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 besided 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 $t = p$,

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

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

For $t \neq p$,

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

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

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

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

Thus,

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

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

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

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

□

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. □

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 utilze 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.

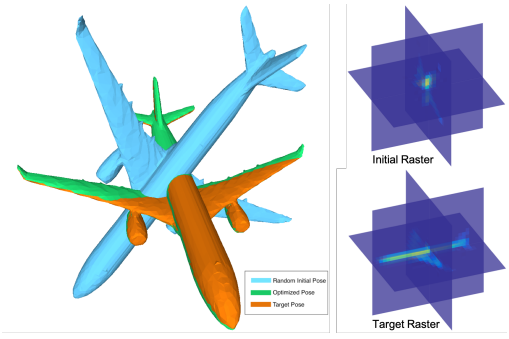


Figure 8: Mesh pose and rasters before and after opt.

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.

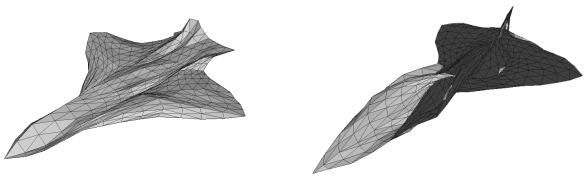
D2. Single Image Mesh Estimation

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.

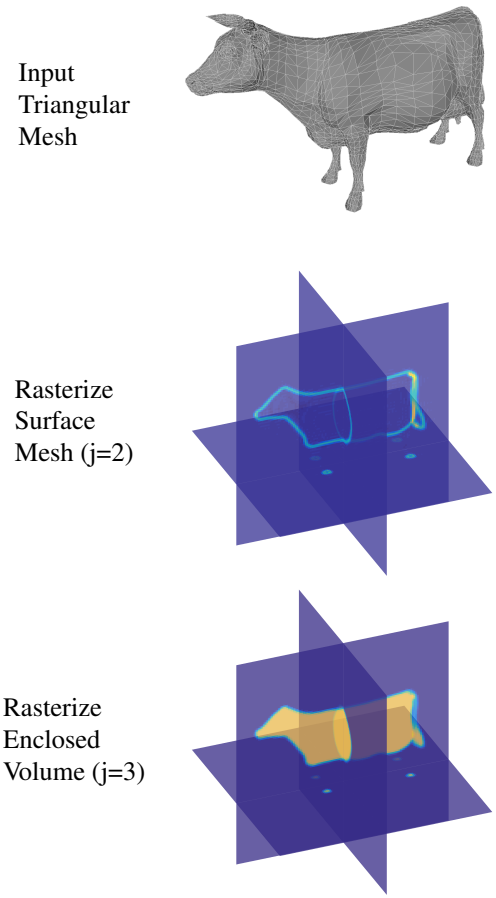


Figure 10: In this example above, the input is a watertight triangluar mesh represented by vertices and faces. It can be rasterized in-situ in a 3-dimensional grid differentiably. The value is approximately 0 or 1 indicating signal densities.