Accurate Differential Operators for Hybrid Neural Fields

Supplementary Material

A. Experimental details

In this section, we provide the implementation details for our experiments described in Section 4.

A.1. Dataset

We perform pre-training on shapes from the FamousShape dataset [4]. We filter out shapes with non-watertight meshes or incorrectly oriented normals. This is because non-watertight meshes do not admit a valid SDF and in order to compute the correct ground truth, we require meshes with correct normals. This gave us a set of 15 shapes. We further center the meshes at the origin and normalize them to lie inside the $[-1, 1]^3$ cube.

A.2. Pre-training

The inputs for our experiments are the pre-trained hybrid neural SDFs of the shapes. In this section, we present details about how we obtain the pre-trained models. First, we provide a description and architectural details of our hybrid neural fields:

- Instant NGP [11]: We retained the original architecture from the paper. We implemented our models using `tiny-cuda-nn` [10].
- Dense Grid: A grid-based neural field with dense feature grids, as discussed in Müller *et al*. [11]. We use a multi-resolution grid with 4 levels, starting from a minimum resolution of 16 up to a maximum resolution of 256.
- Tri-Plane [2]: Instead of volumetric grids, they consist of 3 planar grids (one each for XY, YZ, and XZ planes), with a feature embedding residing on each grid point. For a given query point, the features are combined using bi-linear interpolation on each plane and then further summed together. Finally, the feature is passed through an MLP to obtain the output. We used planes with a resolution of 512, feature embeddings of size 32, and an MLP with 2 hidden layers of size 128.

We follow the same data sampling procedure for training neural SDFs as described by Müller *et al*. [11] for training the Instant NGP. We trained all models for 10^4 steps using the Adam [6] optimizer with an initial learning rate of $1\mathrm{e}{-3}$ and reduced the learning rate by a factor of 0.2 every 5 steps.

A.3. Post hoc operator

In this section, we provide details for the hyperparameter selection procedure used for our post hoc polynomial-fitting operator. We used a fixed value of 256 for k. For the value of σ, we selected the best value using telescopic search in two levels: the first sweep is conducted over $10^i : -5 \leq i \leq 1$, after which we zoom in to the interval bounded by the best value, σ_1 and its best neighbor σ_2. Assuming here for simplicity that $\sigma_1 < \sigma_2$, we then conduct a sweep over 20 values taken at uniform intervals from $[\sigma_1, \sigma_2]$.

Baselines. We compare our polynomial-fitting operator with automatic differentiation and finite difference for computing surface normals and mean curvatures of the shapes. For automatic differentiation, we directly query the network using PyTorch's [12] automatic differentiation toolkit. For the finite difference operators, we used a centered difference approach, sampling local axis-aligned neighbors of the query point and using them to compute the operator. The finite difference operator had a hyperparameter h for the stencil size. In essence, it gives the size of the finite difference grid cell, if we were to set up a global grid for computing finite differences . We selected this hyperparameter by sweeping over the set $\{\frac{2}{2^i} : 5 \leq i \leq 9\}$. Here 2^i is analogous to the resolution of the global finite difference grid.

A.4. Fine-tuning

As discussed in Section 3.3, we train an ensemble of models where each model is supervised with a different version of the smoothed gradient operator, $\hat{\nabla}_{\mathbf{x}}$ characterized by the amount of smoothing it imposes. For fine-tuning based on polynomial-fitting derivatives, we ensemble using σ values taken uniformly from the interval $[1\mathrm{e}-3, 1\mathrm{e}-2]$ at steps of $5\mathrm{e}-3$. For finite-difference-based fine-tuning we ensemble using stencil sizes from the set $\{2^i : 5 \leq i \leq 9\}$. We fine-tune all models for 4000 steps with a constant learning rate of $2\mathrm{e}-3$, using the Adam optimizer [6].

Baselines. We have also compared our fine-tuning approaches with neural fields trained using eikonal regularization [1, 5]. For the commonly-used eikonal loss that uses autodiff gradients, we follow the same training parameters as previously described in pre-training (Appendix A.2). We just add the eikonal loss to the loss function with a weight of 10^{-3} (selected by sweeping over $\{1, 10^{-1}, 10^{-3}\}$). For the finite differences-based variant of the eikonal loss [7], we found that the same weight gave the best result, and for the size of the finite-difference stencil (ϵ in Li *et al*.) we selected the value for each shape by sweeping over the set $\{\frac{2}{2^i} : 5 \leq i \leq 10\}$.

A.5. Evaluation

To compare our approach against the baselines, we generate ground truth surface normals and mean curvatures using

the meshes of the shapes. First, we compute the vertex normals and discrete mean curvatures [9] of the shapes from the meshes. Next, we sample 2^{18} points on the surface of the meshes. We interpolate the vertex normals and mean curvatures to each point using barycentric interpolation from the mesh vertices. This set of points, their computed normals, and mean curvatures become the ground truth used in our evaluations. The metrics used for our evaluations have been described in Section 4 (under *Metrics*).

B. Additional Results

B.1. Accuracy analysis

In Section 4, we reported the results for the accuracy of our operators. In this section, we provide the full results for the accuracy analysis of our operators and our fine-tuning approach on the FamousShape dataset [4]. Table 2 shows comparisons between our post hoc operator and the baselines on Instant NGP while Table 3 shows how our best fine-tuning approach, i.e., fine-tuning with polynomial-fitting gradients. We show our results for Dense Grid models in Tables 4 and 5. We can observe that we obtain more accurate gradients than the baselines. This also shows that the artifacts that we observed in the case of Instant NGP were not solely a result of its hash encoding. Finally, results presented in Table 6 show that even on a significantly different hybrid architecture like Tri-plane, our operators can provide more accurate surface normals and mean curvatures. At the time of writing, our Tri-plane implementation did not have support for higher-order derivatives through autodiff derivatives. Hence, we were unable to show fine-tuning results.

B.2. Results on images

We also show the benefits of our approaches on a different modality, specifically images. We train an Instant NGP [11] model on an image and evaluate its derivatives using our proposed approaches. For pre-training our model, we used a relative L2 loss and trained using the Adam optimizer with a learning rate of 0.01. For fine-tuning, we use MSE loss for $\mathcal{L}_{\text{con}}$, and weighted weighted $\mathcal{L}_{\text{grad}}$ by 10^{-3}, and trained using a learning rate of 0.02.

