Learning to Rasterize Differentiably

DOI: 10.1111/cgf.15145
Eurographics Symposium on Rendering 2024
E Garces and E. Haines
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 43 (2024), Number 4

Learning to Rasterize Differentiably

C. Wu[1], H. Mailee[2], Z. Montazeri[2] and T. Ritschel[1]

[1]University College London, United Kingdom
[2]University of Manchester, United Kingdom

Abstract

Differentiable rasterization changes the standard formulation of primitive rasterization —by enabling gradient flow from a pixel to its underlying triangles— using distribution functions in different stages of rendering, creating a "soft" version of the original rasterizer. However, choosing the optimal softening function that ensures the best performance and convergence to a desired goal requires trial and error. Previous work has analyzed and compared several combinations of softening. In this work, we take it a step further and, instead of making a combinatorial choice of softening operations, parameterize the continuous space of common softening operations. We study meta-learning tunable softness functions over a set of inverse rendering tasks (2D and 3D shape, pose and occlusion) so it generalizes to new and unseen differentiable rendering tasks with optimal softness.

CCS Concepts

• ***Computing methodologies*** → ***Rendering; Rasterization;*** *Artificial intelligence;*

1. Introduction

While forward rendering generates a 2D image based on 3D scene parameters, inverse rendering optimizes these parameters to reproduce a given 2D reference image. The use of modern gradient-based optimizers in this context requires the rendering process to be differentiable, facilitating applications such as model reconstruction [KUH18], pose estimation [LB14, GMCG20], and the estimation of lighting and materials [CLG*21, NDDJK21]. However, differentiation is challenging due to the discontinuities commonly present in rendering.

The renderer can generally be differentiated in two main ways: either by approximating the gradients using the exact forward rendering process, which may require manually designed gradients or by directly approximating the forward rendering process to enable Automatic Differentiation (AD). For gradient approximation, Loper et al. [LB14] leverage the differences between neighboring pixels, while Kato et al. [KUH18] employ a hand-crafted function. Li et al. [LADL18] proposed the integration of gradients using Monte Carlo ray tracing. In contrast, other studies achieve natural differentiability through probabilistic perturbation, such as [RRR*15], which uses Gaussian distributions to blur rasterization and approximates the rasterized value as a density parameter controlling the transparency of blobby objects. Subsequent studies have explored using functions like the square root of a logistic distribution [LLCL19] and logistic distribution [PBKD21] in similar ways. Another approach involves smoothing the cost function via Monte Carlo convolution across optimization parameters [FR23].

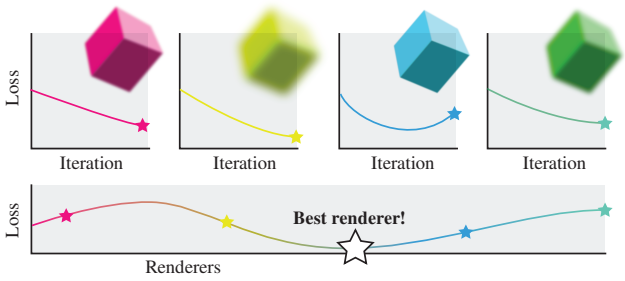


Figure 1: *Among the continuously differentiable rasterization renderers, we identify the one most suited to solving a family of inverse rendering tasks.*

These softer rasterization variants have enabled effective inverse rendering by overcoming the challenges posed by discontinuities. However, identifying the appropriate softness function remains a significant challenge. We argue that no single function is universally optimal; the choice of softness depends on the specific problem, and adapting this choice should be automated and systematic.

In this work, we propose a principled approach that addresses this issue at a higher level of abstraction as it is visualized in Figure 1. Instead of focusing on individual inverse rendering problems, we consider the entire spectrum of potential problems. Utilizing a training set of inverse rendering challenges, we identify the most ef-

fective modifications to the renderer in terms of convergence speed and/or quality, making it task-specific and differentiable.

Concretely, we introduce a meta-learning strategy to learn the optimal edge and occlusion softness for a differentiable rasterizer in the form of an Multi-layer Perceptron (MLP). We demonstrate that our method surpasses state-of-the-art techniques that all rely on manually selected softness scales for given parametric distributions.

To facilitate further research, we publish the code at https://github.com/Theo-Wu/MetaRas.

