DDSL: Deep Differentiable Simplex Layer for Learning Geometric Signals

Chiyu "Max" Jiang[*1]   Dana Lansigan[*1]   Philip Marcus[1]   Matthias Nießner[2]

[1]UC Berkeley   [2]Technical University of Munich

Abstract

We present a Deep Differentiable Simplex Layer (DDSL) for neural networks for geometric deep learning. The DDSL is a differentiable layer compatible with deep neural networks for bridging simplex mesh-based geometry representations (point clouds, line mesh, triangular mesh, tetrahedral mesh) with raster images (e.g., 2D/3D grids). The DDSL uses Non-Uniform Fourier Transform (NUFT) to perform differentiable, efficient, anti-aliased rasterization of simplex-based signals. We present a complete theoretical framework for the process as well as an efficient backpropagation algorithm. Compared to previous differentiable renderers and rasterizers, the DDSL generalizes to arbitrary simplex degrees and dimensions. In particular, we explore its applications to 2D shapes and illustrate two applications of this method: (1) mesh editing and optimization guided by neural network outputs, and (2) using DDSL for a differentiable rasterization loss to facilitate end-to-end training of polygon generators. We are able to validate the effectiveness of gradient-based shape optimization with the example of airfoil optimization, and using the differentiable rasterization loss to facilitate end-to-end training, we surpass state of the art for polygonal image segmentation given ground-truth bounding boxes.
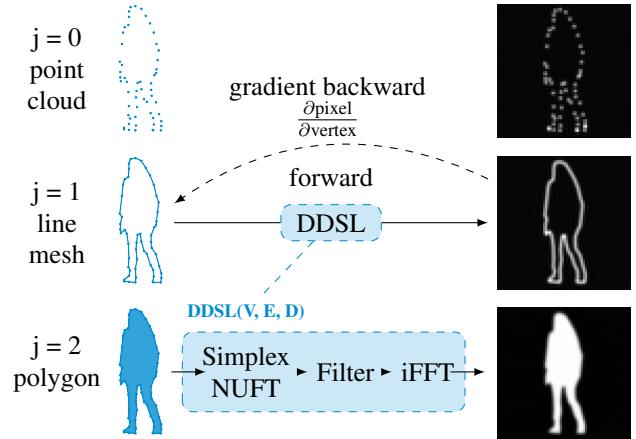
Figure 1: A schematic of the DDSL layer with 2D simplex meshes. The DDSL algorithm is general for handling simplex meshes of arbitrary dimensions and simplex degrees. The input to DDSL is a simplex mesh described by three matrices: float matrix V for vertex coordinates, uint matrix E for simplex connectivity, 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.

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 [3].

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) [18] 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

---

[*] Equal contributions

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. 4.2). Next, to further 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. 4.2). 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. 4.3). 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[1], with compelling computational performance benchmarks.

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.

---

[1]Code available: https://github.com/maxjiang93/DDSL

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 [36, 37], various algorithms perform convolutions natively on the mesh manifold, [17, 15, 2], the graph [10, 24, 46] 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 [32, 45], Truncated Signed-Distance Function (TSDF) based algorithms [7, 48, 40, 8], multi-view image based algorithms [41, 21], and hybrids [19, 6]. 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. [30] proposed an approximate differentiable rasterizer for inverse graphics. [22] proposed a deep neural renderer that uses linear approximations for the gradients of the pixel intensity with respect to the vertex positions. [26] introduced a differentiable ray-tracer for differentiability of additional rendering effects. Very recently, [28] proposed a differentiable rasterizer that approximates rendering derivatives with soft boundaries. Various studies in face mesh reconstruction applications [11, 42, 43, 38] and general mesh reconstruction tasks [20, 25] 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 has been used for gradient-based optimizations with sensitivities acquired from physics simulators

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 1: List of math symbols in our method.

[35, 16]. Recently, machine learning algorithms such as multilayer perceptrons have been used as surrogate models for the response surface to speed up evaluation and optimization [23, 31]. More recently, CNNs have been used for the evaluation of aerodynamic properties [49], and gradient-based optimization methods coupled with CNNs have been explored [14]. 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 [29, 39, 44, 12, 9, 27]. However, more recently, to address the need of assisting human annotators to create ground-truth segmentation labels, new network architectures such as PolygonRNN [4] 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.

3. Method

3.1. DDSL Overview

A schematic of the DDSL layer is presented in Fig. 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.