Figure 1 shows our results. For reference, we use the derivatives obtained using Sobel filtering, similar to Sitzmann *et al*. [13]. Firstly, we observe that our fine-tuning approach preserves the initial image, with a minor drop in the PSNR over the pre-trained image. We also compare the accuracy of derivatives using a weighted mean angular error, where the weights are the reference gradient magnitudes. This is because image gradients are usually more important in regions with high gradient magnitudes (the edges). Our post hoc operator gives more accurate gradients than finite differences. We also observe that autodiff gradients obtained after our fine-tuning approach are more accurate than naively applying autodiff to the pre-trained signal.

B.3. Runtime Analysis

We compare the wall time of our local polynomial-fitting approach with finite difference and autodiff operators. For our operator, we use $k = 256$. We computed the mean and standard deviation of wall-time required by all methods on a single query point, averaged over 7 runs each running 1000 instances of the method.

Method	Time (μs)
AD	1520 ± 12.9
FD	509 ± 91.1
Ours	459 ± 16.3

Table 1. **Runtime Analysis**

Table 1 summarizes the results. We found that our operator performs competitively in terms of runtime compared to finite difference (FD) and autodiff (AD) gradient operators. All these methods were benchmarked using an Instant NGP model [11].

Our proposed fine-tuning approach takes $\sim$700s to reach $\sim$90% of the reported performance. Although vanilla Instant NGP can reach equivalent reconstruction loss in $\sim$20s, its derivatives are nowhere near as accurate as our approach even after $\sim$1000s worth of training. That said, if training cost is a concern, we can trade off training cost for test-time compute using our post hoc operator. Also, note that ours is a naive implementation which can be sped up with engineering tricks (e.g., sharing local neighborhoods or sampling fewer points, trading off derivative accuracy).

B.4. Comparing PDE simulation with INSR [3]

While we have used the framework of INSR for our PDE simulation experiments, a direct apples-to-apples comparison with INSR is not possible due to INSR utilizing a different architecture (SIREN). As we discuss later (in Appendix E), while SIREN also suffers from inaccurate derivatives, the nature and cause of those accuracies differ significantly from the high-frequency noise that we claim to address. Tackling SIREN's derivative errors would require an altogether different approach that we hope to address in future work.

However, to show how our approach with hybrid neural fields stands relative to a current state-of-the-art approach like INSR, we show a comparison between the errors of our approach and INSR in the same setup as discussed in Section 5.3. Figure 2 shows our results. We can see that while INSR performs better, using our approach to compute derivatives with hybrid neural fields allows hybrid neural fields to perform competitively against INSR, which is not possible with autodiff derivatives.

C. Comparison to Marching Cubes

As discussed in Section 3.2, one other alternative for computing derivatives is by directly extracting the mesh using

Method	Wt. Ang ↓
Autodiff	28.07°
AD w/ fine-tuning	**18.40°**
Finite Differences	7.01°
Polynomial-fitting	**4.64°**

Figure 1. **Results on images.** We show the application of our operators on a hybrid neural field trained to represent an image. For reference, we use the image derivative obtained using Sobel filtering, similar to Sitzmann *et al*. [13]. We compare the image gradient obtained using our post hoc and fine-tuning approaches with the baselines. For the zeroth-order signal, we show the PSNR (inset) which shows that fine-tuning preserves the initial image. For the image gradient, we show the weighted mean angular error, weighted by the reference gradient magnitude. Applying autodiff after our fine-tuning approach leads to more accurate gradients than direct autodiff. Using our post hoc operator also leads to more accurate gradients than finite differences.

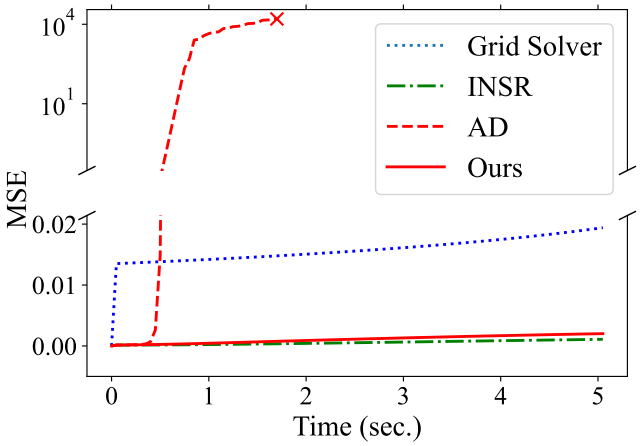


Figure 2. **Comparison to INSR [3].** While INSR performs better, our approach allows hybrid neural fields to perform competitively, which is not possible when using autodiff gradients directly.

the Marching Cubes algorithm [8]. While mesh extraction with Marching Cubes can take time, this cost can be amortized over multiple queries for derivatives using the extracted mesh. Hence, for a fair runtime comparison to Marching Cubes, we compare the runtime of our operator with Marching Cubes on a larger point set of size 2^{18} sampled uniformly from a 3D shape, in this case, the Stanford Bunny. Since the points sampled may not always lie on the extracted mesh for Marching Cubes, we compute the normals at the closest on-surface point. Figure 3 illustrates how getting comparably accurate derivatives requires running Marching Cubes at a high grid resolution (512) which takes up almost $\mathbf{15}\times$ the time taken by our approach. We can try to save time by running marching cubes at a lower resolution, however, this leads to inaccurate derivatives, resulting in almost $\mathbf{7}\times$ the error incurred by our approach. Thus, getting accurate derivatives from Marching Cubes is quite expensive compared to our approach, and can become increasingly prohibitive in applications like physical simulation, where frequent derivative queries may be required from an evolving underlying signal.

