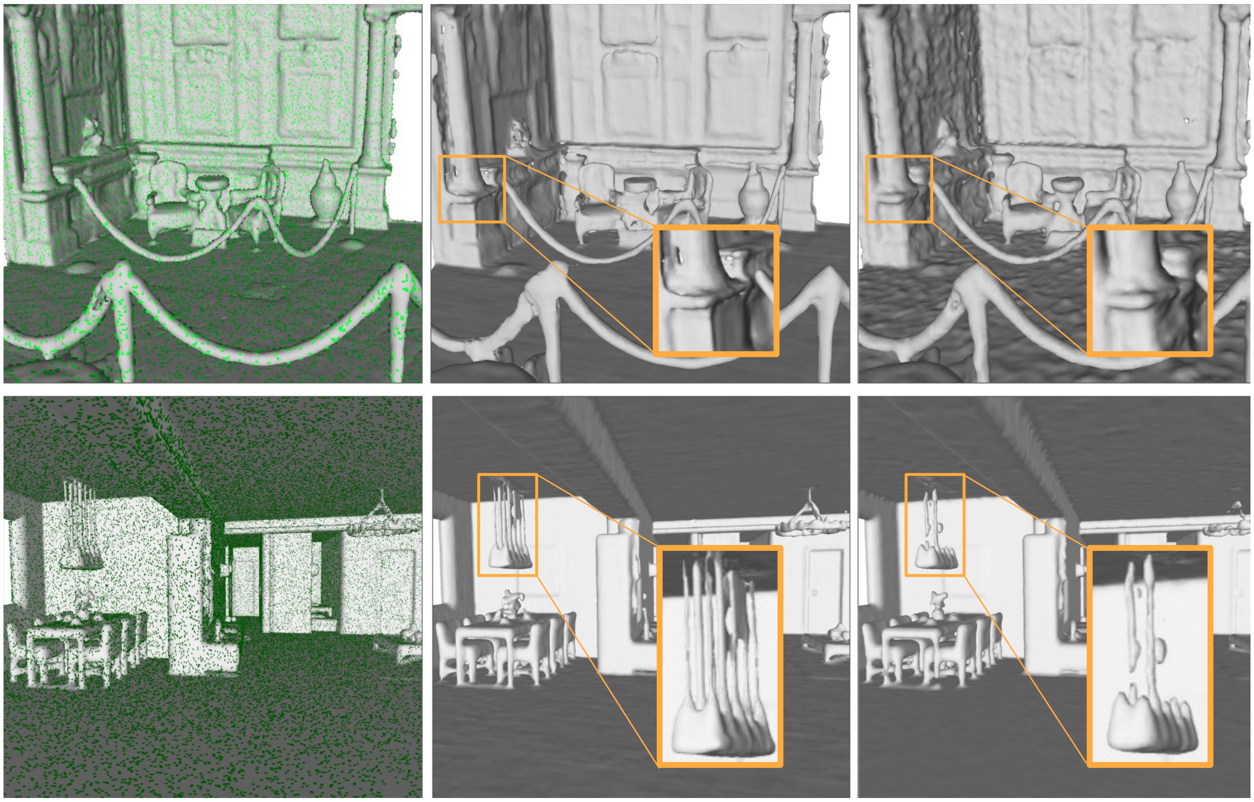


Figure 4.7: Qualitative comparisons of scene reconstruction performance from sparse oriented point samples. Left: Ground truth scene mesh, with input point cloud overlay. Middle: Our reconstruction. Right: Reconstruction using PSR10. Top row: Scene from Matterport 3D dataset. Bottom row: SceneNet dataset. Our method is significantly better at reconstructing sharp edges and thin structures.

network to an entire scene. Our Local Implicit Grids are able to capture global structures as well as local details.

Scene reconstruction from sparse oriented points

Task As a final task and our main application, we apply our reconstruction method to the classic task in computer graphics to reconstruct geometries from sparse points. This is an important application since surface reconstruction from points is a crucial step in the process of digitizing the 3-dimensional world. The input to the reconstruction pipeline is the sparse point samples that we randomly sample from the surface mesh of the scene datasets. We study reconstruction performances with a varied number of input point samples and point densities.

Baseline We compare our method to the traditional Poisson Surface Reconstruction (PSR) method [54, 55] with a high octree depth value (depth=10) for the scene reconstruction experiment, which remains the state-of-the-art method for surface reconstruction tasks. For one scenario (see Table 4.3 we also compare with PSR at depths 8 and 9, which produces similar results to PSR-10). While various deep learning based methods [77, 66, 47] have been proposed for surface reconstruction from points in a similar setting, all of the deep learning based methods are object-specific, trained and tested on specific object categories in ShapeNet, with no anticipated transferability to unseen categories or scenes, as we have shown in the experiment in Sec. 4.4. Furthermore, as both PSR and our method require no training/finetuning on the scene level datasets, the task is based on the premise that high-quality 3D training data is costly to acquire or unavailable for scenes. For our method, we adaptively use different part sizes for different point densities. We use 25cm (1000 pts/m^2), 35cm (500 pts/m^2), 50cm (100 pts/m^2) and 75cm (20 pts/m^2) corresponding to different point densities for optimal performance.

Data We evaluate the reconstruction performance of the methods on a synthetic dataset: SceneNet [35], and a high quality scanned dataset: Matterport 3D [12] (validation split). As both SceneNet and Matterport 3D datasets are not watertight, and in addition to that, SceneNet dataset has various artifacts such as double-sided faces that produce conflicting normal samples, we preprocess both datasets using the watertight manifold algorithm as describe in [40]. For both datasets, as the scenes vary in sizes, we sample a constant density of points on mesh surfaces (20, 100, 500 and 1000 points per m^2). As preprocessing produces large empty volumes for SceneNet, we drop scenes that have a volume-to-surface-area ratio lower than 0.13.

Results Discussion We compare the reconstruction performances in Table 4.3 and 4.4, and Fig. 4.7. With a high number of input point samples, both PSR10 and our method are able to reconstruct the original scene with high fidelity. However, with a low number of point samples, our method is able to leverage geometric priors to perform a much better reconstruction than PSR. Additionally, our method is able to reconstruct thin structures very well whereas PSR fails to do so. However, since our method only reconstructs finite thickness surfaces as determined by finite part size, it creates double sided surfaces on the enclosed non-visible interiors, leading to degraded performance in F-Score for the 500 and 1000 pts/m^2 scenarios in Table 4.3.

4.5 Ablation Study

Additionally, we study the effects of two important aspects of our method: the part scale that we choose for reconstructing each scene, and overlapping latent grids. We choose SceneNet reconstruction from 100 point samples / m^2 as a representative case for the ablation study. See Table 4.5 for a comparison. As seen from the results, the reconstruction results are affected by the choice of part scale, albeit not very heavily influenced. Overlapping latent grids significantly improves the quality of the overall reconstruction. With a smaller latent code size of 8, the performance is slightly deteriorated due to more limited expressivity for part geometries.

points/m^2	Method	CD(↓)	Normal(↑)	F-Score(↑)
20	PSR10	0.0768	0.8023	0.3172
	Ours	**0.0181**	**0.9194**	**0.8540**
100	PSR8	0.0310	0.8913	0.7207
	PSR9	0.0353	0.8899	0.7205
	PSR10	0.0354	0.8896	0.7250
	Ours	**0.0132**	**0.9594**	**0.9356**
500	PSR10	0.0235	0.9587	**0.9570**
	Ours	**0.0130**	**0.9742**	0.9258
1000	PSR10	0.0258	0.9752	**0.9837**
	Ours	**0.0132**	**0.9774**	0.8961

Table 4.3: Reconstruction performance on SceneNet dataset.

points/m^2	Method	CD(↓)	Normal(↑)	F-Score(↑)
20	PSR10	0.1665	0.6549	0.2759
	Ours	**0.0283**	**0.8134**	**0.6913**
100	PSR10	0.1064	0.7567	0.4550
	Ours	**0.0134**	**0.8829**	**0.8894**
500	PSR10	0.1029	0.8710	0.7776
	Ours	**0.0083**	**0.9279**	**0.9703**
1000	PSR10	0.1015	0.9101	0.8619
	Ours	**0.0070**	**0.9451**	**0.9853**

Table 4.4: Reconstruction performance on Matterport dataset.

CL	PS	Overlap	CD(↓)	Normal(↑)	F-Score(↑)
32	25cm	Yes	0.0142	0.9484	0.8713
32	50cm	Yes	0.0132	0.9594	0.9356
32	75cm	Yes	0.0139	0.9440	0.9212
32	50cm	No	0.0247	0.8853	0.8423
8	50cm	Yes	0.0181	0.9238	0.8516

Table 4.5: Ablation study on the effects of the choice of latent code length (CL), part scale (PS) and overlapping latent grid design on the reconstruction performance for scenes.

4.6 Discussion and Future Work

The Local Implicit Grid (LIG) representation for 3D scenes is a regular grid of overlapping part-sized local regions, each encoded with an implicit feature vector. Experiments show that LIG is capable of reconstructing 3D surfaces of objects from classes unseen in training. Furthermore, to our knowledge, it is the first learned 3D representation for reconstructing scenes from sparse point sets in a scalable manner. Topics for future work include ways to constrain the LIG optimization to produce latent codes near training examples, explore alternate implicit function representations (eg. OccNet), and to investigate the best ways to use LIG for 3D reconstruction from image(s).