3.2. 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 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 $v_0, \ldots, v_j \in \mathbb{R}^n$ is

$$
\begin{aligned}
C &= \mathbf{conv}\{v_0, \ldots, v_j\} \\
&= \{\theta_0 v_0 + \cdots + \theta_j v_j \mid \boldsymbol{\theta} \succeq 0, \ \mathbf{1}^T \boldsymbol{\theta} = 1\}
\end{aligned} \quad (1)
$$

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

Definition 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 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 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:

$$
f_n^j(\boldsymbol{x}) = \begin{cases} \rho_n, \boldsymbol{x} \in \Omega_n^j \\ 0, \boldsymbol{x} \notin \Omega_n^j \end{cases} \ , \quad f^j(\boldsymbol{x}) = \sum_{n=1}^N f_n^j(\boldsymbol{x}) \quad (2)
$$

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

Proposition 3.1 (Forward pass). The NUFT of a PCF over a simplex in a mesh is

$$
F_n^j(\boldsymbol{k}) = \rho_n i^j \gamma_n^j S \quad (3)
$$

$$
S := \sum_{t=1}^{j+1} \frac{e^{-i\sigma_t}}{\prod_{l=1, l \neq t}^{j+1} (\sigma_t - \sigma_i)}, \quad \sigma_t := \boldsymbol{k} \cdot \boldsymbol{x}_t \quad (4)
$$

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:

$$C_n^j = \sqrt{\frac{(-1)^{j+1}}{2^j(j!)^2}det(\hat{B}_n^j)} \qquad (5)$$

$$\hat{B}_n^j := \begin{bmatrix} 0 & 1 & 1 & 1 & \cdots \\ 1 & 0 & d_{12}^2 & d_{13}^2 & \cdots \\ 1 & d_{21}^2 & 0 & d_{23}^2 & \cdots \\ 1 & d_{31}^2 & d_{32}^2 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \end{bmatrix} \qquad (6)$$

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

$$\gamma_n^j = \frac{C_n^j}{C_I^j} = j!C_n^j \qquad (7)$$

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

$$F^j(\boldsymbol{k}) = \sum_{n=1}^N F_n^j(\boldsymbol{k}) = \sum_n^N \rho_n i^j \gamma_n^j S \qquad (8)$$

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

Corollary 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. (3) is modified as follows:*

$$F_n^j(\boldsymbol{k}) = i^j \sum_{n'=1}^{N_n'} s_{n'} \gamma_{n'}^j \left(\frac{(-1)^j}{\prod_{l=1}^j \sigma_l} \right.$$
$$\left. + \sum_{t=1}^j \frac{e^{-i\sigma_t}}{\sigma_t \prod_{l=1,l\neq t}^j (\sigma_t - \sigma_l)} \right) \qquad (9)$$

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:

$$s_{n'}\gamma_{n'}^j = j! \det(J) = j! \det([\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_j]) \qquad (10)$$

Proof. Refer to [18]. □

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 $F_n^j(\boldsymbol{k})$.

Lemma 3.1 (Derivative of the content distortion factor). *The derivative of γ_n^j with respect to vertex coordinate $\boldsymbol{x}_p$ is*

$$\frac{\partial \gamma_n^j}{\partial \boldsymbol{x}_p} = \frac{(-1)^{j+1}/2^j}{\gamma_n^j} \sum_{\substack{m=1 \\ m\neq p}}^{j+1} A_{pm}\boldsymbol{D}_{pm} \qquad (11)$$

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

$$S_t := \frac{e^{-i\sigma_t}}{\prod_{l=1,l\neq t}^{j+1}(\sigma_t - \sigma_l)} \qquad (12)$$

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

$$\frac{\partial S}{\partial \boldsymbol{x}_p} = \left(-iS_p + \sum_{t=1,t\neq p}^{j+1} \frac{S_t + S_p}{\sigma_t - \sigma_p} \right) \boldsymbol{k} \qquad (13)$$

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

Proposition 3.2 (Backward pass). Following from Lemmas 3.1 and 3.2, the derivative of $F_n^j(\boldsymbol{k})$ with respect to a point $\boldsymbol{x}_p$ in the simplex element n is

$$\frac{\partial F_n^j(\boldsymbol{k})}{\partial \boldsymbol{x}_p} = \rho_n i^j \left(\Lambda\boldsymbol{k} + \Gamma \sum_{\substack{m=1 \\ m\neq p}}^{j+1} A_{pm}\boldsymbol{D}_{pm} \right) \qquad (14)$$

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

$$\Lambda := \gamma_n^j \left(-iS_p + \sum_{t=1,t\neq p}^{j+1} \frac{S_t + S_p}{\sigma_t - \sigma_p} \right) \qquad (15)$$

$$\Gamma := \frac{(-1)^{j+1}/2^j}{\gamma_n^j} S \qquad (16)$$

We provide a detailed derivation of Eqn. 14 as well as proofs of Lemmas 3.1 and 3.2 in Sec. A1 of the Appendix.

3.3. Deep Learning Architectures and Pipelines

We present the a schematic of the deep learning model-driven shape optimization (Sec. 4.2) in Fig. 2, and a schematic of the polygon segmentation network (PolygonNet) in Figs. 3 and 4. A detailed description of the architectures is presented in Appendix B.