2. Background

Differentiable rendering allows computing gradients of 3D scene parameters with respect to the image pixels.

2.1. Problem setting

Let $\mathcal{R}_\mathrm{h}$ be a common hard renderer that takes scene parameters θ and maps them to an image. Differentiating this function is not possible due to the discontinuity of the parameters and gradients that are typically zero. Hence, the subscript h is used to denote "hard". Formally, let $\theta^\star$ be the optimal scene parameters for an image I and let us assume they are unique. Then, unfortunately,

$$\mathtt{opt}(||\mathcal{R}_\mathrm{h}(\theta) - I||_2, \theta) \neq \theta^\star,$$

where $\mathtt{opt}()$ is an optimizer, such as gradient descent that minimizes the first argument (here, the image difference of rendering and reference image) by changing the second argument (here, the scene parameters).

A differentiable soft renderer $\mathcal{R}_\mathrm{s}$, however, would potentially converge to the global optimum, as in:

$$\mathtt{opt}(||\mathcal{R}_\mathrm{s}(\theta) - I||_2, \theta) = \theta^\star.$$

Although $\mathcal{R}_\mathrm{s}$ and $\mathcal{R}_\mathrm{h}$ are not identical, the crucial insight is that as long as the gradient-based optimizer converges to the same minimum using its gradients, the specific function used may not significantly impact the outcome. This introduces the possibility of replacing $\mathcal{R}_\mathrm{h}$ with a differentiable function, $\mathcal{R}_\mathrm{s}$, that results in similar optimal parameters.

In general, $\mathcal{R}_\mathrm{s}$ and $\mathcal{R}_\mathrm{h}$ are not identical; thus, we change the target function we aim to optimize. Nonetheless, the differentiable (soft) version maintains mostly non-zero gradients, fostering optimism that these gradients will converge toward the true, desired optimum. A fundamental insight of this work is the notion that the specific form of $\mathcal{R}_\mathrm{s}$ and $\mathcal{R}_\mathrm{h}$ does not crucially impact the optimization outcome, as long as a gradient-based optimizer can effectively use their gradients to converge to the same minimum.

This leads us to an important question: if we have the flexibility to modify the renderer, how should we systematically determine the best settings? Our goal is to substitute the original function with a differentiable variant that yields similar optimal results when subjected to gradient-based optimization.

The two dominant rendering techniques are path tracing and rasterization. Differentiable Monte Carlo path-tracing is adept at managing all types of illumination given sufficient computational resources, as discussed in prior studies [LADL18, ZMY*20], without needing explicit boundary sampling [BLD20] or sampling silhouette edges [LHJ19] by approximating the pre-filtered gradient [YBAF22] to handle discontinuities. Our focus in this paper, however, is on differentiable rasterization. This specific renderer type is restricted to direct illumination but offers the advantage of efficient computation.

2.2. Differentiable rasterization

In this section, we first discuss rasterization and then explain its differentiable variant.

Hard rasterization: Rasterization generally refers to the process of determining which pixels fall within a geometric primitive that is projected onto the image plane. This initial step does not involve assigning attributes to each pixel; such tasks are usually handled at the fragment level by dedicated hardware or software. Specifically, rasterization involves processing a set of 2D triangles, each with defined depth and attributes such as color at its vertices, to determine the attributes of each pixel in a 2D image. A common method employs an *edge function* [Pin88] to test whether a pixel lies within the boundaries defined by the edges of the triangle. If a pixel is determined to be within these boundaries, its attributes are unconditionally assigned based on its position relative to the triangle. This assignment uses a Heaviside step function at the triangle edges, rendering it non-differentiable; the gradient is zero across the field except at the boundaries, where it is undefined due to the abrupt change.

Furthermore, in scenarios where multiple primitives overlap on the same pixel, the attribute selected for that pixel corresponds to the one associated with the closest primitive. This selection process, known as z-buffering, ensures that only the attribute of the closest primitive is retained, while the attributes of the other overlapping primitives are discarded. However, similar to other aspects of rasterization, the z-buffering process is also non-differentiable.

This indicates that differentiating rasterization involves addressing both occlusion and edge tests. This process essentially reduces to differentiating a function based on: $h(d) \in \mathbb{R} \to [0,1]$ where d represents a *distance* measurement, which could pertain to either 2D spatial dimensions or depth within the scene.