Bibliography

[1] David Acuna et al. "Efficient interactive annotation of segmentation datasets with polygon-rnn++". In: *arXiv preprint arXiv:1803.09693* (2018).

[2] Iro Armeni et al. "Joint 2d-3d-semantic data for indoor scene understanding". In: *arXiv preprint arXiv:1702.01105* (2017).

[3] James Atwood and Don Towsley. "Diffusion-convolutional neural networks". In: *Advances in Neural Information Processing Systems*. 2016, pp. 1993–2001.

[4] Armen Avetisyan et al. "Scan2CAD: Learning CAD model alignment in RGB-D scans". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2614–2623.

[5] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "Segnet: A deep convolutional encoder-decoder architecture for image segmentation". In: *arXiv preprint arXiv:1511.00561* (2015).

[6] John R Baumgardner and Paul O Frederickson. "Icosahedral discretization of the two-sphere". In: *SIAM Journal on Numerical Analysis* 22.6 (1985), pp. 1107–1115.

[7] Davide Boscaini et al. "Learning shape correspondence with anisotropic convolutional neural networks". In: *Advances in Neural Information Processing Systems*. 2016, pp. 3189–3197.

[8] Michael M Bronstein et al. "Geometric deep learning: going beyond euclidean data". In: *IEEE Signal Processing Magazine* 34.4 (2017), pp. 18–42.

[9] Joan Bruna et al. "Spectral networks and locally connected networks on graphs". English (US). In: *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*. 2014.

[10] Lluis Castrejon et al. "Annotating object instances with a polygon-rnn". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 5230–5238.

[11] Angel X Chang et al. "Shapenet: An information-rich 3d model repository". In: *arXiv preprint arXiv:1512.03012* (2015).

[12] Angel Chang et al. "Matterport3D: Learning from RGB-D Data in Indoor Environments". In: *International Conference on 3D Vision (3DV)* (2017).

[13] Angel Chang et al. "Matterport3d: Learning from rgb-d data in indoor environments". In: *arXiv preprint arXiv:1709.06158* (2017).

[14] Zhiqin Chen and Hao Zhang. "Learning implicit fields for generative shape modeling". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 5939–5948.

[15] Christopher B Choy et al. "3d-r2n2: A unified approach for single and multi-view 3d object reconstruction". In: *European conference on computer vision*. Springer. 2016, pp. 628–644.

[16] Taco S. Cohen et al. "Spherical CNNs". In: *International Conference on Learning Representations*. 2018. URL: `https://openreview.net/forum?id=Hkbd5xZRb`.

[17] Benjamin Coors, Alexandru Paul Condurache, and Andreas Geiger. "SphereNet: Learning Spherical Representations for Detection and Classification in Omnidirectional Images". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 518–533.

[18] Marius Cordts et al. "The cityscapes dataset for semantic urban scene understanding". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 3213–3223.

[19] Angela Dai and Matthias Nießner. "3DMV: Joint 3D-Multi-View Prediction for 3D Semantic Scene Segmentation". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018.

[20] Angela Dai and Matthias Nießner. "Scan2Mesh: From Unstructured Range Scans to 3D Meshes". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 5574–5583.

[21] Angela Dai, Charles Ruizhongtai Qi, and Matthias Nießner. "Shape Completion using 3D-Encoder-Predictor CNNs and Shape Synthesis". In: *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*. 2017.

[22] Angela Dai, Charles Ruizhongtai Qi, and Matthias Nießner. "Shape completion using 3d-encoder-predictor cnns and shape synthesis". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 5868–5877.

[23] Angela Dai et al. "ScanComplete: Large-Scale Scene Completion and Semantic Segmentation for 3D Scans". In: *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*. 2018.

[24] Angela Dai et al. "Scancomplete: Large-scale scene completion and semantic segmentation for 3d scans". In: *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[25] Angela Dai et al. "ScanNet: Richly-Annotated 3D Reconstructions of Indoor Scenes." In: *CVPR*. Vol. 2. 2017, p. 10.

[26] Jifeng Dai, Kaiming He, and Jian Sun. “Instance-aware semantic segmentation via multi-task network cascades”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 3150–3158.

[27] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. “Convolutional neural networks on graphs with fast localized spectral filtering”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 3844–3852.

[28] Boyang Deng et al. “CvxNets: Learnable Convex Decomposition”. In: *arXiv preprint arXiv:1909.05736* (2019).

[29] Carlos Esteves et al. “Learning SO (3) Equivariant Representations with Spherical CNNs”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 52–68.

[30] Haoqiang Fan, Hao Su, and Leonidas J Guibas. “A point set generation network for 3d object reconstruction from a single image”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 605–613.

[31] Andreas Geiger et al. “Vision meets Robotics: The KITTI Dataset”. In: *International Journal of Robotics Research (IJRR)* (2013).

[32] Kyle Genova et al. “Learning Shape Templates with Structured Implicit Functions”. In: *arXiv preprint arXiv:1904.06447* (2019).

[33] Kyle Genova et al. “Unsupervised Training for 3D Morphable Model Regression”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8377–8386.

[34] Thibault Groueix et al. “AtlasNet: A Papier-Mache Approach to Learning 3D Surface Generation”. In: *arXiv preprint arXiv:1802.05384* (2018).

[35] Ankur Handa et al. “Understanding real world indoor scenes with synthetic data”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 4077–4085.

[36] Rana Hanocka et al. “MeshCNN: a network with an edge”. In: *ACM Transactions on Graphics (TOG)* 38.4 (2019), pp. 1–12.

[37] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[38] Kaiming He et al. “Mask r-cnn”. In: *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE. 2017, pp. 2980–2988.

[39] Oliver Hennigh. “Automated Design using Neural Networks and Gradient Descent”. In: *arXiv preprint arXiv:1710.10352* (2017).

[40] Jingwei Huang, Hao Su, and Leonidas Guibas. “Robust Watertight Manifold Surface Generation Method for ShapeNet Models”. In: *arXiv preprint arXiv:1802.01698* (2018).

[41] Jingwei Huang et al. “TextureNet: Consistent Local Parametrizations for Learning from High-Resolution Signals on Meshes”. In: *arXiv preprint arXiv:1812.00020* (2018).

[42] Jingwei Huang et al. “Texturenet: Consistent local parametrizations for learning from high-resolution signals on meshes”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4440–4449.

[43] Antony Jameson, L Martinelli, and NA Pierce. “Optimum aerodynamic design using the Navier–Stokes equations”. In: *Theoretical and computational fluid dynamics* 10.1-4 (1998), pp. 213–237.

[44] Simon Jégou et al. “The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation”. In: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*. IEEE. 2017, pp. 1175–1183.

[45] Chiyu Max Jiang et al. “Convolutional Neural Networks on Non-uniform Geometrical Signals Using Euclidean Spectral Transformation”. In: *International Conference on Learning Representations*. 2019. URL: `https://openreview.net/forum?id=B1G5ViAqFm`.

[46] Chiyu Max Jiang et al. “Spherical CNNs on Unstructured Grids”. In: *International Conference on Learning Representations*. 2019. URL: `https://openreview.net/forum?id=Bkl-43C9FQ`.

[47] Chiyu Jiang et al. “Convolutional Neural Networks on non-uniform geometrical signals using Euclidean spectral transformation”. In: *arXiv preprint arXiv:1901.02070* (2019).

[48] Chiyu Jiang et al. “DDSL: Deep Differentiable Simplex Layer for Learning Geometric Signals”. In: *arXiv preprint arXiv:1901.11082* (2019).

[49] Chiyu Jiang et al. “Spherical CNNs on unstructured grids”. In: *International Conference on Learning Representations*. 2019.

[50] Evangelos Kalogerakis et al. “3D shape segmentation with projective convolutional networks”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2017, pp. 6630–6639.

[51] Angjoo Kanazawa et al. “Learning Category-Specific Mesh Reconstruction from Image Collections”. In: *arXiv preprint arXiv:1803.07549* (2018).

[52] Asako Kanezaki, Yasuyuki Matsushita, and Yoshifumi Nishida. “Rotationnet: Joint object categorization and pose estimation using multiviews from unsupervised viewpoints”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 5010–5019.

[53] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. “Neural 3d mesh renderer”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3907–3916.

[54] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. “Poisson surface reconstruction”. In: *Proceedings of the fourth Eurographics symposium on Geometry processing*. Vol. 7. 2006.

[55] Michael Kazhdan and Hugues Hoppe. “Screened poisson surface reconstruction”. In: *ACM Transactions on Graphics (ToG)* 32.3 (2013), p. 29.

[56] Manas Khurana, Hadi Winarto, and Arvind Sinha. “Airfoil optimisation by swarm algorithm with mutation and artificial neural networks”. In: *47th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*. 2009, p. 1278.

[57] Diederik P. Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: *Proc. ICLR*. 2014.