4. Experiments

4.1. 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.
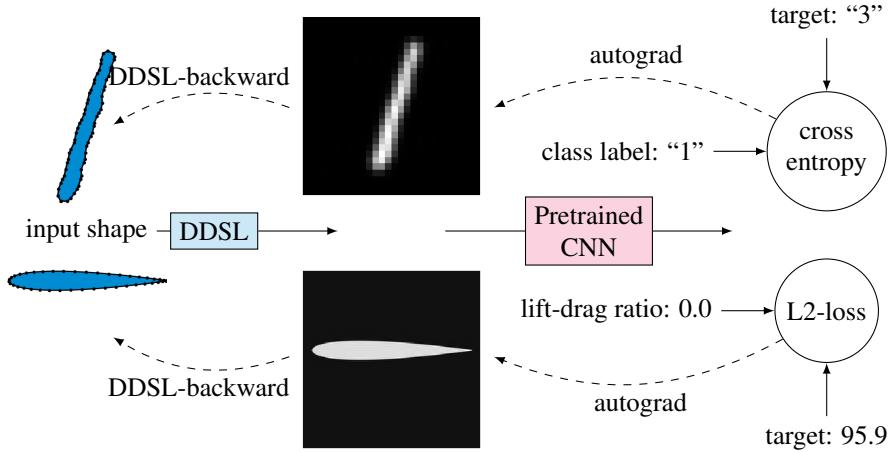
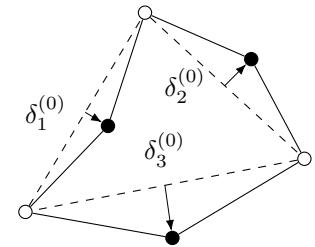Figure 2: Schematic of deep learning model driven shape optimization pipeline.

Figure 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.
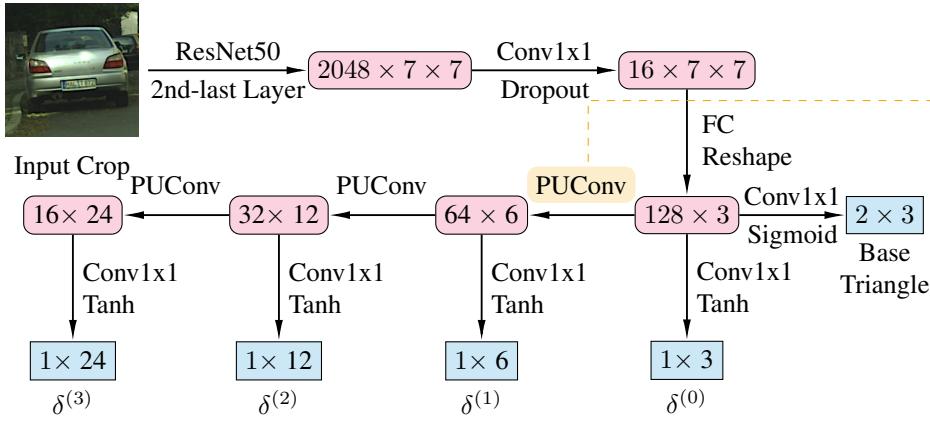

Figure 4: Schematic of the deep learning architecture for polygon segmentation (PolygonNet). All intermediate layers are followed by BatchNorm and ReLU. A Periodic Upsampling Convolution (PUConv) is used to generate vertex offsets (δ) at the consecutive level. For each level, we learn a learnable scale factor for all offsets.

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. 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. 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 run-time tests are shown in Fig. 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.
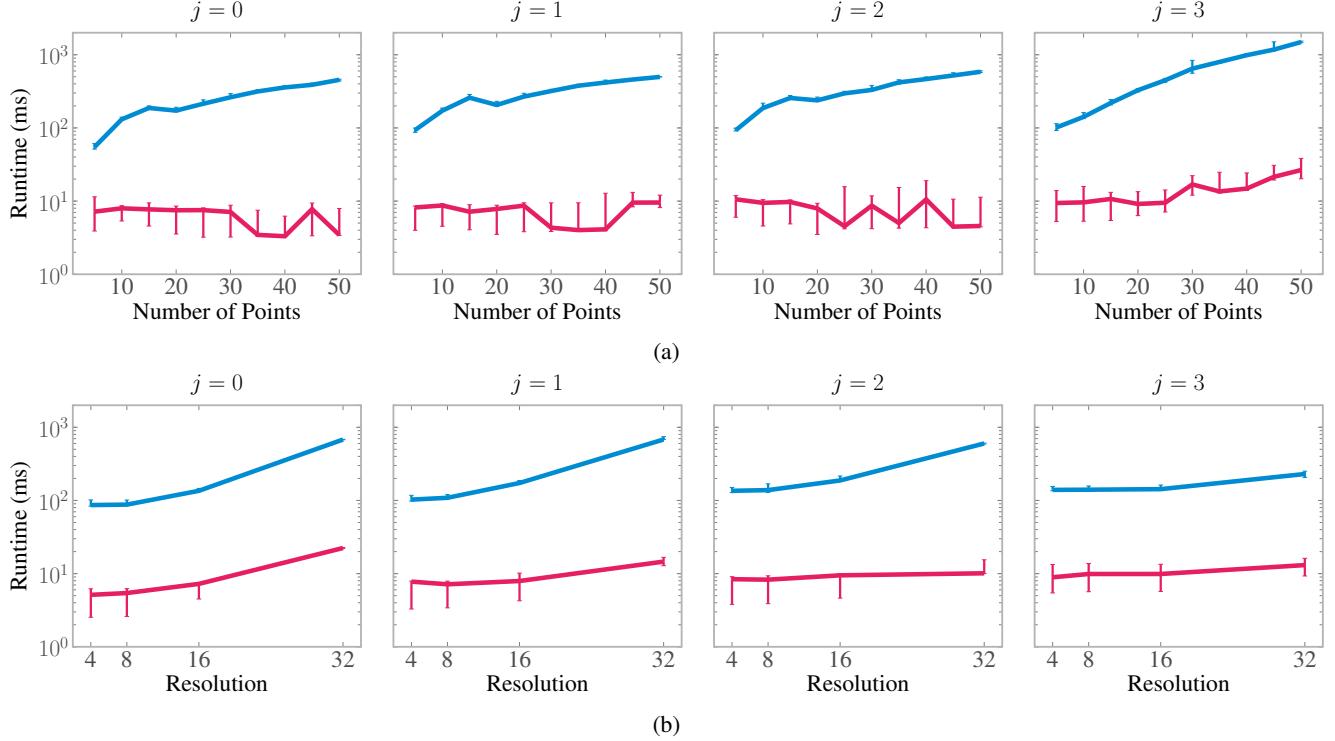
Figure 5: Comparison of the analytic (pink) and numeric (blue) derivative runtimes for the (a) mesh size and (b) resolution tests. All rasters are computed for a square cube, and resolution is per dimension.