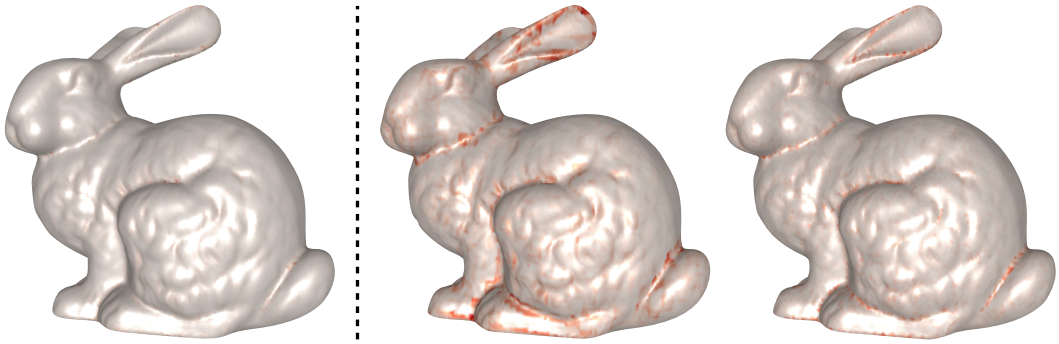
D. Application Setups

In this section, we describe the details of the experiential setup used in each application described in Section 5.

Rendering. In our rendering experiments (Section 5.1) for both shapes, we used the Instant NGP model [11]. The training and hyperparameter selection were done using the same process as described in Appendix A.2 and Appendix A.4 respectively. For our polynomial-fitting operator, we use $\sigma = 0.03$ and $k = 256$ for the sphere and $\sigma = 0.002$ and $k = 256$ for the Armadillo, selected using telescopic search. For the results of the fine-tuning approach, we queried all models in the ensemble and selected the best render after visual comparison. For the finite difference operator, we selected a stencil size of $\frac{2}{32}$ for the sphere and $\frac{2}{512}$ by conducting a sweep as described in Appendix A.3.

Simulating Collisions. For our experiments on simulating collisions (Section 5.2), the hybrid neural SDF of the sphere was a Dense Grid model. The model had a minimum resolution of 16, a maximum resolution of 128, and consisted of 4 grid levels. For our polynomial-fitting operator, we used $\sigma = 0.03, k = 64$, selected using telescopic search.

PDE simulation. For the PDE simulation experiment (Section 5.3), we used the same model architecture as the collision experiments, with a minimum resolution of 16, a maximum resolution of 128, and 4 grid levels. We modify the code shared by the authors of INSR [3] to solve the 2D advection problem. However, we retain the data sampling and the training strategies used by the authors such as uniform sampling of the domain for training the implicit field,



1.35°, 0.92 sec.
Ours

9.11°, 0.35 sec.
64^3 grid

3.02°, 13.45 sec.
512^3 grid

Figure 3. **Marching Cubes for derivatives.** Mean angle error and time required by Marching Cubes to compute surface normals (first-order derivative) on the Bunny shape (Red denotes error). This approach can be expensive ($\mathbf{15}\times$ time) for obtaining accurate surface normals. Reducing the grid resolution can reduce time but trades off accuracy for efficiency ($\mathbf{7}\times$ error). Comparatively, our approach provides accurate normals efficiently.

and early stopping during optimization. Our initial condition is a Gaussian pulse, given by:

$$f(x, y) = e^{-\left(\frac{(x-\mu_1)^2+(y-\mu_2)^2}{2\sigma^2}\right)} \tag{1}$$

where $\mu_1 = -0.6, \mu_2 = -0.6, \sigma = 0.1$. We run our simulations in a square of side length 2 centered at (1, 1). For the boundary conditions, we use the Dirichlet boundary condition, i.e., the field becomes 0 at the boundary, the same as INSR [3] in their 1D advection setting. Other details are shared in Section Section 5.3.

E. Effectiveness on a non-hybrid neural field (SIREN [13])

While our approaches are not tied to a particular architecture, they can only address the high-frequency noise in neural fields. As we illustrated in Section 3, signals learned by hybrid neural fields like Instant NGP [11] are abundant in such high-frequency noise.

We also investigated if similar kinds of artifacts arise in non-hybrid networks, specifically SIREN [13]. We trained a SIREN network with $\omega_0 = 30$ and two hidden layers of size 128 each. Our first observation was that even for SIREN, derivatives, particularly higher-order derivatives, suffer from inaccuracies. However, unlike hybrid neural fields, we found that SIREN has a lower degree of high-frequency noise. The errors in SIREN seem to stem from low-frequency errors. Figure 4 illustrates this phenomenon. We observed similar trends for different values of ω_0 ($\omega_0 \in \{20, 50\}$) and with varying hidden sizes (over $\{64, 256\}$)

Using our operators to compute the spatial derivatives of SIREN only helps to a limited degree (Figure 5). The observations on gradient are not very interesting as the autodiff gradient itself for SIREN is quite good and our operator leads to minor improvements. However, when computing the curvature (Laplacian), we observe that while autodiff curvatures are quite inaccurate, our operator can recover some reasonable values from the field, but noticeable errors remain. We believe that while our operator can address the high-frequency noise component in the underlying field, it is not able to overcome the low-frequency errors in SIREN.

To conclude, our preliminary experiments reveal that neural fields learned by SIREN have a lower degree of high-frequency noise and higher low-frequency errors compared to hybrid neural fields. As a result, while our operators can deal with high-frequency noise, low-frequency errors still result in inaccurate derivatives. Dealing with these low-frequency errors would require an altogether different approach and would be an interesting direction for future work.

F. Training hybrid neural fields with accurate autodiff normals from scratch

As discussed in Section 3.3, we can also use our proposed loss (Eq. (3)) for training hybrid neural fields from scratch. For this, we first train the model, F_Θ for s (> 0) steps as a *warm-start* phase. This allows the model to learn a good initial estimate of the zeroth-order signal. Next, we train using our loss (Eq. (3)) for $(n - s)$ steps. For computing the smoothed gradient operator, we require M which is a hybrid neural field with a good initial fit over the zeroth-order signal. In this case, we set M as the frozen weights of F_Θ at the end of s training steps.