[58] Thomas N. Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *International Conference on Learning Representations (ICLR)*. 2017.

[59] Thomas N Kipf and Max Welling. “Semi-supervised classification with graph convolutional networks”. In: *arXiv preprint arXiv:1609.02907* (2016).

[60] Risi Imre Kondor and John Lafferty. “Diffusion kernels on graphs and other discrete structures”. In: *Proceedings of the 19th international conference on machine learning*. Vol. 2002. 2002, pp. 315–322.

[61] Abhijit Kundu, Yin Li, and James M Rehg. “3D-RCNN: Instance-Level 3D Object Reconstruction via Render-and-Compare”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3559–3568.

[62] Yann LeCun. “The MNIST database of handwritten digits”. In: *http://yann. lecun. com/exdb/mnist/* (1998).

[63] Chun-Liang Li et al. “Point cloud gan”. In: *arXiv preprint arXiv:1810.05795* (2018).

[64] Tzu-Mao Li et al. “Differentiable monte carlo ray tracing through edge sampling”. In: *SIGGRAPH Asia 2018 Technical Papers*. ACM. 2018, p. 222.

[65] Yi Li et al. “Fully convolutional instance-aware semantic segmentation”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2017, pp. 4438–4446.

[66] Yiyi Liao, Simon Donne, and Andreas Geiger. “Deep marching cubes: Learning explicit surface representations”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2916–2925.

[67] Shichen Liu et al. “Soft Rasterizer: Differentiable Rendering for Unsupervised Single-View Mesh Reconstruction”. In: *arXiv preprint arXiv:1901.05567* (2019).

[68] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

[69] Matthew M Loper and Michael J Black. “OpenDR: An approximate differentiable renderer”. In: *European Conference on Computer Vision*. Springer. 2014, pp. 154–169.

[70] Anton Lundberg et al. “Automated aerodynamic vehicle shape optimization using neural networks and evolutionary optimization”. In: *SAE International Journal of Passenger Cars-Mechanical Systems* 8.2015-01-1548 (2015), pp. 242–251.

[71] Haggai Maron et al. “Convolutional neural networks on surfaces via seamless toric covers”. In: *ACM Trans. Graph* 36.4 (2017), p. 71.

[72] Jonathan Masci et al. “Geodesic convolutional neural networks on riemannian manifolds”. In: *Proceedings of the IEEE international conference on computer vision workshops*. 2015, pp. 37–45.

[73] Daniel Maturana and Sebastian Scherer. “Voxnet: A 3d convolutional neural network for real-time object recognition”. In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE. 2015, pp. 922–928.

[74] Lars Mescheder et al. “Occupancy networks: Learning 3d reconstruction in function space”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4460–4470.

[75] Mayur Mudigonda et al. “Segmenting and Tracking Extreme Climate Events using Neural Networks”. In: *First Workshp Deep Learning for Physical Sciences*. Neural Information Processing Systems (NIPS), 2017.

[76] Richard B Neale et al. “Description of the NCAR community atmosphere model (CAM 5.0)”. In: *NCAR Tech. Note NCAR/TN-486+ STR* 1.1 (2010), pp. 1–12.

[77] Jeong Joon Park et al. “DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 165–174.

[78] Mark Pauly et al. “Example-based 3D scan completion”. In: *Symposium on Geometry Processing*. CONF. 2005, pp. 23–32.

[79] Pedro O Pinheiro, Ronan Collobert, and Piotr Dollár. “Learning to segment object candidates”. In: *Advances in Neural Information Processing Systems*. 2015, pp. 1990–1998.

[80] Pedro O Pinheiro et al. “Learning to refine object segments”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 75–91.

[81] Olivier Pironneau. “On optimum design in fluid mechanics”. In: *Journal of Fluid Mechanics* 64.1 (1974), pp. 97–110.

[82] Charles Ruizhongtai Qi et al. “Pointnet++: Deep hierarchical feature learning on point sets in a metric space”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5099–5108.

[83] Charles R Qi et al. “Deep hough voting for 3d object detection in point clouds”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 9277–9286.

[84] Charles R Qi et al. “Pointnet: Deep learning on point sets for 3d classification and segmentation”. In: *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE* 1.2 (2017), p. 4.

[85] Charles R Qi et al. “Volumetric and multi-view cnns for object classification on 3d data”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 5648–5656.

[86] Evan Racah et al. “ExtremeWeather: A large-scale climate dataset for semi-supervised detection, localization, and understanding of extreme weather events”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 3402–3413.

[87] Elad Richardson et al. “Learning detailed face reconstruction from a single image”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2017, pp. 5553–5562.

[88] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[89] Lars Ruthotto and Eldad Haber. “Deep Neural Networks motivated by Partial Differential Equations”. In: *arXiv preprint arXiv:1804.04272* (2018).

[90] Shunsuke Saito et al. “PIFu: Pixel-Aligned Implicit Function for High-Resolution Clothed Human Digitization”. In: *arXiv preprint arXiv:1905.05172* (2019).

[91] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. “Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations”. In: *arXiv preprint arXiv:1906.01618* (2019).

[92] Shuran Song et al. “Im2Pano3D: Extrapolating 360 Structure and Semantics Beyond the Field of View”. In: *arXiv preprint arXiv:1712.04569* (2017).

[93] Shuran Song et al. “Semantic Scene Completion from a Single Depth Image”. In: *Proceedings of 30th IEEE Conference on Computer Vision and Pattern Recognition* (2017).

[94] Shuran Song et al. “Semantic scene completion from a single depth image”. In: *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE. 2017, pp. 190–198.

[95] Yu-Chuan Su and Kristen Grauman. “Learning spherical convolution for fast features from 360 imagery”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 529–539.

[96] Hang Su et al. “Multi-view convolutional neural networks for 3d shape recognition”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 945–953.

[97] Danhang Tang et al. “Real-time compression and streaming of 4d performances”. In: *SIGGRAPH Asia 2018 Technical Papers*. ACM. 2018, p. 256.

[98] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. “Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2088–2096.

[99] Lyne Tchapmi et al. “Segcloud: Semantic segmentation of 3d point clouds”. In: *3D Vision (3DV), 2017 International Conference on*. IEEE. 2017, pp. 537–547.

[100] Ayush Tewari et al. “Mofa: Model-based deep convolutional face autoencoder for unsupervised monocular reconstruction”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 3715–3724.

[101] Ayush Tewari et al. “Self-supervised multi-level face model learning for monocular reconstruction at over 250 hz”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2549–2559.

[102] Maria Vakalopoulou et al. “AtlasNet: multi-atlas non-linear deep networks for medical image segmentation”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2018, pp. 658–666.

[103] Nanyang Wang et al. “Pixel2mesh: Generating 3d mesh models from single rgb images”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 52–67.

[104] Panqu Wang et al. “Understanding convolution for semantic segmentation”. In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2018, pp. 1451–1460.

[105] Peng-Shuai Wang et al. “O-cnn: Octree-based convolutional neural networks for 3d shape analysis”. In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), pp. 1–11.

[106] Yue Wang et al. “Dynamic graph CNN for learning on point clouds”. In: *arXiv preprint arXiv:1801.07829* (2018).

[107] Yue Wang et al. “Dynamic graph cnn for learning on point clouds”. In: *ACM Transactions on Graphics (TOG)* 38.5 (2019), pp. 1–12.

[108] Francis Williams et al. “Deep geometric prior for surface reconstruction”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 10130–10139.

[109] Jiajun Wu et al. “Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 82–90.

[110] Zhirong Wu et al. “3d shapenets: A deep representation for volumetric shapes”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1912–1920.

[111] Qiangeng Xu et al. “DISN: Deep Implicit Surface Network for High-quality Single-view 3D Reconstruction”. In: *arXiv preprint arXiv:1905.10711* (2019).

[112] Guandao Yang et al. “Pointflow: 3d point cloud generation with continuous normalizing flows”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 4541–4550.

[113] Li Yi et al. “Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 2282–2290.

[114] Fisher Yu and Vladlen Koltun. “Multi-Scale Context Aggregation by Dilated Convolutions”. In: *ICLR*. 2016.

[115] Andy Zeng et al. "3dmatch: Learning local geometric descriptors from rgb-d reconstructions". In: *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE. 2017, pp. 199–208.

[116] Yao Zhang, Woong Je Sung, and Dimitri N Mavris. "Application of Convolutional Neural Network to Predict Airfoil Lift Coefficient". In: *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*. 2018, p. 1903.

[117] Qiang Zhao et al. "Distortion-aware CNNs for Spherical Images." In: *IJCAI*. 2018, pp. 1198–1204.

Appendix A

Additional Details for DDSL

In the appendix we provide additional details for deriving the derivative of the NUFT process as well as control point methods (Sec. A.1), network architecture and training details (Sec. A.2). In Sec. A.3 we provide additional computational performance benchmarks for the DDSL layer. In Sec. A.4 we showcase additional applications of the DDSL towards 3D applications besided the 2D examples in the main paper. In Sec. A.5 we provide additional visualizations for the DDSL rasterization of 3D meshes.