4.2. 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 pre-scribed 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. A2. 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 precomputed physical quantities such as drag and lift coefficients at different angles of attack and 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
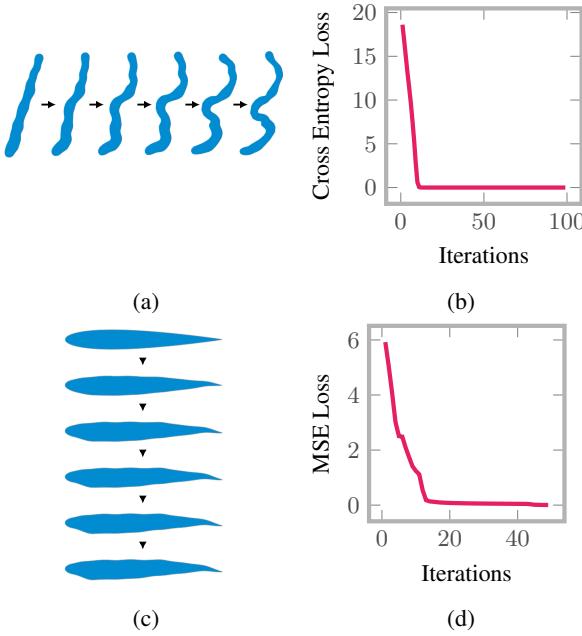
Figure 6: Optimization of (a), (b): a '1' from the MNIST dataset to a '3' by minimizing the cross-entropy between the input and the target class. (c), (d): the NACA 0012 airfoil (with an original lift-drag ratio of 0) to a lift-drag ratio of 95.9. The airfoil is set at an angle of attack of zero, and the Reynolds number is set to 1×10^6.

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. 6, 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 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.

4.3. 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 [4, 1].

Experiment Setup   For direct comparison with state-of-the-art, we follow the experiment setup of [4] 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 [5]. 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:

$$\mathcal{L}_{\text{mres}} = \sum_{i,res} ||D_{res}(G_\theta^{(i)}(x)) - D_{res}(y)||_1 \qquad (17)$$

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

$$\mathcal{L}_{\text{smooth}} = \frac{1}{n} \sum_j^n (\frac{A_j(G_\theta^{(3)}(x))}{\pi} - 1)^2 \qquad (18)$$

$$\mathcal{L} = \mathcal{L}_{\text{mres}} + \lambda \mathcal{L}_{\text{smooth}} \qquad (19)$$

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. 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 B3.

Results   We evaluate our model against state-of-the-art models and detail the results in Table 2, where we evaluate runtime on a single Titan X (Pascal) GPU. We provide a visual comparison in Fig. 7. Our model surpasses state of the art for class-averaged IoU. In particular, the simplicity of our network architecture is highlighted in Table 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