Intuitively, if M fits the zeroth-order signal well, the smoothed gradient operator would be more accurate, leading to a more accurate supervision signal for $\mathcal{L}_{\text{grad}}$. We want s to be large enough so that we have a reasonably good fir with M. Selecting a very small s can lead to a poor fit and unstable optimization in the next stage. On the

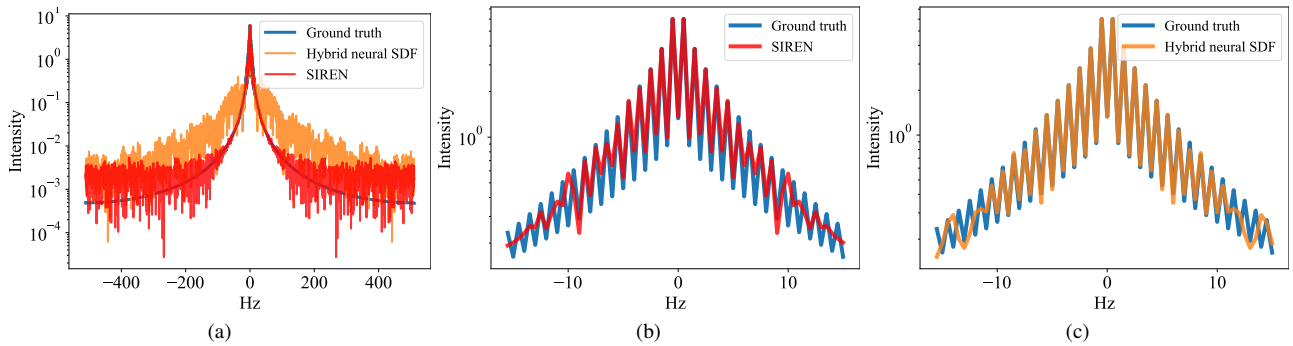


Figure 4. **Fourier spectrum of SIREN Vs. hybrid neural SDF.** Computed over a 1D slice (shown in Figure 5) of the SDF of a 2D circle. Note the lower degree of high-frequency noise compared to the hybrid neural SDF. Further zooming in (Figures 4b and 4c) to visualize the low-frequency components reveals the low-frequency errors in SIREN. Comparatively, the hybrid neural SDF more accurately captures the lower frequencies.

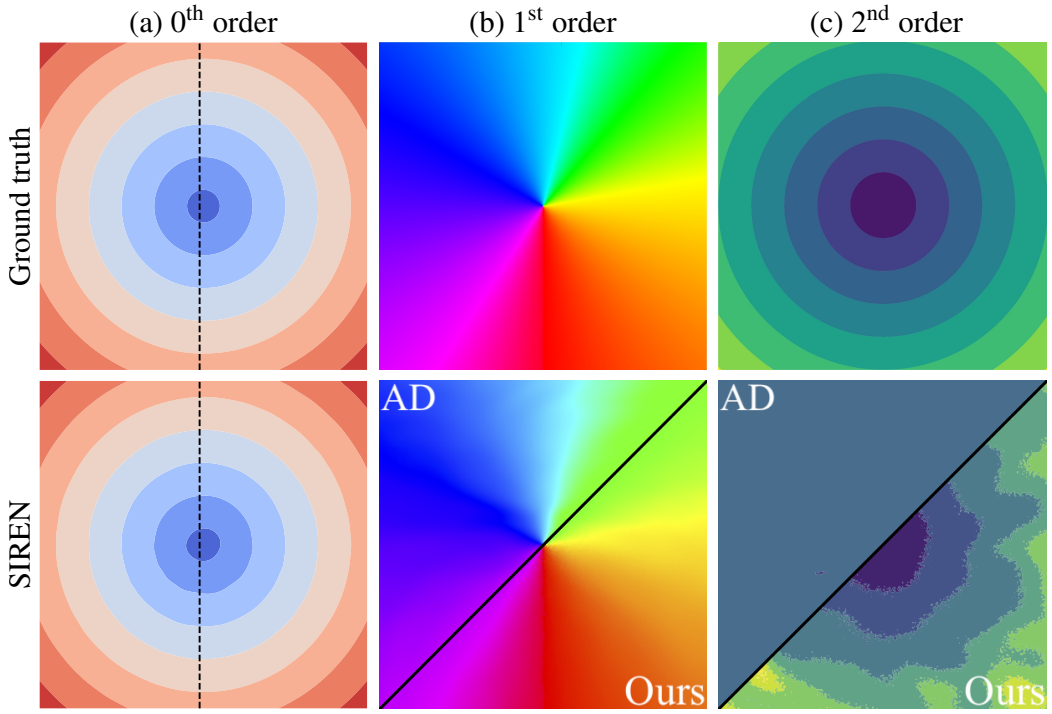
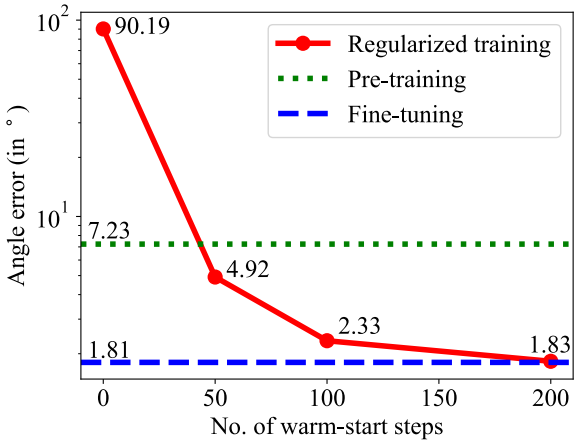


Figure 5. **Differential operators of SIREN.** SIREN trained on the SDF of a circle in 2D. While the first-order operator (spatial gradient) for SIREN is quite accurate, the second-order operator (or the Laplacian) exhibits large errors. Applying our operators shows limited effectiveness, addressing the high-frequency noise in the signal but struggling with the low-frequency errors.

other hand, choosing a very large s can lead to a time and resource-intensive training run.

In this section, we analyze how the choice of s affects the accuracy of autodiff normals. We train an Instant NGP [11] model on the Armadillo shape. We fix a total training budget of 500 steps. For each $s \in \{0, 50, 100, 200\}$, we train another hybrid neural field using the regularization approach described above and compare the angle error of their autodiff normals. Figure 6 shows our results. We also compare against a hybrid neural field that is trained normally, i.e., using only MSE loss for 500 steps. Let us consider this as the *pre-trained* model (green dotted). We also fine-tune the pre-trained model using our fine-tuning approach described in Section 3.3 for 300 more training steps (blue dashed). As expected, higher values of s lead to more accurate autodiff normals. For $s = 200$ (i.e., 40% of the total training budget), we observe that the accuracy of autodiff normals is comparable to the fine-tuned model, which is trained for a total of 800 steps (160% of the training budget). For $s = 0$, i.e., applying Eq. (3) from the first step leads to unstable optimization, causing the angle error to explode. Interestingly, for as low as $s = 50$ (10% of training budget) training steps, we observe that the accuracy of autodiff normals improves compared to the pre-trained model.