A.1 Mathematical Derivations

NUFT Derivative Derivation

Proof of Lemma 2.3.1. Using Jacobi's formula and chain rule,

$$\frac{\partial \gamma_n^j}{\partial \boldsymbol{x}_p} = \frac{(-1)^{j+1}}{2\sqrt{2^j(-1)^{j+1}det(\hat{B}_n^j)}} tr\left(adj(\hat{B}_n^j)\frac{\partial \hat{B}_n^j}{\partial \boldsymbol{x}_p}\right) \tag{A.1}$$

$$= \frac{(-1)^{j+1}/2^j}{2\gamma_n^j}\sum_{m=1}^{j+2}\sum_{n=1}^{j+2}\tilde{A}_{mn}\tilde{D}_{nm} \tag{A.2}$$

where $\tilde{A}$ is $adj(\hat{B}_n^j)$ and $\tilde{D}$ is $\frac{\partial \hat{B}_n^j}{\partial \boldsymbol{x}_p}$. Since $\hat{B}_n^j$ is symmetric, its adjunctive and derivative with respect to $\boldsymbol{x}_p$ are also symmetric. The elements on the diagonal and the first row and column of $\tilde{D}$ are zero, since the elements in the same positions in $\hat{B}_n^j$ are constant. The elements not in the $(p+1)$th row or the $(p+1)$th column of $\tilde{D}$ are also zero, since the elements in these positions in $\hat{B}_n^j$ do not depend on $\boldsymbol{x}_p$. Thus,

$$\tilde{D} = \begin{bmatrix} 0 & \dots & 0 & 0 & 0 & \dots \\ \vdots & \ddots & \vdots & \vdots & \vdots & \\ 0 & \dots & 0 & \tilde{D}_{p,p+1} & 0 & \dots \\ 0 & \dots & \tilde{D}_{p+1,p} & 0 & \tilde{D}_{p+1,p+2} & \dots \\ 0 & \dots & 0 & \tilde{D}_{p+2,p+2} & 0 & \dots \\ \vdots & & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{A.3}$$

Each nonzero element of $\tilde{D}$ is computed as follows:

$$\tilde{D}_{p+1,n} = \frac{\partial d^2_{p,n-1}}{\partial \boldsymbol{x}_p} = 2(\boldsymbol{x}_p - \boldsymbol{x}_{n-1}) \tag{A.4}$$

$$\tilde{D}_{m,p+1} = \frac{\partial d^2_{m-1,p}}{\partial \boldsymbol{x}_p} = 2(\boldsymbol{x}_p - \boldsymbol{x}_{m-1}) \tag{A.5}$$

It follows that the double summation term in Eqn. A.2 simplifies to

$$\sum_{m=1}^{j+2} \sum_{n=1}^{j+2} \tilde{A}_{mn} \tilde{D}_{nm} = 2 \sum_{\substack{m=2 \\ m \neq p+1}}^{j+2} \tilde{A}_{p+1,m} \tilde{D}_{p+1,m} \tag{A.6}$$

For clarity and ease of implementation, we modify the indexing in Eqn. A.6 and the derivative of the content distortion factor is finally

$$\frac{\partial \gamma^j_n}{\partial \boldsymbol{x}_p} = \frac{(-1)^{j+1}/2^j}{\gamma^j_n} \sum_{\substack{m=1 \\ m \neq p}}^{j+1} A_{pm} \boldsymbol{D}_{pm} \tag{A.7}$$

$\square$

Proof of Lemma 2.3.2. By the sum rule,

$$\frac{\partial S}{\partial \boldsymbol{x}_p} = \sum_{t=1}^{j+1} \frac{\partial S_t}{\partial \boldsymbol{x}_p} \tag{A.8}$$

We examine two cases, when $t = p$ and when $t \neq p$. For $t = p$,

$$\begin{aligned}
\frac{\partial S_t}{\partial \boldsymbol{x}_p} =& \frac{1}{\left(\prod_{l=1,l\neq p}^{j+1}(\sigma_p - \sigma_l)\right)^2}\boldsymbol{k} \\
& \left[\left(\prod_{l=1,l\neq p}^{j+1}(\sigma_p - \sigma_l)\right)\left(-ie^{-i\sigma_p}\right)\right. \\
& \left. + e^{-i\sigma_p}\left(\frac{\partial}{\partial \boldsymbol{x}_p}\left(\prod_{l=1,l\neq p}^{j+1}(\sigma_p - \sigma_l)\right)\right)\right] && \text{(A.9)} \\
=& -\frac{e^{-i\sigma_p}}{\prod_{l=1,l\neq p}^{j+1}(\sigma_p - \sigma_l)}\left(i + \sum_{q=1,q\neq p}^{j+1}\frac{1}{\sigma_p - \sigma_q}\right)\boldsymbol{k} && \text{(A.10)}
\end{aligned}$$

For $t \neq p$,

$$\begin{aligned}
\frac{\partial S_t}{\partial \boldsymbol{x}_p} =& \frac{\partial}{\partial \boldsymbol{x}_p}\left(\frac{e^{-i\sigma_t}}{(\sigma_t - \sigma_1)...(\sigma_t - \sigma_p)...(\sigma_t - \sigma_{j+1})}\right) && \text{(A.11)} \\
=& \left(\frac{e^{-i\sigma_t}}{\begin{array}{r}(\sigma_t - \sigma_1)...(\sigma_t - \sigma_{p-1})(\sigma_t - \sigma_{p+1}) \\ ...(\sigma_t - \sigma_{j+1})\end{array}}\right) \\
& \left(\frac{\partial}{\partial \boldsymbol{x}_p}\left(\frac{1}{\sigma_t - \sigma_p}\right)\right) && \text{(A.12)} \\
=& \left(\frac{e^{-i\sigma_t}}{\begin{array}{r}(\sigma_t - \sigma_1)...(\sigma_t - \sigma_{p-1})(\sigma_t - \sigma_{p+1}) \\ ...(\sigma_t - \sigma_{j+1})\end{array}}\right) \\
& \left(\frac{1}{(\sigma_t - \sigma_p)^2}\boldsymbol{k}\right) && \text{(A.13)} \\
=& \frac{e^{-i\sigma_t}}{\prod_{l=1,l\neq t}^{j+1}(\sigma_t - \sigma_l)}\left(\frac{1}{\sigma_t - \sigma_p}\right)\boldsymbol{k} && \text{(A.14)}
\end{aligned}$$

Thus,

$$
\begin{aligned}
\frac{\partial S}{\partial \boldsymbol{x}_p} =& \sum_{t=1}^{j+1} \frac{\partial S_t}{\partial \boldsymbol{x}_p} && \text{(A.15)}\\
=& \left(\sum_{t=1,t\neq p}^{j+1} \left(\frac{e^{-i\sigma_t}}{\prod_{l=1,l\neq t}^{j+1}(\sigma_t-\sigma_l)} \left(\frac{1}{\sigma_t-\sigma_p} \right) \right) \right. \\
& \left. - \frac{e^{-i\sigma_p}}{\prod_{l=1,l\neq p}^{j+1}(\sigma_p-\sigma_l)} \left(i + \sum_{q=1,q\neq p}^{j+1} \frac{1}{\sigma_p-\sigma_q} \right) \right) \boldsymbol{k} && \text{(A.16)}\\
=& \left(-i \frac{e^{-i\sigma_p}}{\prod_{l=1,l\neq p}^{j+1}(\sigma_p-\sigma_l)} + \sum_{t=1,t\neq p}^{j+1} \frac{1}{\sigma_t-\sigma_p} \right. \\
& \left. \left[\frac{e^{-i\sigma_t}}{\prod_{l=1,l\neq t}^{j+1}(\sigma_t-\sigma_l)} + \frac{e^{-i\sigma_p}}{\prod_{l=1,l\neq p}^{j+1}(\sigma_p-\sigma_l)} \right] \right) \boldsymbol{k} && \text{(A.17)}\\
=& \left(-iS_p + \sum_{t=1,t\neq p}^{j+1} \frac{S_t+S_p}{\sigma_t-\sigma_p} \right) \boldsymbol{k} && \text{(A.18)}
\end{aligned}
$$

□

Derivation of Eqn. 2.14. Using the product rule,

$$
\frac{\partial F_n^j(\boldsymbol{k})}{\partial \boldsymbol{x}_p} = \rho_n i^j \left(\frac{\partial \gamma_n^j}{\partial \boldsymbol{x}_p} S + \frac{\partial S}{\partial \boldsymbol{x}_p} \gamma_n^j \right) \tag{A.19}
$$

We obtain Eqn. 2.14 by substituting Eqns. 2.11 and 2.12 into Eqn. A.19. □

Control Points

We use linear blend skinning to control mesh deformation using control points. The new position of a point $\boldsymbol{v}'$ on the shape is computed as the weighted sum of handle transformations applied to its rest position $\boldsymbol{v}$:

$$
\boldsymbol{v}' = \sum_{j=1}^{m} w_j(\boldsymbol{v}) \boldsymbol{T}_j \begin{pmatrix} \boldsymbol{v} \\ 1 \end{pmatrix}
$$

Where $\boldsymbol{T}_j$ is the transformation matrix for the j-th control point, $w_j(\boldsymbol{v})$ is the normalized weight on vertex $\boldsymbol{v}$ corresponding to control point j. The transformation is represented in homogeneous coordinates, hence the extra dimension.