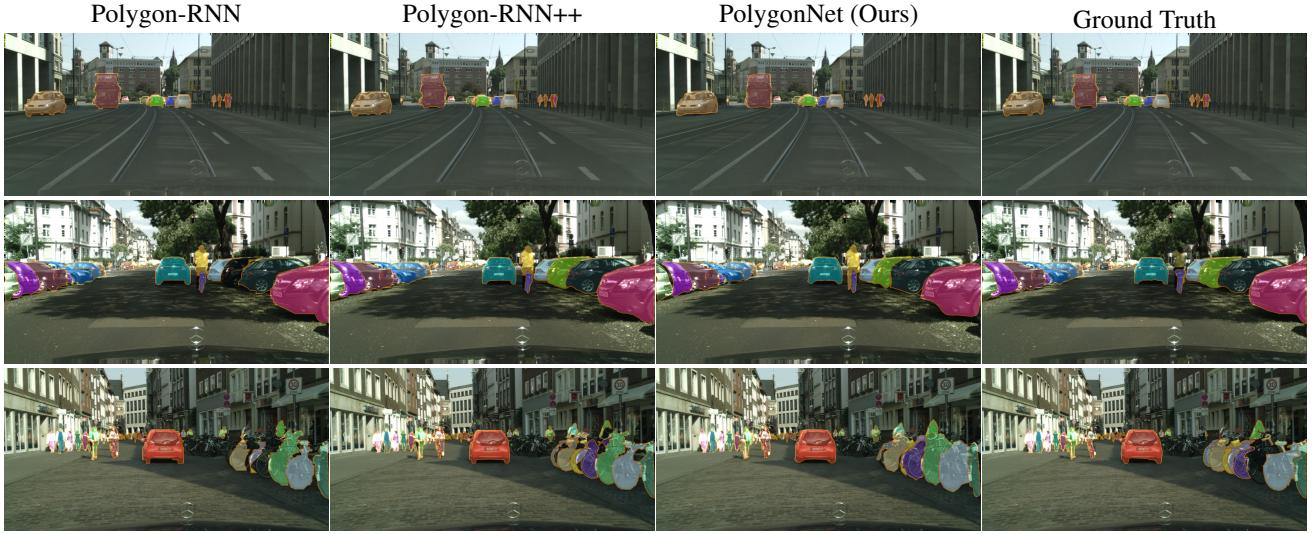| Polygon-RNN | Polygon-RNN++ | PolygonNet (Ours) | Ground Truth |

Figure 7: Visualization of image segmentation results. Ground-truth bounding boxes are given for all models to create image crops as inputs to the networks.

Model	Bicycle	Bus	Person	Train	Truck	Motorcycle	Car	Rider	Mean
SquareBox [4]	35.41	53.44	26.36	39.34	54.75	39.47	46.04	26.09	40.11
Dilation10 [47]	46.80	48.35	49.37	44.18	35.71	26.97	61.49	38.21	43.89
DeepMask [33]	47.19	69.82	47.93	62.20	63.15	47.47	61.64	52.20	56.45
SharpMask [34]	52.08	73.02	53.63	64.06	65.49	51.92	65.17	56.32	60.21
Polygon-RNN [4]	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: Comparison of Cityscape image segmentation IoU against baseline algorithms on test set.

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

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.

6. Acknowledgements

References

[1] David Acuna, Huan Ling, Amlan Kar, and Sanja Fidler. Efficient interactive annotation of segmentation datasets with polygon-rnn++. *arXiv preprint arXiv:1803.09693*, 2018. 3, 7, 8

[2] Davide Boscaini, Jonathan Masci, Emanuele Rodolà, and Michael Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 3189–3197, 2016. 2

[3] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017. 1

[4] Lluis Castrejon, Kaustav Kundu, Raquel Urtasun, and Sanja Fidler. Annotating object instances with a polygon-rnn. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5230–5238, 2017. 3, 7, 8

[5] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016. 7

[6] 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. 2

[7] 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. 2

[8] Angela Dai, Daniel Ritchie, Martin Bokeloh, Scott Reed, Jürgen Sturm, and Matthias Nießner. Scancomplete: Large-scale scene completion and semantic segmentation for 3d scans. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2018. 2

[9] 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*, pages 3150–3158, 2016. 3

[10] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016. 2

[11] Kyle Genova, Forrester Cole, Aaron Maschinot, Aaron Sarna, Daniel Vlasic, and William T Freeman. Unsupervised training for 3d morphable model regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8377–8386, 2018. 2

[12] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017. 3

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 3

[14] Oliver Hennigh. Automated design using neural networks and gradient descent. *arXiv preprint arXiv:1710.10352*, 2017. 3

[15] Jingwei Huang, Haotian Zhang, Li Yi, Thomas Funkhouser, Matthias Nießner, and Leonidas Guibas. Texturenet: Consistent local parametrizations for learning from high-resolution signals on meshes. *arXiv preprint arXiv:1812.00020*, 2018. 2

[16] Antony Jameson, L Martinelli, and NA Pierce. Optimum aerodynamic design using the navier–stokes equations. *Theoretical and computational fluid dynamics*, 10(1-4):213–237, 1998. 2

[17] Chiyu Max Jiang, Jingwei Huang, Karthik Kashinath, Prabhat, Philip Marcus, and Matthias Niessner. Spherical CNNs on unstructured grids. In *International Conference on Learning Representations*, 2019. 2, 4

[18] Chiyu Max Jiang, Dequan Wang, Jingwei Huang, Philip Marcus, and Matthias Niessner. Convolutional neural networks on non-uniform geometrical signals using euclidean spectral transformation. In *International Conference on Learning Representations*, 2019. 1, 4

[19] Evangelos Kalogerakis, Melinos Averkiou, Subhransu Maji, and Siddhartha Chaudhuri. 3d shape segmentation with projective convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6630–6639. IEEE, 2017. 2

[20] Angjoo Kanazawa, Shubham Tulsiani, Alexei A Efros, and Jitendra Malik. Learning category-specific mesh reconstruction from image collections. *arXiv preprint arXiv:1803.07549*, 2018. 2