This analysis shows that our proposed loss function (Eq. (3)), can also be used to train hybrid neural fields from



Setting	# of training steps (n)	# of warm-start steps (s)	Mean angle error (in °)
Pre-training	500	-	7.23
Regularized training	500	0	90.19
		50	4.92
		100	2.33
		200	1.83
Fine-tuning	800	500*	1.81

*For fine-tuning, s is actually the no. of pre-training steps

Figure 6. **Effect of s on angle error.** We show the effects of the number of warm-start steps (s) on the accuracy of autodiff normals. We can observe that a higher value of s leads to more accurate autodiff normals. We use the same Instant NGP architecture trained on the Armadillo shape for all settings. All models except for the fine-tuned version are trained for a total of 500 steps. The fine-tuned model (blue dashed) is trained for 300 more steps with our loss function after pre-training.

scratch such that they have more accurate spatial autodiff gradients. This requires an initial warm-start phase where we train the network to fit the zeroth-order signal followed by training with our proposed loss function (Eq. (3)). A higher number of warm-start steps leads to more accurate autodiff normals.

G. Additional Results for Rendering

In this section, we provide some additional results for rendering. Figure 7 shows our results on a large-scale scene (top) and a complex shape (bottom). We observe that our post hoc operator is relatively better at preserving sharp details, such as the boundary between the lid and the box (top), and the contours of the lips (bottom) while reducing the noisy artifacts caused by autodiff surface normals. Our fine-tuning approach also improves the accuracy of autodiff surface normals.

References

[1] Matan Atzmon and Yaron Lipman. SAL: Sign agnostic learning of shapes from raw data. In *Proc. CVPR*, 2020. 1

[2] Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. Efficient geometry-aware 3D generative adversarial networks. In *arXiv*, 2021. 1

[3] Honglin Chen, Rundi Wu, Eitan Grinspun, Changxi Zheng, and Peter Yichen Chen. Implicit neural spatial representations for time-dependent pdes. In *International Conference on Machine Learning*, 2023. 2, 3, 4

[4] Philipp Erler, Paul Guerrero, Stefan Ohrhallinger, Niloy J. Mitra, and Michael Wimmer. Points2Surf: Learning implicit surfaces from point clouds. In *Computer Vision – ECCV 2020*, pages 108–124. Springer International Publishing, 2020. 1, 2, 7, 8, 9

[5] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. In *Proc. ICML*, 2020. 1

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Proc. ICLR*, 2014. 1

[7] Zhaoshuo Li, Thomas Müller, Alex Evans, Russell H Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin. Neuralangelo: High-fidelity neural surface reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 1

[8] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, page 163–169, New York, NY, USA, 1987. Association for Computing Machinery. 3

[9] Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan H. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and Mathematics III*, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. 2

[10] Thomas Müller. tiny-cuda-nn, 2021. 1

[11] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, 2022. 1, 2, 3, 4, 5

[12] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, et al. Pytorch: An imperative style, high-performance deep learning library. In *Proc. NeurIPS*, 2019. 1

[13] Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Proc. NeurIPS*, 2020. 2, 3, 4

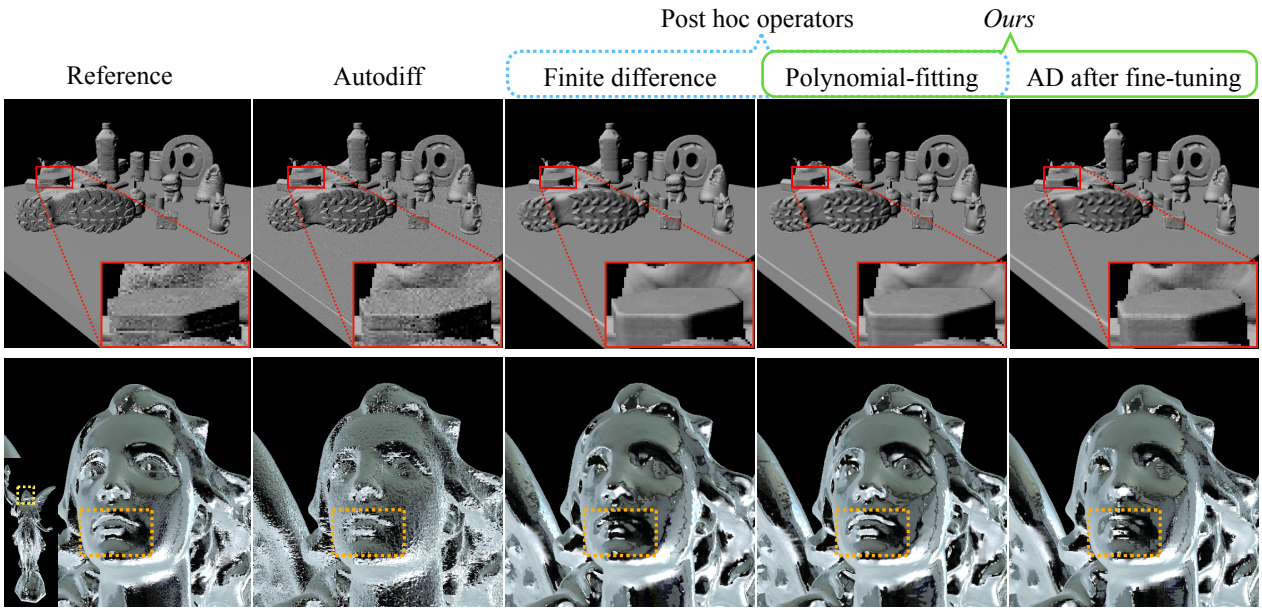


Figure 7. **Additional results for rendering.** A large-scale scene lit by a light source put in front of it (top) and specular Lucy (inset) lit by an environment map (bottom).