Consider control points with 3 degrees of freedom: (t_x, t_y, θ) where t_x and t_y represent translations in x and y and θ represents rotation around that control point. Hence we have

$$\begin{cases} v'_x = & \sum_{j=1}^{N} w_j(\boldsymbol{v})\big(\cos(\theta_j - \tilde{\theta}_j)v_x - \sin(\theta_j - \tilde{\theta}_j)v_y \\ & -\cos(\theta_j - \tilde{\theta}_j)c_x + \sin(\theta_j - \tilde{\theta}_j)c_y \\ & +c_x + v_x + t_x\big) \\ v'_y = & \sum_{j=1}^{N} w_j(\boldsymbol{v})\big(\sin(\theta_j - \tilde{\theta}_j)v_x + \cos(\theta_j - \tilde{\theta}_j)v_y \\ & -\sin(\theta_j - \tilde{\theta}_j)c_x - \cos(\theta_j - \tilde{\theta}_j)c_y \\ & +c_y + v_y + t_y\big) \end{cases}$$

Where $\tilde{\theta_j}$ is the original orientation of the control points. It does not matter since we will be taking the derivatives with respect to θ, and $\tilde{\theta_j}$ terms will disappear. The jacobian of $\boldsymbol{v}$ with respect to the three degrees of freedom is:

$$\begin{aligned} \boldsymbol{J} &= \left[\frac{\partial \boldsymbol{v}}{\partial t_x}, \frac{\partial \boldsymbol{v}}{\partial t_y}, \frac{\partial \boldsymbol{v}}{\partial \theta}\right] \\ &= \begin{bmatrix} w_j(\boldsymbol{v}) & 0 & w_j(\boldsymbol{v})(-v_y + c_y) \\ 0 & w_j(\boldsymbol{v}) & w_j(\boldsymbol{v})(v_x - c_x) \end{bmatrix} \end{aligned}$$

A.2 Network Architecture and Training Details

In this section, we detail all the network architectures and training routines for the reader's reference.

Notation	Meaning
Conv(a, b, c, d)	Convolutional layer with a input channels, b output channels, kernel size c, and stride d.
MaxPool(a)	Maximum Pooling with a kernel size of a.
ReLU	Rectified Linear Unit activation function.
FC(a, b)	Fully connected layer with a input channels and b output channels.
ResNet-50(a)	ResNet-50 architecture with a output channels.
BN	Batch Normalization.

Table A.1: Network architecture notation list.

MNIST

We use a standard LeNet-5 architecture with 3 convolutional layers and 2 fully connected layers.

Network Architecture The input is a 28x28 pixel image, which is normalized according to the mean and standard deviation of the entire dataset. The network architecture is as follows:

Conv(1, 10, 5, 1) + MaxPool(2) + ReLU → Conv(10, 20, 5, 1) + Dropout + MaxPool(2) + ReLU → FC(320, 250) + ReLU → Dropout → FC(250, 10)

Total number of parameters: 88,040

Training Details We train the neural network with a batch size of 64 and an initial learning rate of 1×10^{-2} with a decay of 0.5 per 10 epochs. We use the Stochastic Gradient Descent optimizer with a momentum of 0.5 and a cross entropy loss.

Airfoil

We use ResNet-50 [37] followed by three fully connected layers to predict the lift-drag ratio on the airfoil.

Network Architecture The input is a 224x224 pixel image of the airfoil. For each piece of data, we append the Reynolds number and angle of attack after ResNet-50 and before the fully connected layers. The network architecture is as follows:

ResNet-50(1000) + BN + ReLU → append Reynolds number and angle of attack → FC(1002, 512) + BN + ReLU → FC(512, 64) + BN + ReLU → FC(64, 32) + BN

Total number of parameters: 26,100,345

Training Details We train the neural network with a batch size of 240 and an initial learning rate of 1×10^{-2} with a decay of 1×10^{-1} per 20 epochs. We use the Adam optimizer and a mean squared error loss.

Polygon Image Segmentation

We present a novel polygon decoder architecture that is paired with a standard pre-trained ResNet50 as input.

Network Architecture The model architecture is detailed in Fig. 2.4. All ground-truth polygons are normalized to the range [0,1) corresponding to the relative positions within the bounding boxes. Using this network architecture, we first predict the three (x, y) coordinates associated with the base triangle. Then, we progressively predict the offsets of the vertices in the next polygon hierarchy (See Fig. 2.3). The resulting polygon is rasterized with the DDSL to compute the rasterization loss compared with the rasterized target. Smoothness loss can be directly computed based on the vertex positions and does not require rasterization.

Total number of parameters: 24,274,426

Training Details We train the network end-to-end, with a batch size of 48, learning rate of 10^{-3} for 200 epochs. We use a smoothness penalty of $\lambda = 1$. We use the Adam optimizer.

A.3 Additional Computational Efficiency Tests

In addition to the computational speed benchmarks in Fig. 2.5 highlighting the performance gain of analytic derivative computation over numerical derivatives, we perform additional tests for 2D and 3D computation speeds on more complex polygons and meshes to show the applicability of DDSL to 2D and 3D computer vision problems.

Res^2	16	32	64	128	256
Fwd Time (ms)	2.30	1.88	2.48	5.02	20.13
Bwd Time (ms)	4.33	3.80	5.93	16.69	59.15

Table A.2: 2D Computational speed (polygon w/ 250 edges).

Res^3	4	8	16	32
Fwd Time (ms)	9.88	9.32	14.21	78.62
Bwd Time (ms)	14.47	10.06	34.26	239.51

Table A.3: 3D Computational speed (tri-mesh w/ 1300 faces).

A.4 3D Geometric Applications

To showcase the generalizabilty of the DDSL to 3D domain, we demonstrate its application in two separate 3D tasks that utilze the differentiablity of the simplex rasterization layer.

3D Rotational Pose Estimation

In Fig. A.1, we use DDSL to create a differentiable volumetric loss comparing current and target shapes, the gradients of which can be backpropagated to the pose. More specifically, we parameterize the rotational pose as a quaternion $\boldsymbol{q} = a + b\hat{\boldsymbol{i}} + c\hat{\boldsymbol{j}} + d\hat{\boldsymbol{k}}, \quad s.t. ||\boldsymbol{q}||_2 = 1$. The rasterization loss is defined as:

$$\mathcal{L}(\boldsymbol{q}) = ||D_{32}(V(\boldsymbol{q})) - D_{32}(V_{tg})||_1$$

where D_{32} is the rasterization operator at resolution 32^3 and V_{tg} is the target mesh.

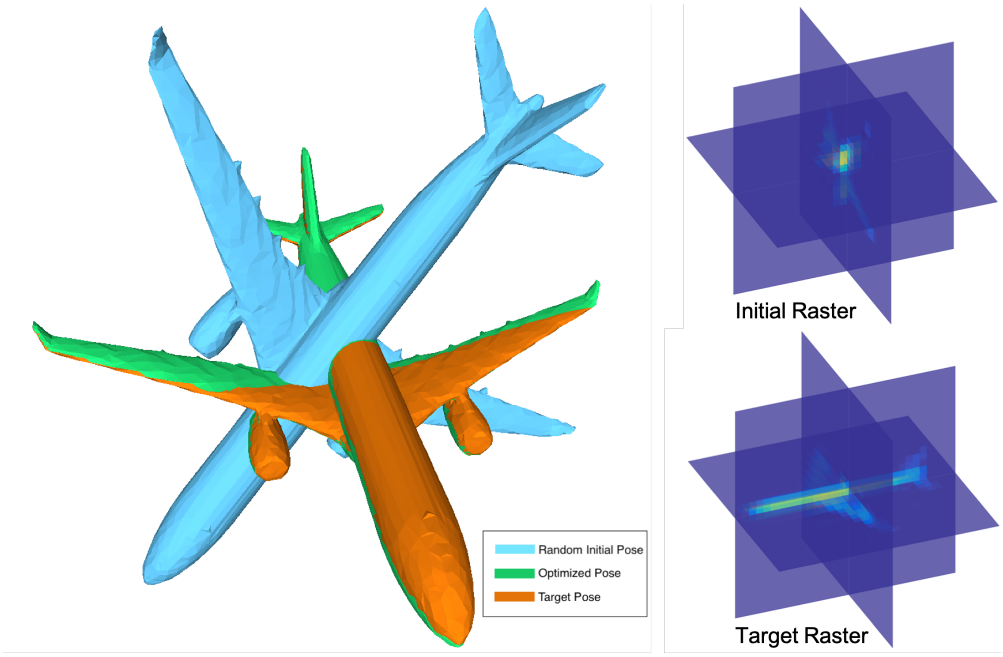


Figure A.1: Mesh pose and rasters before and after opt.

Although the volumetric rasterization loss is not a globally convex loss for pose alignment, with certain initialization of the target poss, the pose can be estimated by minimizing the DDSL rasterization loss.