[21] 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*, pages 5010–5019, 2018. 2

[22] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3907–3916, 2018. 2

[23] 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*, page 1278, 2009. 3

[24] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 2

[25] 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*, pages 3559–3568, 2018. 2

[26] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. In *SIGGRAPH Asia 2018 Technical Papers*, page 222. ACM, 2018. 2

[27] Yi Li, Haozhi Qi, Jifeng Dai, Xiangyang Ji, and Yichen Wei. Fully convolutional instance-aware semantic segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4438–4446. IEEE, 2017. 3

[28] Shichen Liu, Weikai Chen, Tianye Li1, and Hao Li. Soft rasterizer: Differentiable rendering for unsupervised single-view mesh reconstruction. *arXiv preprint arXiv:1901.05567*, 2019. 2

[29] 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*, pages 3431–3440, 2015. 3

[30] Matthew M Loper and Michael J Black. Opendr: An approximate differentiable renderer. In *European Conference on Computer Vision*, pages 154–169. Springer, 2014. 2

[31] Anton Lundberg, Per Hamlin, Davangere Shankar, Alexander Broniewicz, Tim Walker, and Christoffer Landström. Automated aerodynamic vehicle shape optimization using neural networks and evolutionary optimization. *SAE International Journal of Passenger Cars-Mechanical Systems*, 8(2015-01-1548):242–251, 2015. 3

[32] 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*, pages 922–928. IEEE, 2015. 2

[33] Pedro O Pinheiro, Ronan Collobert, and Piotr Dollár. Learning to segment object candidates. In *Advances in Neural Information Processing Systems*, pages 1990–1998, 2015. 8

[34] Pedro O Pinheiro, Tsung-Yi Lin, Ronan Collobert, and Piotr Dollár. Learning to refine object segments. In *European Conference on Computer Vision*, pages 75–91. Springer, 2016. 8

[35] Olivier Pironneau. On optimum design in fluid mechanics. *Journal of Fluid Mechanics*, 64(1):97–110, 1974. 2

[36] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 1(2):4, 2017. 2

[37] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, pages 5099–5108, 2017. 2

[38] Elad Richardson, Matan Sela, Roy Or-El, and Ron Kimmel. Learning detailed face reconstruction from a single image. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5553–5562. IEEE, 2017. 2

[39] 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*, pages 234–241. Springer, 2015. 3

[40] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 190–198. IEEE, 2017. 2

[41] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015. 2

[42] Ayush Tewari, Michael Zollhöfer, Pablo Garrido, Florian Bernard, Hyeongwoo Kim, Patrick Pérez, and Christian Theobalt. 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*, pages 2549–2559, 2018. 2

[43] Ayush Tewari, Michael Zollhofer, Hyeongwoo Kim, Pablo Garrido, Florian Bernard, Patrick Perez, and Christian Theobalt. Mofa: Model-based deep convolutional face autoencoder for unsupervised monocular reconstruction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3715–3724, 2017. 2

[44] Panqu Wang, Pengfei Chen, Ye Yuan, Ding Liu, Zehua Huang, Xiaodi Hou, and Garrison Cottrell. Understanding convolution for semantic segmentation. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1451–1460. IEEE, 2018. 3

[45] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in Neural Information Processing Systems*, pages 82–90, 2016. 2

[46] Li Yi, Hao Su, Xingwen Guo, and Leonidas J Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2282–2290, 2017. 2

[47] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016. 8

[48] Andy Zeng, Shuran Song, Matthias Nießner, Matthew Fisher, Jianxiong Xiao, and Thomas Funkhouser. 3dmatch: Learning local geometric descriptors from rgb-d reconstructions. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 199–208. IEEE, 2017. 2

[49] 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*, page 1903, 2018. 3

Appendix

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

A. Mathematical Derivations

A1. NUFT Derivative Derivation

Proof of Lemma 3.1. Using Jacobi's formula and chain rule,

$$\frac{\partial \gamma_n^j}{\partial \boldsymbol{x}_p} = \frac{(-1)^{j+1}}{2\sqrt{2^j(-1)^{j+1}det(\hat{B}_n^j)}} tr\left(adj(\hat{B}_n^j)\frac{\partial \hat{B}_n^j}{\partial \boldsymbol{x}_p}\right) \tag{20}$$

$$= \frac{(-1)^{j+1}/2^j}{2\gamma_n^j} \sum_{m=1}^{j+2}\sum_{n=1}^{j+2} \tilde{A}_{mn}\tilde{D}_{nm} \tag{21}$$

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,

$$\tilde{D} = \begin{bmatrix} 0 & \dots & 0 & 0 & 0 & \dots \\ \vdots & \ddots & \vdots & \vdots & \vdots & \\ 0 & \dots & 0 & \tilde{D}_{p,p+1} & 0 & \dots \\ 0 & \dots & \tilde{D}_{p+1,p} & 0 & \tilde{D}_{p+1,p+2} & \dots \\ 0 & \dots & 0 & \tilde{D}_{p+2,p+2} & 0 & \dots \\ \vdots & & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{22}$$