Soft rasterization: The fundamental concept behind soft rasterization, as introduced by Liu et al. [LLCL19], involves transforming the traditional hard step function h into a soft function $s(d) \in \mathbb{R} \to [0,1]$ with global support, ensuring non-zero gradients throughout. The output of this soft function is utilized in alpha-compositing, also considering depth attributes. Examples of such functions include Gaussian distributions [RRR*15], the square-root of logistic distributions [LLCL19], and exponential functions [CLG*19].

Petersen et al. [PBDC19] proposed a method to soften the z-buffer using a weighted softmax function defined over depth values. Additionally, dedicated aggregation functions have been proposed

for silhouette computation; these functions differentiate scenes using binary color and are independent of depth. Subsequently, [PGBD22] further refined these concepts by defining them as T-conorms and exploring various implementations. Their research demonstrated that several functions could be effective as long as they are monotonous, and it analyzed the performance of each.

This line of work serves as another important inspiration for our approach, in which we transition from fixed function choices to a task-specific, continuously optimized selection of soft functions. Our approach does not deeply analyze the mathematical properties of these functions; rather, it focuses on fitting the data to practical scenarios, where the primary benefit is reduced computational cost in optimization processes, even without a comprehensive understanding of the underlying reasons.

Furthermore, [LLCL19] introduced an aggregation function, $A(d,z)$, which softens both spatial distance and depth, allowing gradients to influence both visible and occluded primitives and their z coordinates effectively.

Similarly, Laine et al. [LHK*20] defined the soft function with local support on surface coverage instead of distance to the edge. They approximate the coverage by the position of edges' crossing points between adjacent pixel pairs. This can be seen as a variant of the truncated linear function on a transformed space.

To be systematic, we control variables in our comparison and only compare with our backbone GenDR, where everything is consistent except for the MLP to eliminate the influence of other implementations. Nevertheless, the idea of meta-learned softness is independent of differentiable rasterizer implementation. For example, meta-learning an MLP to replace the linear blending operation in Nvdiffrast [LHK*20] might also improve it.

For a study of differentiable rasterization in general we refer the readers to the survey by [KBM*20].

3. Meta-learning a differentiable rasterizer

3.1. Meta problem setting

To make systematic progress we move the problem to another level of abstraction. We phrase the challenge as finding a renderer $\mathcal{R}_s$, parameterized in some way by $\theta_i^\star$, that converges best over a set of tasks I_i, where i refers to the i_{th} category of task:

$$\arg\min_{\mathcal{R}_s} \mathbb{E}_i[||\texttt{opt}(||\mathcal{R}_s(\theta)-I_i||_2,\theta)-\theta_i^\star||]\,.$$

where opt refers to an optimizer.

In practice, sometimes the ground truth parameters $\theta_i^\star$ are not available; instead, we often have only the images I_i. Under these circumstances, the challenge can be reformulated as follows:

$$\arg\min_{\mathcal{R}_s} \mathbb{E}_i[||\mathcal{R}_s(\texttt{opt}(||\mathcal{R}_s(\theta)-I_i||_2,\theta))-I_i||]\,.$$

We do not confine ourselves to a predefined, discrete set of functions for manual selection. Instead, we explore the continuous space of all possible soft renderers using meta-learning, which is explained as follows.

3.2. Meta-learning

Meta-learning, often described as "learning to learn", is an algorithmic approach that enhances a model by observing how different models perform across various tasks. The model-agnostic meta-learning (MAML) algorithm was introduced by Finn et al. [FAL17], which has shown success in learning new tasks with limited training samples by utilizing a double-loop training process.

Given the unique aspects of our tasks, we employ a method similar to MAML, as outlined in Algo. 1. Our approach operates with nested loops (L2 and L5): the outer loop adjusts meta-parameters—the shape of the softening functions—while the inner loop, using the Adam optimizer (L8), optimizes the scene settings parameters. Importantly, the inner loop is designed to unfold into a formula that the outer loop can differentiate automatically. The outer loop's gradient update (L11) modifies the parameters governing the inner optimization, in our case, the differentiable renderer (L12). This ensures that the optimization converges more closely to its target with each iteration.