Shape	Surface Normals														Mean Curvature			
	L2 ↓			Ang ↓			AA@1 ↑			AA@2 ↑			σ	h	RRE ↓		σ	h
	AD	FD	Ours	AD	FD	Ours	AD	FD	Ours	AD	FD	Ours			FD	Ours		
Angel	0.12	0.04	**0.03**	7.14	2.30	**1.50**	1.93	26.47	**52.09**	7.50	61.66	**81.56**	1.5e-3	2/512	1.60	**1.47**	9.5e-1	2/256
Armadillo	0.13	0.03	**0.02**	7.27	1.83	**1.18**	1.79	24.69	**52.52**	6.88	63.93	**86.84**	2.0e-3	2/512	1.66	**0.81**	1.5e-1	2/512
Bunny	0.12	0.03	**0.02**	6.95	1.79	**1.26**	2.13	42.46	**67.98**	8.19	78.31	**88.39**	5.0e-3	2/256	1.37	**0.72**	2.5e-1	2/256
Column	0.72	0.28	**0.15**	46.15	16.27	**8.47**	0.31	0.51	**4.54**	1.20	2.08	**15.87**	3.5e-3	2/256	2.54	**0.88**	4.5e-1	2/128
Cup	0.12	0.02	**0.01**	7.06	1.24	**0.88**	2.02	62.26	**72.20**	7.93	84.37	**88.66**	8.0e-3	2/128	4.59	**0.83**	2.0e-2	2/64
Dragon	0.11	0.03	**0.02**	6.45	1.88	**1.36**	2.32	29.22	**54.68**	8.86	69.00	**86.67**	2.0e-3	2/512	1.46	**0.89**	9.0e-1	2/256
Flower	0.26	0.08	**0.06**	15.21	4.50	**3.39**	0.63	39.12	**57.18**	2.52	66.99	**69.30**	1.0e-2	2/128	13.40	**0.87**	2.0e-2	2/512
Galera	0.12	0.04	**0.03**	7.10	2.10	**1.65**	1.82	21.89	**37.75**	7.00	58.76	**75.89**	2.0e-3	2/512	1.85	**0.82**	2.5e-1	2/512
Hand	0.14	0.04	**0.02**	8.03	2.12	**1.44**	1.40	19.60	**39.32**	5.54	55.55	**79.82**	1.5e-3	2/512	1.27	**0.86**	3.5e-1	2/256
Netsuke	0.12	0.04	**0.03**	7.00	2.19	**1.67**	1.89	21.48	**41.84**	7.21	56.91	**74.48**	2.0e-3	2/512	2.40	**0.82**	3.5e-1	2/512
Serapis	0.11	**0.03**	**0.03**	6.59	1.88	**1.53**	2.24	36.02	**49.25**	8.60	68.78	**76.18**	4.0e-3	2/256	1.91	**0.92**	2.5e-2	2/128
Tortuga	0.11	0.03	**0.02**	6.07	1.51	**1.08**	2.51	46.14	**63.30**	9.70	80.90	**89.30**	3.0e-3	2/256	2.36	**0.74**	2.0e-1	2/512
Utah Teapot	0.14	0.04	**0.03**	8.33	2.53	**2.00**	1.51	27.91	**42.20**	5.91	62.64	**73.89**	4.5e-3	2/256	7.68	**0.76**	3.5e-2	2/512
XYZ Dragon	0.16	0.11	**0.07**	9.19	6.37	**4.11**	1.09	4.40	**6.83**	4.37	15.56	**23.86**	8.0e-4	2/512	2.00	**0.92**	1.5e-1	2/512
XYZ Statuette	0.61	0.25	**0.18**	37.50	14.53	**10.55**	0.12	0.72	**2.11**	0.46	2.94	**7.86**	1.5e-3	2/512	9.00	**0.97**	2.0e-3	2/512
Mean	0.21	0.07	**0.05**	12.40	4.20	**2.80**	1.58	26.86	**42.92**	6.12	55.22	**67.90**	-	-	3.67	**0.89**	-	-

Table 2. **Post hoc operator evluation for Instant NGP**. We compare our operators on the FamousShape dataset [4]. σ, h indicate the selected hyperparameters for our approach and finite difference (FD) respectively. Note that our approach provides more accurate derivatives than the baselines.

Shape	Before fine-tuning						After fine-tuning						σ
	L2 ↓	Ang ↓	AA@1 ↑	AA@2 ↑	CD ↓	F-Score ↑	L2 ↓	Ang ↓	AA@1 ↑	AA@2 ↑	CD ↓	F-Score ↑	
Angel	0.12	7.13	1.92	7.54	5.32e-4	93.47	**0.04**	**2.06**	**31.74**	**67.25**	5.38e-4	93.45	1.5e-3
Armadillo	0.13	7.23	1.80	6.95	1.63e-4	96.15	**0.03**	**1.72**	**31.04**	**69.70**	1.65e-4	96.14	2.0e-3
Bunny	0.12	6.98	2.08	8.11	7.26e-4	93.26	**0.02**	**1.37**	**60.74**	**86.52**	7.09e-4	93.35	5.5e-3
Column	0.73	46.25	0.31	1.24	2.93e-3	85.89	**0.14**	**8.35**	**4.66**	**16.16**	2.95e-3	85.71	3.5e-3
Cup	0.12	7.05	2.06	7.91	3.24e-4	94.47	**0.02**	**1.15**	**60.76**	**84.76**	3.27e-4	89.11	1.0e-2
Dragon	0.11	6.46	2.31	8.81	1.99e-3	89.97	**0.03**	**1.90**	**33.25**	**70.87**	1.98e-3	89.96	2.0e-3
Flower	0.26	15.20	0.67	2.52	3.40e-4	96.48	**0.06**	**3.32**	**55.34**	**70.17**	3.47e-4	91.49	1.0e-2
Galera	0.12	7.10	1.82	6.93	8.37e-4	92.29	**0.03**	**1.97**	**28.11**	**65.63**	8.35e-4	92.31	2.0e-3
Hand	0.14	8.02	1.39	5.53	2.64e-3	88.08	**0.04**	**2.10**	**21.30**	**57.41**	2.67e-3	88.10	1.5e-3
Netsuke	0.12	7.01	1.90	7.29	1.86e-4	96.13	**0.04**	**2.11**	**26.40**	**61.64**	1.87e-4	96.11	2.0e-3
Serapis	0.11	6.57	2.21	8.55	1.18e-3	91.79	**0.03**	**1.65**	**44.96**	**73.21**	1.18e-3	91.73	4.0e-3
Tortuga	0.11	6.04	2.53	9.75	3.29e-4	96.04	**0.02**	**1.18**	**61.18**	**87.00**	3.28e-4	96.07	3.5e-3
Utah Teapot	0.14	8.29	1.50	5.92	6.23e-4	94.30	**0.04**	**2.06**	**38.10**	**70.07**	6.31e-4	94.13	4.0e-3
XYZ Dragon	0.16	9.18	1.13	4.40	9.72e-4	90.40	**0.10**	**5.81**	**4.62**	**16.47**	9.72e-4	90.37	1.5e-3
XYZ Statuette	0.61	37.46	0.13	0.46	9.69e-5	97.29	**0.19**	**11.11**	**1.77**	**6.76**	9.97e-5	96.17	1.5e-3
Mean	0.21	12.38	1.58	6.12	9.24e-4	93.07	**0.05**	**3.20**	**33.59**	**60.24**	9.28e-4	92.28	-

Table 3. **Fine-tuning using polynomial-fitting for Instant NGP**. Full results for fine-tuning using polynomial-fitting over the FamousShape dataset [4]. σ denotes the hyperparameter value with the best results from the ensemble.