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

$$\tilde{D}_{p+1,n} = \frac{\partial d_{p,n-1}^2}{\partial \boldsymbol{x}_p} = 2(\boldsymbol{x}_p - \boldsymbol{x}_{n-1}) \tag{23}$$

$$\tilde{D}_{m,p+1} = \frac{\partial d_{m-1,p}^2}{\partial \boldsymbol{x}_p} = 2(\boldsymbol{x}_p - \boldsymbol{x}_{m-1}) \tag{24}$$

It follows that the double summation term in Eqn. 21 simplifies to

$$\sum_{m=1}^{j+2}\sum_{n=1}^{j+2} \tilde{A}_{mn}\tilde{D}_{nm} = 2 \sum_{\substack{m=2 \\ m \neq p+1}}^{j+2} \tilde{A}_{p+1,m}\tilde{D}_{p+1,m} \tag{25}$$

For clarity and ease of implementation, we modify the indexing in Eqn. 25 and the derivative of the content distortion factor is finally

$$\frac{\partial \gamma_n^j}{\partial \boldsymbol{x}_p} = \frac{(-1)^{j+1}/2^j}{\gamma_n^j} \sum_{\substack{m=1 \\ m \neq p}}^{j+1} A_{pm}\boldsymbol{D}_{pm} \tag{26}$$

□

Proof of Lemma 3.2. By the sum rule,

$$\frac{\partial S}{\partial \boldsymbol{x}_p} = \sum_{t=1}^{j+1} \frac{\partial S_t}{\partial \boldsymbol{x}_p} \tag{27}$$

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

$$\frac{\partial S_t}{\partial \boldsymbol{x}_p} = \frac{1}{\left(\prod_{l=1,l\neq p}^{j+1}(\sigma_p - \sigma_l)\right)^2}\boldsymbol{k}$$
$$\left[\left(\prod_{l=1,l\neq p}^{j+1}(\sigma_p - \sigma_l)\right)\left(-ie^{-i\sigma_p}\right)\right.$$
$$\left. +e^{-i\sigma_p}\left(\frac{\partial}{\partial \boldsymbol{x}_p}\left(\prod_{l=1,l\neq p}^{j+1}(\sigma_p - \sigma_l)\right)\right)\right] \tag{28}$$

$$= -\frac{e^{-i\sigma_p}}{\prod_{l=1,l\neq p}^{j+1}(\sigma_p - \sigma_l)}\left(i + \sum_{q=1,q\neq p}^{j+1}\frac{1}{\sigma_p - \sigma_q}\right)\boldsymbol{k} \tag{29}$$

For $t \neq p$,

$$\frac{\partial S_t}{\partial \boldsymbol{x}_p} = \frac{\partial}{\partial \boldsymbol{x}_p}\left(\frac{e^{-i\sigma_t}}{(\sigma_t - \sigma_1)...(\sigma_t - \sigma_p)...(\sigma_t - \sigma_{j+1})}\right) \tag{30}$$

$$= \left(\frac{e^{-i\sigma_t}}{(\sigma_t - \sigma_1)...(\sigma_t - \sigma_{p-1})(\sigma_t - \sigma_{p+1})\atop ...(\sigma_t - \sigma_{j+1})}\right)$$
$$\left(\frac{\partial}{\partial \boldsymbol{x}_p}\left(\frac{1}{\sigma_t - \sigma_p}\right)\right) \tag{31}$$

$$= \left(\frac{e^{-i\sigma_t}}{(\sigma_t - \sigma_1)...(\sigma_t - \sigma_{p-1})(\sigma_t - \sigma_{p+1})\atop ...(\sigma_t - \sigma_{j+1})}\right)$$
$$\left(\frac{1}{(\sigma_t - \sigma_p)^2}\boldsymbol{k}\right) \tag{32}$$

$$= \frac{e^{-i\sigma_t}}{\prod_{l=1,l\neq t}^{j+1}(\sigma_t - \sigma_l)}\left(\frac{1}{\sigma_t - \sigma_p}\right)\boldsymbol{k} \tag{33}$$

Thus,

$$\frac{\partial S}{\partial \boldsymbol{x}_p} = \sum_{t=1}^{j+1} \frac{\partial S_t}{\partial \boldsymbol{x}_p} \tag{34}$$

$$= \left(\sum_{t=1,t\neq p}^{j+1} \left(\frac{e^{-i\sigma_t}}{\prod_{l=1,l\neq t}^{j+1}(\sigma_t - \sigma_l)} \left(\frac{1}{\sigma_t - \sigma_p} \right) \right) \right.$$
$$\left. - \frac{e^{-i\sigma_p}}{\prod_{l=1,l\neq p}^{j+1}(\sigma_p - \sigma_l)} \left(i + \sum_{q=1,q\neq p}^{j+1} \frac{1}{\sigma_p - \sigma_q} \right) \right) \boldsymbol{k} \tag{35}$$

$$= \left(-i \frac{e^{-i\sigma_p}}{\prod_{l=1,l\neq p}^{j+1}(\sigma_p - \sigma_l)} + \sum_{t=1,t\neq p}^{j+1} \frac{1}{\sigma_t - \sigma_p} \right.$$
$$\left. \left[\frac{e^{-i\sigma_t}}{\prod_{l=1,l\neq t}^{j+1}(\sigma_t - \sigma_l)} + \frac{e^{-i\sigma_p}}{\prod_{l=1,l\neq p}^{j+1}(\sigma_p - \sigma_l)} \right] \right) \boldsymbol{k} \tag{36}$$

$$= \left(-iS_p + \sum_{t=1,t\neq p}^{j+1} \frac{S_t + S_p}{\sigma_t - \sigma_p} \right) \boldsymbol{k} \tag{37}$$