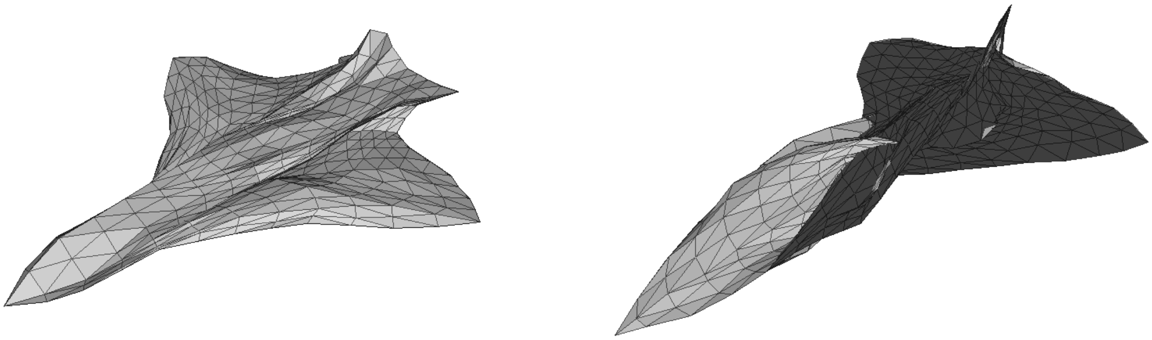
Single Image Mesh Estimation

In Fig. A.2, we evaluate our method in the context of 3D deep learning. Our model consists of an image encoder from ResNet18, spherical convolutions [46] for generating a distortion map for a spherical mesh, and a loss function which is a weighted sum of DDSL rasterization loss (at 32^3 resolution), Chamfer loss from point samples, Laplacian regularization loss, and Edge length regularization loss. We train on the airplane category in ShapeNet dataset, with (w/) and without (w/o) DDSL loss. We evaluate using accuracy, completeness, and chamfer distance metrics (see Tab. A.4).

Since surface based Chamfer distance does not signal the network to produce consistently oriented surfaces and does not consistently enclose volume, it leads to incorrectly oriented surfaces. DDSL loss effective regularizes surface orientation based on the volume enclosed according to the surface orientations, and improves overall results.

DDSL	Accuracy	Complete	Chamfer
w/o	8.47	9.84	9.16
w/	**2.15**	**1.83**	**1.99**

Table A.4: Evaluation resultsn($\times 10^{-}2$).



(a) w/ DDSL

(b) w/o DDSL

Figure A.2: Qualitative visualization of generated samples.

A.5 Additional 3D Visualizations

We provide visualizations for rasterizing 3D shapes, rasterizing the enclosed volume as well as the surface mesh.

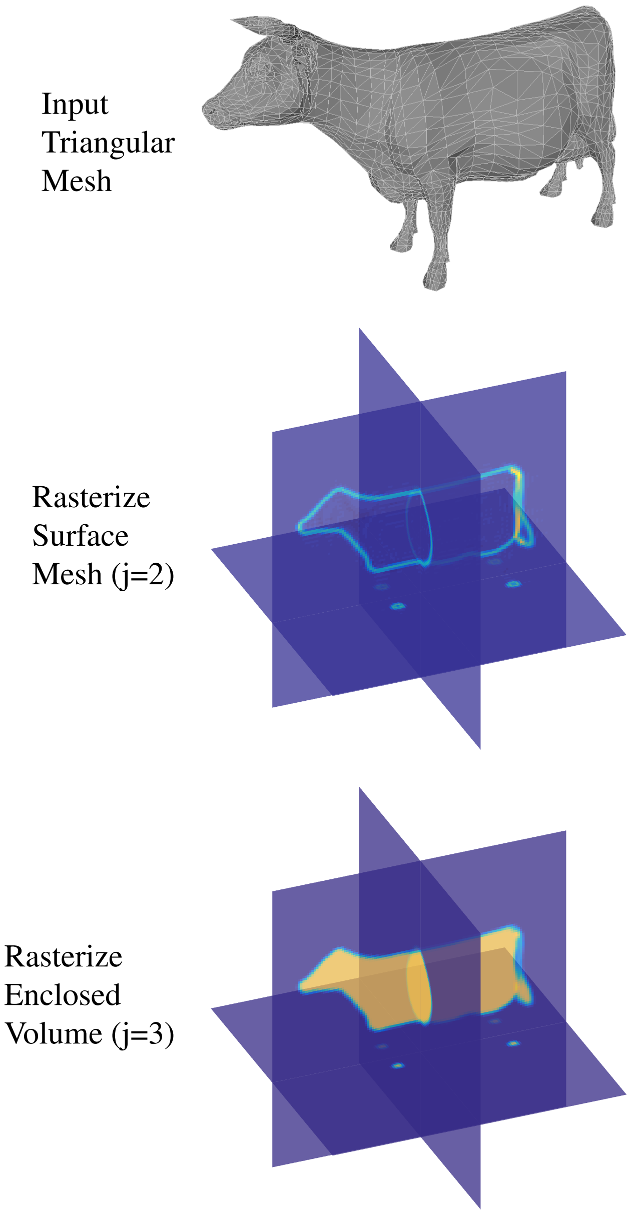


Figure A.3: In this example above, the input is a watertight triangluar mesh represented by vertices and faces. It can be rasterized in-situ in a 3-dimensional grid differentiably. The value is approximately 0 or 1 indicating signal densities.

Appendix B

Additional Details for UGSCNN

Notation	Meaning
MeshConv(a, b)	Mesh convolution layer with a input channels and producing b output channels
MeshConv(a, b)$^{\mathrm{T}}$	Mesh transpose convolution layer with a input channels and producing b output channels.
BN	Batch Normalization.
ReLU	Rectified Linear Unit activation function
DownSamp	Downsampling spherical signal at the next resolution level.
ResBlock(a, b, c)	As illustrated in Fig. 3.1, where a, b, c stands for input channels, bottle neck channels, and output channels.
$[\,]_{\mathrm{L}i}$	The layers therein is at a mesh resolution of Li.
Concat	Concatenate skip layers of same resolution.

Table B.1: Network architecture notation list

B.1 Network Architecture and Training Details

In this section we provide detailed network architecture and training parameters for reproducing our results in Sec. 3.4. We use Fig. 3.2 as a reference.

Spherical MNIST

Architecture Since the input signal for this experiment is on a level-4 mesh, the input pathway is slightly altered. The network architecture is as follows:

[MeshConv(1,16) + BN + ReLU]$_{\mathrm{L4}}$ + [DownSamp + ResBlock(16, 16, 64)]$_{\mathrm{L3}}$ + [DownSamp + ResBlock(64, 64, 256)]$_{\mathrm{L2}}$ + AvgPool + MLP(256, 10)

Total number of parameters: 61658

Training details We train our network with a batch size of 16, initial learning rate of 1×10^{-2}, step decay of 0.5 per 10 epochs, and use the Adam optimizer. We use the cross-entropy loss for training the classification network.

3D Object Classification

Architecture The input signal is at a level-5 resolution. The network architecture closely follows that in the schematics in Fig. 3.2. We present two network architectures, one that corresponds to the network architecture with the highest accuracy score (the full model), and another that scales well with low parameter counts (the lean model). The full model:

[MeshConv(6, 32) + BN + ReLU]$_{\mathrm{L5}}$ + [DownSamp + ResBlock(32, 32, 128)]$_{\mathrm{L4}}$ + [DownSamp + ResBlock(128, 128, 512)]$_{\mathrm{L3}}$ + [DownSamp + ResBlock(512, 512, 2048)]$_{\mathrm{L2}}$ + AvgPool + MLP(2048, 40)

Total number of parameters: 3737160

The lean model:

[MeshConv(6, 8) + BN + ReLU]$_{\mathrm{L5}}$ + [DownSamp + ResBlock(8, 8, 16)]$_{\mathrm{L4}}$ + [DownSamp + ResBlock(16, 16, 64)]$_{\mathrm{L3}}$ + [DownSamp + ResBlock(64, 64, 256)]$_{\mathrm{L2}}$ + AvgPool + MLP(256, 40)

Total number of parameters: 70192

Training details We train our network with a batch size of 16, initial learning rate of 5×10^{-3}, step decay of 0.7 per 25 epochs, and use the Adam optimizer. We use the cross-entropy loss for training the classification network.