Shape	Surface Normals														Mean Curvature			
	L2 ↓			Ang ↓			AA@1 ↑			AA@2 ↑			σ	h	RRE ↓		σ	h
	AD	FD	Ours	AD	FD	Ours	AD	FD	Ours	AD	FD	Ours			FD	Ours		
Angel	0.09	0.05	**0.04**	5.20	2.71	**2.39**	13.56	33.30	**43.95**	33.37	58.60	**66.99**	2.0e-3	2/512	3.41	**0.87**	4.5e-1	2/256
Armadillo	0.08	0.04	**0.03**	4.87	2.09	**1.75**	7.84	24.00	**32.72**	23.34	58.00	**68.52**	2.0e-3	2/512	1.75	**1.49**	9.0e-1	2/256
Bunny	0.07	0.03	**0.02**	3.78	1.73	**1.26**	13.10	47.91	**67.93**	37.00	79.02	**88.29**	5.0e-3	2/256	1.25	**0.81**	1.5e-1	2/256
Column	0.27	0.21	**0.14**	16.07	11.98	**8.33**	1.96	4.09	**11.48**	6.96	12.64	**24.65**	3.5e-3	2/512	4.62	**0.83**	2.0e-2	2/64
Cup	0.06	0.02	**0.01**	3.45	1.20	**0.86**	21.27	64.09	**72.44**	45.64	84.11	**88.60**	8.0e-3	2/128	1.24	**0.72**	2.5e-1	2/256
Dragon	0.08	0.04	**0.03**	4.56	2.25	**1.76**	10.73	26.83	**44.20**	30.92	60.64	**76.86**	2.5e-3	2/512	1.52	**0.89**	9.0e-1	2/256
Flower	0.14	0.07	**0.06**	8.13	4.26	**3.36**	14.50	**57.72**	57.60	37.26	**70.87**	69.32	1.0e-2	2/128	3.28	**0.87**	2.0e-2	2/32
Galera	0.08	0.04	**0.04**	4.62	2.40	**2.07**	10.22	23.97	**32.01**	29.11	55.73	**65.33**	2.0e-3	2/512	1.70	**0.82**	2.5e-1	2/512
Hand	0.09	0.04	**0.04**	4.93	2.55	**2.03**	8.24	19.34	**26.81**	25.71	50.25	**62.64**	2.0e-3	2/512	1.90	**0.86**	3.5e-1	2/64
Netsuke	0.08	0.04	**0.03**	4.56	2.26	**1.99**	10.12	26.22	**33.08**	28.77	57.91	**65.06**	2.0e-3	2/512	1.45	**0.82**	3.5e-1	2/256
Serapis	0.07	0.03	**0.03**	4.01	1.89	**1.54**	17.73	39.21	**48.89**	36.91	67.65	**75.46**	4.0e-3	2/256	1.98	**0.93**	2.5e-2	2/128
Tortuga	0.05	0.03	**0.02**	2.94	1.47	**1.13**	17.44	50.36	**62.65**	45.51	80.91	**87.94**	3.0e-3	2/256	1.60	**0.74**	2.0e-1	2/512
Utah Teapot	0.06	0.04	**0.03**	3.51	2.28	**1.98**	22.17	36.37	**42.62**	47.55	67.44	**73.79**	4.5e-3	2/256	0.96	**0.76**	3.5e-2	2/32
XYZ Dragon	0.16	0.13	**0.12**	9.39	7.37	**6.89**	2.16	3.58	**4.00**	8.15	12.72	**14.01**	1.5e-3	2/512	2.13	**0.92**	1.5e-1	2/256
XYZ Statuette	0.31	0.23	**0.21**	18.26	13.13	**12.27**	1.36	2.91	**3.88**	4.82	9.41	**12.28**	1.5e-3	2/512	10.54	**0.97**	2.5e-3	2/512
Mean	0.11	0.07	**0.06**	6.55	3.97	**3.31**	11.49	30.66	**38.95**	29.40	55.06	**62.65**	-	-	2.62	**0.89**	-	-

Table 4. **Post hoc operator evluation on Dense Grid**. Comparison on the FamousShape dataset [4]. σ, h indicate the selected hyperparameters for our approach and finite difference (FD) respectively. Note that our approach provides more accurate surface normals and mean curvature than the baselines.