$\square$

Derivation of Eqn. 14. Using the product rule,

$$\frac{\partial F_n^j(\boldsymbol{k})}{\partial \boldsymbol{x}_p} = \rho_n i^j \left(\frac{\partial \gamma_n^j}{\partial \boldsymbol{x}_p} S + \frac{\partial S}{\partial \boldsymbol{x}_p} \gamma_n^j \right) \tag{38}$$

We obtain Eqn. 14 by substituting Eqns. 11 and 13 into Eqn. 38. $\square$

A2. 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:

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

B. Network Architecture and Training Details

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

B1. 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

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 4: Network architecture notation list.

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.

B2. Airfoil

We use ResNet-50 [13] 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 $\rightarrow$ append Reynolds number and angle of attack $\rightarrow$ FC(1002, 512) + BN + ReLU $\rightarrow$ FC(512, 64) + BN + ReLU $\rightarrow$ 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.

B3. 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. 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. 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.

C. Additional Computational Efficiency Tests

In addition to the computational speed benchmarks in Fig. 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 5: 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 6: 3D Computational speed (tri-mesh w/ 1300 faces).

D. 3D Geometric Applications

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

D1. 3D Rotational Pose Estimation

In Fig. 8, 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:

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

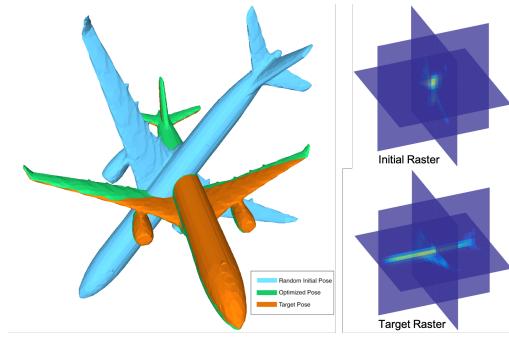where D_{32} is the rasterization operator at resolution 32^3 and V_{tg} is the target mesh.

Figure 8: 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.

D2. Single Image Mesh Estimation

In Fig. 9, we evaluate our method in the context of 3D deep learning. Our model consists of an image encoder from ResNet18, spherical convolutions [17] 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. 7).

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 7: Evaluation resultsn($\times 10^{-2}$).
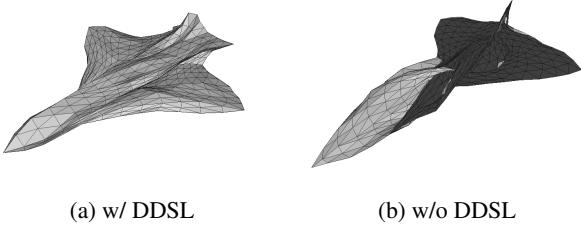

| (a) w/ DDSL | (b) w/o DDSL |

Figure 9: Qualitative visualization of generated samples.

E. Additional 3D Visualizations

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

Input Triangular Mesh

Rasterize Surface Mesh (j=2)
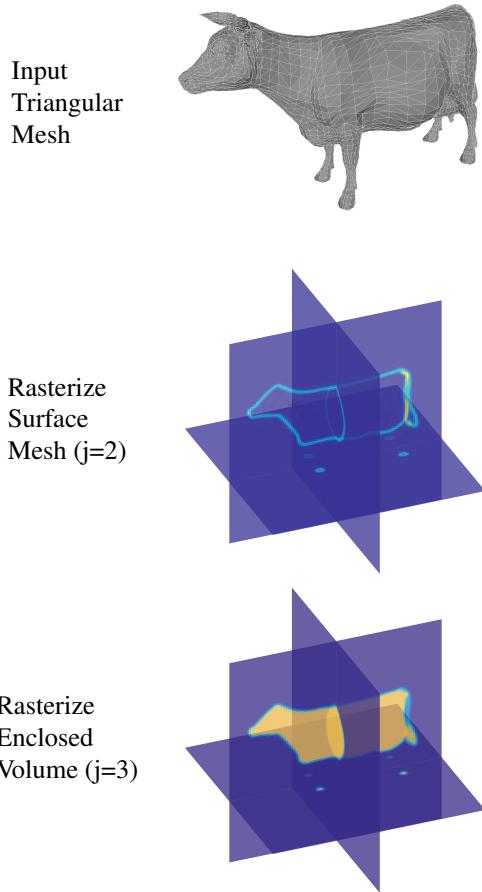
Rasterize Enclosed Volume (j=3)

Figure 10: 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.