Omnidirectional Image Segmentation

Architecture Input signal is sampled at a level-5 resolution. The network architecture is identical to the segmentation network in Fig.3.2. Encoder parameters are as follows:

[MeshConv(4,32)]$_{\mathrm{L5}}$ + [DownSamp + ResBlock(32, 32, 64)]$_{\mathrm{L4}}$ + [DownSamp + ResBlock(64, 64, 128)]$_{\mathrm{L3}}$ + [DownSamp + ResBlock(128, 128, 256)]$_{\mathrm{L2}}$ + [DownSamp + ResBlock(256, 256, 512)]$_{\mathrm{L1}}$ + [DownSamp + ResBlock(512, 512, 512)]$_{\mathrm{L0}}$

Decoder parameters are as follows:

[MeshConv$^{\mathrm{T}}$(512,512) + Concat + ResBlock(1024, 256, 256)]$_{\mathrm{L1}}$ + [MeshConv$^{\mathrm{T}}$(256,256) + Concat + ResBlock(512, 128, 128)]$_{\mathrm{L2}}$ + [MeshConv$^{\mathrm{T}}$(128,128) + Concat + ResBlock(256, 64, 64)]$_{\mathrm{L3}}$ + [MeshConv$^{\mathrm{T}}$(64,64) + Concat + ResBlock(128, 32, 32)]$_{\mathrm{L4}}$ + [MeshConv$^{\mathrm{T}}$(32,32) + Concat + ResBlock(64, 32, 32)]$_{\mathrm{L5}}$ + [MeshConv(32,15)]$_{\mathrm{L5}}$

Total number of parameters: 5180239

Training details Note that the number of output channels is 15, since the 2D3DS dataset has two additional classes (invalid and unknown) that are not evaluated for performance. We train our network with a batch size of 16, initial learning rate of 1×10^{-2}, step decay of 0.7 per 20 epochs, and use the Adam optimizer. We use the weighted cross-entropy loss for training. We weight the loss for each class using the following weighting scheme:

$$w_c = \frac{1}{1.02 + \log(f_c)} \tag{B.1}$$

where w_c is the weight corresponding to class c, and f_c is the frequency by which class c appears in the training set. We use zero weight for the two dropped classes (invalid and unknown).

Climate Pattern Segmentation

Architecture We use the same network architecture as the Omnidirectional Image Segmentation task in Sec.B.1. Minor difference being that all feature layers are cut by 1/4.

Total number of parameters: 328339

Training details We train our network with a batch size of 256, initial learning rate of 1×10^{-2}, step decay of 0.4 per 20 epochs, and use the Adam optimizer. We train using weighted cross-entropy loss, using the same weighting scheme as in Eqn.B.1.

B.2 Detailed statistics for 2D3DS segmentation

We provide detailed statistics for the 2D3DS semantic segmentation task. We evaluate our model's per-class performance against the benchmark models. All statistics are mean over 3-fold cross validation.

Model	Mean	beam	board	bookcase	ceiling	chair	clutter	column	door	floor	sofa	table	wall	window
UNet	0.5080	0.1777	0.4038	**0.5914**	0.9180	0.5088	0.4603	0.0875	0.4398	0.9480	0.2623	0.6865	**0.7717**	0.3481
FCN8s	0.4842	0.1439	0.4413	0.3952	0.8971	0.5244	**0.5759**	0.0564	0.5962	**0.9661**	0.0322	0.6614	0.7359	0.2682
PointNet++	0.3349	0.1928	0.2942	0.3277	0.5448	0.4145	0.2246	**0.3110**	0.2701	0.4596	**0.3391**	0.4976	0.3358	0.1413
Ours	**0.5465**	**0.1964**	**0.4856**	0.4964	**0.9356**	**0.6382**	0.4309	0.2798	**0.6321**	0.9638	0.2103	**0.6996**	0.7457	**0.3897**

Table B.2: Per-class accuracy comparison with baseline models

Model	Mean	beam	board	bookcase	ceiling	chair	clutter	column	door	floor	sofa	table	wall	window
UNet	0.3587	0.0853	0.2721	0.3072	0.7857	0.3531	0.2883	0.0487	0.3377	**0.8911**	0.0817	0.3851	0.5878	**0.2392**
FCN8s	0.3560	0.0572	0.3139	0.2894	0.7981	0.3623	**0.2973**	0.0353	0.4081	0.8884	0.0263	0.3809	0.5849	0.1859
PointNet++	0.2312	**0.0909**	0.1503	0.2210	0.4775	0.2981	0.1610	0.0782	0.1866	0.4426	**0.1844**	0.3332	0.3061	0.0755
Ours	**0.3829**	0.0869	**0.3268**	**0.3344**	**0.8216**	**0.4197**	0.2562	**0.1012**	**0.4159**	0.8702	0.0763	**0.4170**	**0.6167**	0.2349

Table B.3: Mean IoU comparison with baseline models

Appendix C

Additional Details for LIG

C.1 Additional visual results

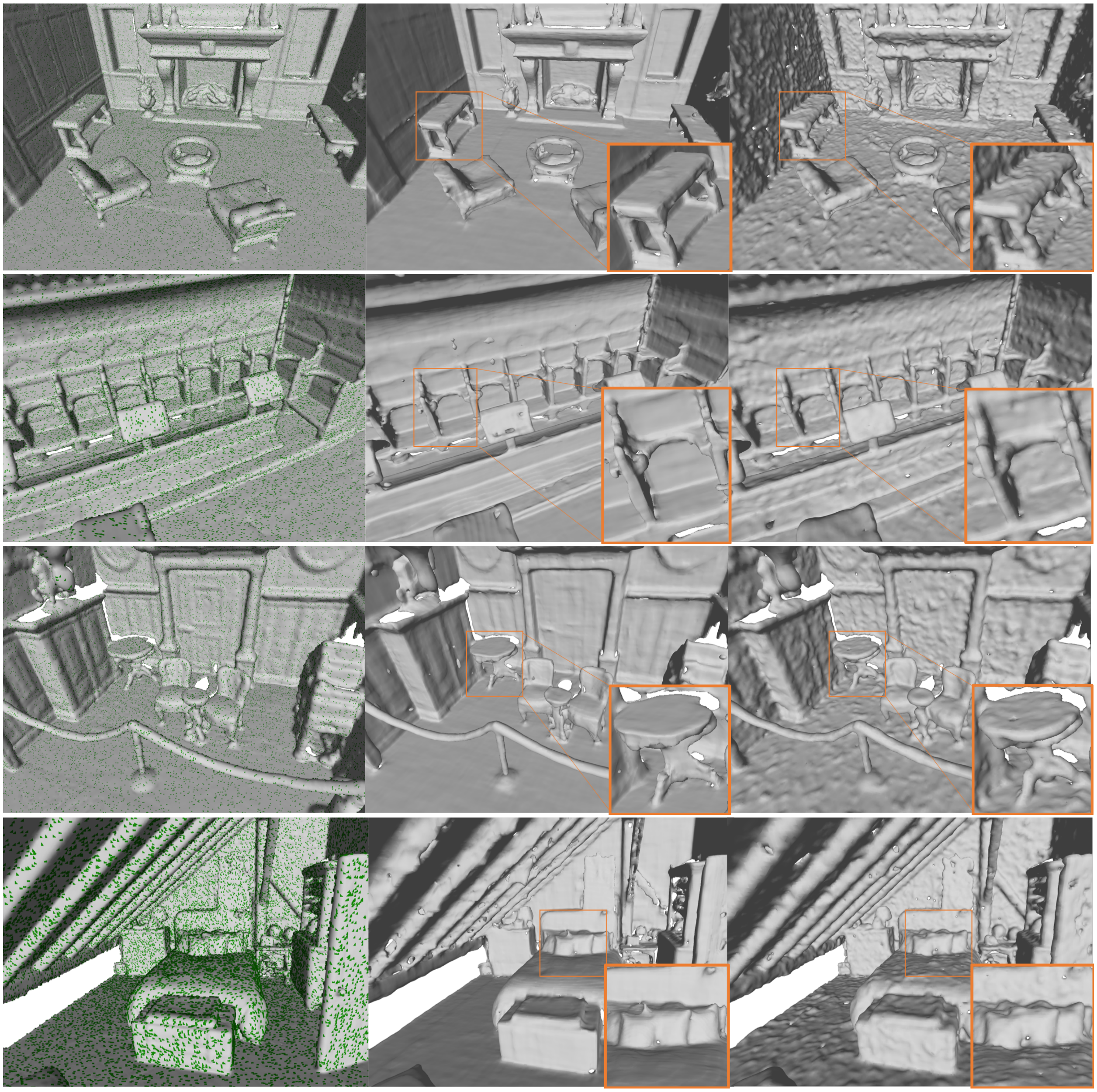


Figure C.1: Left: Ground truth mesh overlaid with input point samples; Middle: Our reconstruction; Right: Screened PSR [55] reconstruction. The input are point samples from the Matterport ground truth mesh at a sample density of 500 points / m^2.