Shape	Before fine-tuning						After fine-tuning						σ
	L2 ↓	Ang ↓	AA@1 ↑	AA@2 ↑	CD ↓	F-Score ↑	L2 ↓	Ang ↓	AA@1 ↑	AA@2 ↑	CD ↓	F-Score ↑	
Angel	0.09	5.20	13.64	33.45	5.33e-4	92.87	**0.06**	**3.62**	**30.12**	**53.67**	5.35e-4	92.50	2.0e-3
Armadillo	0.08	4.87	7.72	23.40	1.65e-4	95.28	**0.06**	**3.27**	**14.84**	**39.06**	1.69e-4	95.02	2.0e-3
Bunny	0.07	3.78	13.11	37.07	7.25e-4	91.00	**0.03**	**1.59**	**52.63**	**81.11**	7.15e-4	90.67	5.0e-3
Column	0.27	16.15	1.87	6.90	2.95e-3	84.82	**0.17**	**9.79**	**4.43**	**13.75**	2.92e-3	73.45	3.5e-3
Cup	0.06	3.48	21.14	45.34	3.24e-4	84.89	**0.02**	**1.11**	**63.64**	**84.39**	3.20e-4	78.32	8.0e-3
Dragon	0.08	4.54	10.74	30.85	1.99e-3	86.31	**0.05**	**2.72**	**26.48**	**57.89**	1.99e-3	85.95	2.5e-3
Flower	0.14	8.14	14.34	37.25	3.40e-4	91.50	**0.06**	**3.40**	**54.03**	**68.48**	3.46e-4	83.98	1.0e-2
Galera	0.08	4.62	10.08	28.87	8.41e-4	85.93	**0.05**	**2.93**	**23.18**	**51.34**	8.36e-4	85.87	2.0e-3
Hand	0.09	4.95	8.16	25.72	2.64e-3	87.68	**0.06**	**3.28**	**13.77**	**39.75**	2.65e-3	87.59	2.0e-3
Netsuke	0.08	4.55	10.21	29.01	1.86e-4	92.93	**0.05**	**2.93**	**20.56**	**47.99**	1.84e-4	92.82	2.0e-3
Serapis	0.07	4.01	17.48	36.62	1.18e-3	85.15	**0.03**	**1.97**	**41.59**	**66.92**	1.17e-3	84.85	4.0e-3
Tortuga	0.05	2.94	17.30	45.35	3.29e-4	93.16	**0.03**	**1.47**	**51.17**	**80.15**	3.30e-4	93.04	3.0e-3
Utah Teapot	0.06	3.52	22.07	47.55	6.22e-4	90.70	**0.04**	**2.19**	**39.15**	**71.07**	6.24e-4	89.93	4.5e-3
XYZ Dragon	0.16	9.38	2.17	8.11	9.72e-4	89.68	**0.15**	**8.66**	**2.82**	**10.23**	9.77e-4	89.28	1.5e-3
XYZ Statuette	0.31	18.23	1.34	4.89	9.70e-5	95.58	**0.26**	**15.33**	**2.20**	**7.55**	1.03e-4	91.53	1.5e-3
Mean	0.11	6.56	11.42	29.35	9.26e-4	89.83	**0.08**	**4.40**	**29.32**	**51.40**	9.25e-4	87.66	-

Table 5. **Fine-tuning using polynomial-fitting on Dense Grid**. Full results for fine-tuning using polynomial-fitting over the FamousShape dataset [4]. σ denotes the hyperparameter value that obtained the best results.

Shape	Surface Normals														Mean Curvature			
	L2 ↓			Ang ↓			AA@1 ↑			AA@2 ↑			σ	h	RRE ↓		σ	h
	AD	FD	Ours	AD	FD	Ours	AD	FD	Ours	AD	FD	Ours			FD	Ours		
Angel	0.08	0.04	**0.03**	4.86	2.30	**1.92**	5.91	27.50	**30.46**	20.85	62.71	**69.31**	1.5e-3	2/512	2.99	**1.63**	9.0e-1	2/512
Armadillo	0.09	0.03	**0.03**	5.35	2.00	**1.48**	4.04	21.41	**35.38**	15.00	58.19	**77.42**	2.0e-3	2/512	1.23	**0.81**	2.0e-1	2/256
Bunny	0.10	0.03	**0.02**	5.74	1.98	**1.31**	3.95	33.16	**65.12**	14.66	71.98	**87.97**	5.0e-3	2/256	1.43	**0.72**	2.5e-1	2/256
Column	0.38	0.24	**0.14**	22.87	14.18	**8.31**	0.70	2.06	**8.21**	2.68	7.60	**23.31**	3.0e-3	2/512	6.07	**0.95**	3.0e-2	2/512
Cup	0.09	0.02	**0.02**	5.20	1.31	**0.90**	4.86	57.47	**71.78**	17.38	83.76	**88.42**	8.0e-3	2/128	6.32	**0.82**	9.0e-4	2/32
Dragon	0.09	0.04	**0.03**	5.24	2.32	**1.77**	4.56	19.24	**36.44**	16.54	53.50	**75.71**	2.5e-3	2/512	1.58	**0.93**	9.0e-1	2/256
Flower	0.18	0.08	**0.06**	10.65	4.47	**3.40**	3.08	44.50	**56.91**	11.46	69.30	**69.07**	1.0e-2	2/128	2.68	**0.87**	2.0e-2	2/64
Galera	0.10	0.04	**0.03**	6.01	2.51	**1.97**	3.25	15.84	**24.95**	12.33	46.93	**63.97**	2.0e-3	2/512	1.37	**0.82**	2.5e-1	2/256
Hand	0.08	0.04	**0.03**	4.46	2.03	**1.59**	6.42	22.34	**33.03**	22.43	59.48	**74.47**	1.5e-3	2/512	1.88	**0.85**	3.5e-1	2/64
Netsuke	0.10	0.04	**0.04**	5.92	2.52	**2.05**	3.47	16.26	**25.08**	12.94	47.23	**62.22**	2.0e-3	2/512	1.55	**0.82**	3.5e-1	2/256
Serapis	0.11	0.04	**0.03**	6.24	2.17	**1.65**	3.47	25.35	**45.18**	12.82	60.02	**74.30**	4.0e-3	2/256	6.32	**0.93**	2.5e-2	2/512
Tortuga	0.09	0.03	**0.02**	5.32	1.76	**1.23**	4.36	34.08	**57.78**	15.83	72.24	**87.04**	3.5e-3	2/256	2.86	**0.74**	2.0e-1	2/512
Utah Teapot	0.11	0.05	**0.04**	6.16	2.62	**2.04**	5.02	28.41	**40.87**	17.91	61.90	**73.30**	4.5e-3	2/256	7.03	**0.75**	3.5e-2	2/512
XYZ Dragon	0.22	0.13	**0.12**	12.80	7.52	**6.96**	0.72	2.40	**2.43**	2.87	9.12	**9.22**	1.5e-3	2/512	2.78	**0.92**	1.5e-1	2/512
XYZ Statuette	0.37	0.23	**0.21**	22.04	13.16	**11.91**	0.31	1.29	**1.35**	1.27	5.13	**5.29**	1.5e-3	2/512	15.64	**0.97**	2.5e-3	2/512
Mean	0.15	0.07	**0.06**	8.59	4.19	**3.23**	3.61	23.42	**35.67**	13.13	51.27	**62.74**	-	-	4.12	**0.90**	-	-

Table 6. **Post hoc operator evluation on Tri-planes**. Comparison on the FamousShape dataset [4]. σ, h indicate the selected hyperparameters for our approach and finite difference (FD) respectively.