Our implementation diverges from traditional MAML, as shown in Figure 2, since we do not apply the learned parameters from our meta-loop to new test instances like classic MAML. Furthermore, rather than accessing ground truth parameters, our approach focuses solely on minimizing image error—a strategy intended to reduce parameter error indirectly. This method avoids the need for ground truth supervision, relying purely on images, which simplifies the learning process. Additionally, we do not meta-train initializations or step sizes, which could potentially offer further advantages depending on the specific inverse problem being addressed.

The meta-optimization over all soft rasterizers, parametrized by ϕ, is outlined as follows:

$$\texttt{opt}(\mathbb{E}_i[||\mathcal{R}_s(\texttt{opt}(||\mathcal{R}_s(\theta,\phi)-I_i||_2,\theta),\phi)-I_i||_2],\phi)\,.$$

These parametrized rasterizers differ from traditional renderers primarily in how they handle triangle edge testing and occlusions. Instead of using hard step functions, they employ a function, $s_\phi(d)$, which is dependent on a specific parameter vector, ϕ. Next, we will explore various soft edge functions in further detail.

Algorithm 1 Meta-learning for Soft Rasterization

Require: Θ: Set of task images
Ensure: Meta-learned soft renderer parameter ϕ
1: ϕ = RANDOM
2: **for** $i \in [1,n]$ **do**
3: $\quad\theta$ = RANDOM
4: $\quad I_i \leftarrow$ SAMPLEIMAGE(Θ)
5: $\quad$**for** $j \in [1,m]$ **do**
6: $\quad\quad\mathcal{L}_\theta \leftarrow ||\mathcal{R}_s(\theta,\phi)-I_i||_2$
7: $\quad\quad\nabla_\theta$ = GRADIENT($\mathcal{L}_\theta,\theta$)
8: $\quad\quad\theta = \theta - \lambda_\theta\nabla_\theta$
9: $\quad$**end for**
10: $\quad\mathcal{L}_\phi \leftarrow ||\mathcal{R}_s(\theta,\phi)-I_i||_2$
11: $\quad\nabla_\phi$ = GRADIENT($\mathcal{L}_\phi,\phi$)
12: $\quad\phi = \phi - \lambda_\phi\nabla_\phi$
13: **end for**

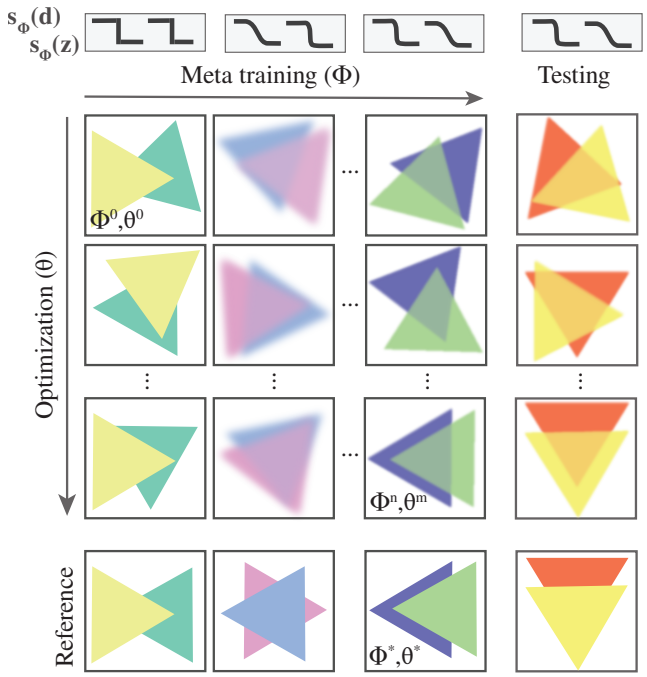


Figure 2: ***Meta-learning.*** *Meta-optimization consists of two training loops to jointly optimize the scene parameters* θ *for one task instance (vertical) and the renderer parameters* ϕ *across many task instances (horizontal). All columns represent the same task category of changing two triangles' positions to match the reference image in the last row. At test time, the optimal renderer is good at solving unseen tasks, as shown in the rightmost column. This is, because, towards the end of meta-training, the optimization itself mimics the reference closely. The top shows the soft functions used to render one column: a soft depth step makes the triangles transparent, and a soft edge function makes the edges blurry.*

3.3. Tunable softness

We explore two types of tunable softness functions: Cumulative Density Function (CDF) and MLPs. Contrary to the approach taken by GenDR [PGBD22], we do not strictly enforce the softening function to be an explicit CDF of another function, as we find this requirement to be optional and not necessary for the function's practical application. Concurrently, with the flexibility of MLP, more complex edge functions can be utilized that seem like variations of S-curves, but are actually optimized on the required task (Figure 3).