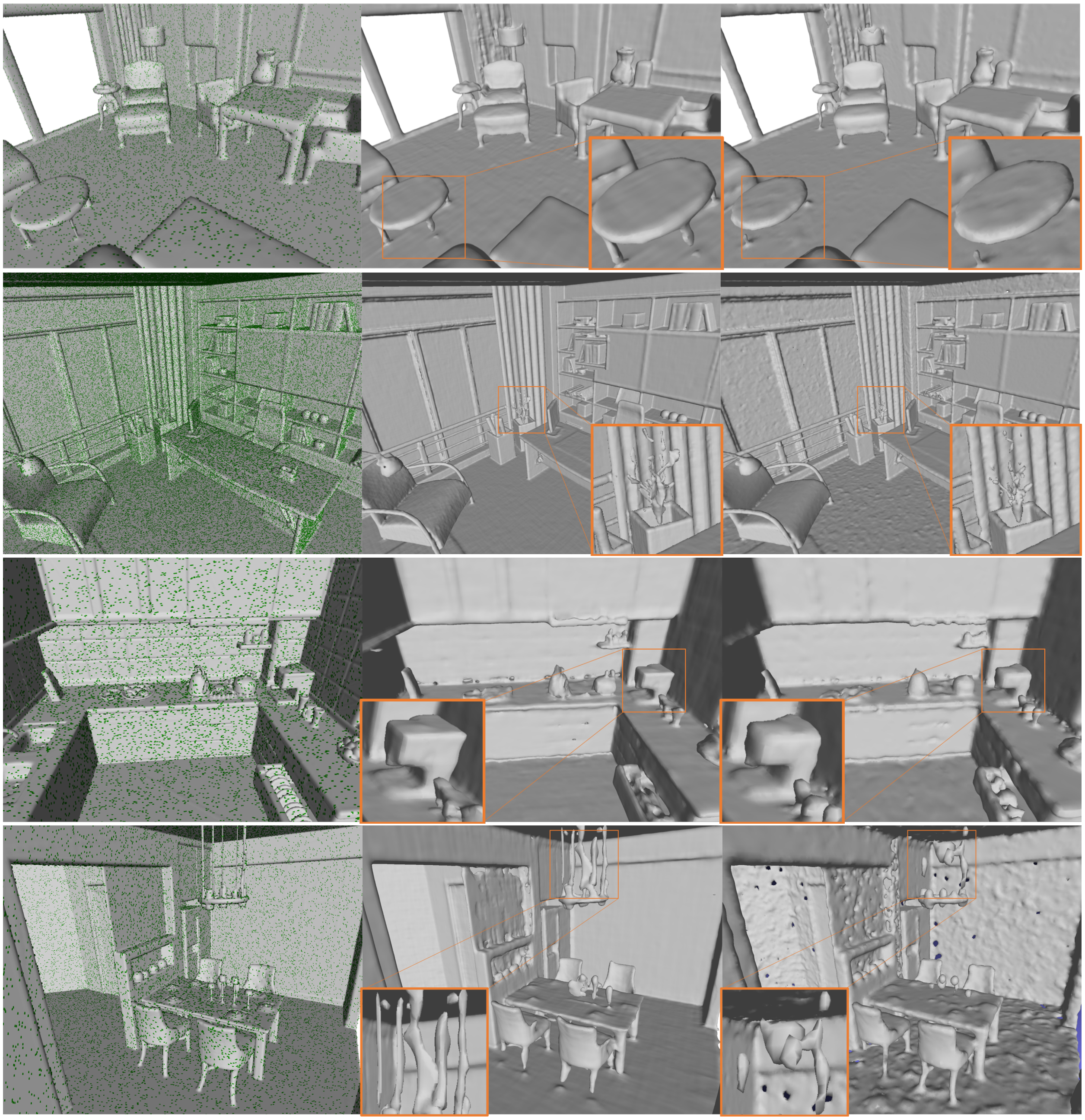
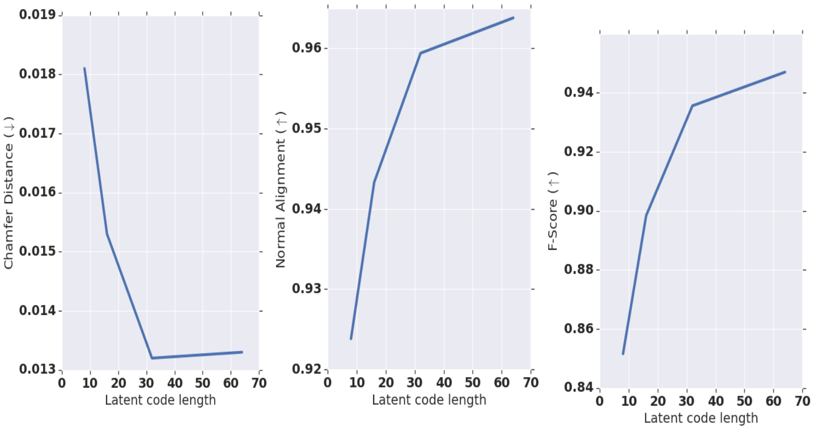


Figure C.2: Left: Ground truth mesh overlaid with input point samples; Middle: Our reconstruction; Right: Screened PSR [55] reconstruction. The input are point samples from the SceneNet ground truth mesh at a sample density of 500 points / m^2.

CL	CD(↓)	Normal(↑)	F-Scor
8	0.0181	0.9238	0.8516
16	0.0153	0.9433	0.8984
32	0.0132	0.9594	0.9356
64	0.0133	0.9638	0.9470

Table C.1: Additional ablation study on the effects of latent code length (CL). Reconstruction performance measured on SceneNet reconstruction from 100 point samples / m^2.



(a) Chamfer Distance

(b) Normal Alignment

(c) F-Score

Figure C.3: Line plot for Chamfer Distance, Normal Alignment and F-Score versus Latent Code Length.

C.2 Additional ablation studies

We perform additional ablation studies on the effects of latent code length on reconstruction performace. See Table C.1 and Fig. C.3 for reference. With increasing number of latent channels, the reconstruction performance improves with diminishing marginal improvement. Our choice of 32 latent channels strikes a good balance between performance and efficiency.

C.3 Implementation details

Model architecture

We present a schematic of our encoder architecture for our part autoencoder in Fig. C.4. The input to the encoder is a normalized TSDF crop of the part to be encoded, and the encoder uses 3D CNNs to encode the input into a latent code of dimensions 32. The encoder is decorated with residue blocks with bottleneck layers for improved performance.

We refer the reader to [14] for the architecture for our refiner. We preserve the architecture of the IM-NET model, but reduce the latent dimension from 128 to 32, and reduce the number of hidden layers in every layer of the model to 1/4 of the original value for improved efficiency, due to the fact that part geometries are easier to learn and represent than entire objects.

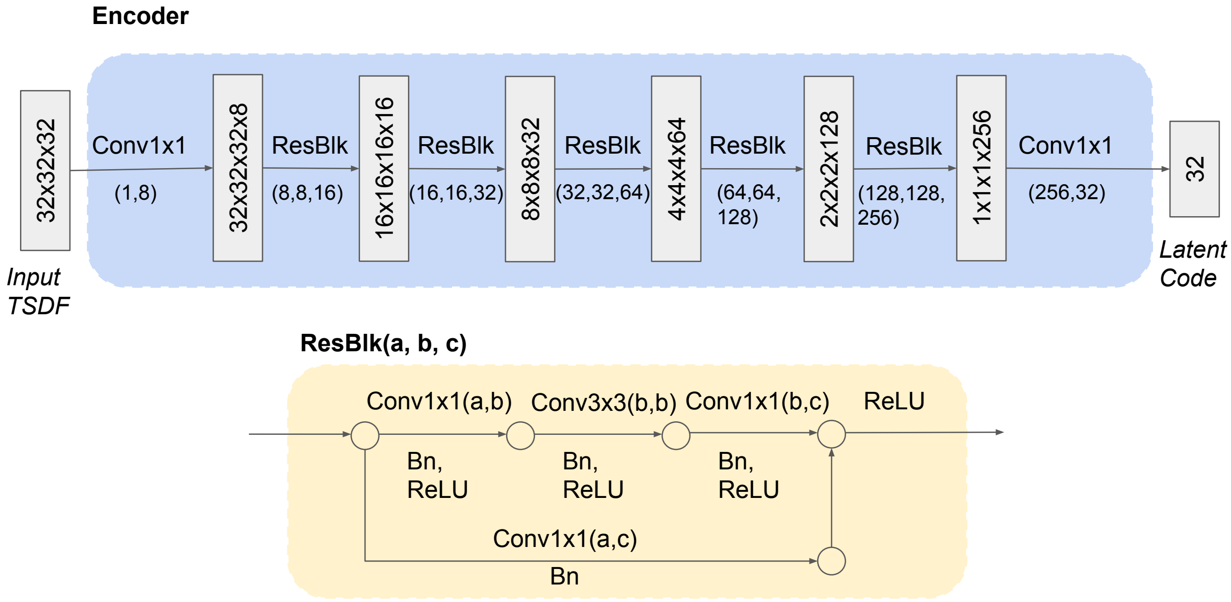


Figure C.4: Encoder architecture. The encoder is a simple 3D CNN decorated with residue blocks, that encodes 3D TSDF tensors into latent codes, which can be decoded into implicit surfaces by an implicit network decoder.

Part autoencoder training

For training the part autoencoder, we use a batch size of 32, and for each shape we sample 2048 point samples. We train with a latent penalty factor $\lambda = 10^{-2}$, learning rate of 10^{-3}. We sample empty volumes with a probability of 10^{-3} to embed empty space. We train the part autoencoder for a total of 10^7 steps.

Inference

For reconstructing geometries from point samples, for each point sample, we sample 10 points along the point normal with a standard deviation of 1cm. For the Local Implicit Grid, we initialize each cell with Gaussian normal random values with a standard deviation of 0.01. During latent grid optimization, we use 32768 random point samples per batch, and optimize with a learning rate of 10^{-3}. We optimize for a fixed 10000 steps. When extracting the final mesh, we extract the mesh at $1/64m$ resolution.