CDF: There are many options in this class of functions, such as a logistic function with a softening parameter ϕ

$$s_{\phi_d} = \mathrm{S}(d) = \frac{1}{1+\exp(-d\cdot\phi)}\,. \tag{1}$$

This class of functions has been previously utilized, allowing for parameter optimization through grid-search due to its low dimensionality. We further refine the optimization of ϕ using meta-learning, aiming for more precise control. Additionally, we find that a more general class of functions yields superior results, which we discuss next.

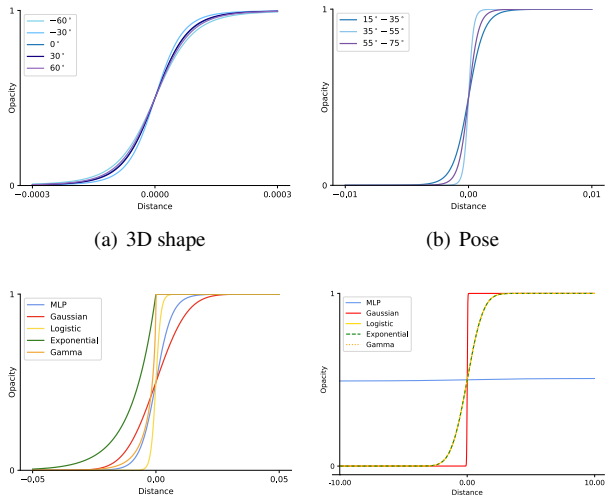
MLP: For more general softening, we employ an MLP that comprises five layers with tanh as internal activations and a residual layer that skips three middle layers, followed by a final sigmoid:

$$s_{\phi_d}(d) = \mathrm{S}(\mathsf{W}_5 T(T(\mathsf{W}_1 d) + \mathsf{W}_4 T(\mathsf{W}_3 T(\mathsf{W}_2 T(\mathsf{W}_1 d))))), \tag{2}$$

where T represents the tanh function. Note the repeated use of W_1, which functions as a residual connection. The network weights, $\phi = \{\mathsf{W}_1,\ldots,\mathsf{W}_5\}$, have a width of 4 and are randomly initialized from a uniform distribution. Our investigations show that the inclusion of biases or affine coordinates, such as $\mathsf{W}_1(d+1)$, does not contribute effectively, so we opt to exclude them from this configuration.

3.4. Tunable depth

To take the renderer's flexibility a step further, we consider the use of CDFs and MLP in the aggregation function, to evaluate our approach's performance in tasks like occlusion. Originally, the sigmoid function is directly used on inverted normalized depth to calculate the contribution of each primitive's color in the final pixel. With the aim of making the depth itself differentiable, we first apply a soft function to the depth and then use it as the input of the sigmoid function. We use the CDFs as is, but some modifications are made to the structure of MLP. Since the sigmoid will be applied in the aggregation function ([LLCL19]), the last layer of the MLP can



(a) 3D shape (b) Pose

(c) 2D shape (d) Occlusion

Figure 3: ***Edge functions for different tasks.*** *We visualize the meta-learned parameters in two sets of tasks. In 3D shape (a) and Pose (b), distinct MLPs are trained for each angle, shown in different shades of blue. For 2D shape (c) and Occlusion (d), the fixed viewpoint necessitates a single MLP, which can be compared to other CDFs with grid-searched softness. The same parameters can be used for more complex scenes (i.e. Transfer tasks) without re-meta-training.*

be removed. To ensure non-negative depth throughout differentiation as the camera is set to origin by default, the activation function is changed from tanh to *ReLU*. Keeping the same structure as our softness function, we have:

$$s_{\phi_z}(z) = \mathsf{W}_5 R(R(\mathsf{W}_1 z) + \mathsf{W}_4 R(\mathsf{W}_3 R(\mathsf{W}_2 R(\mathsf{W}_1 z)))), \quad (3)$$

where R represents the ReLU function.

3.5. Combination

Overall, our proposal involves meta-learning the corresponding parameters ϕ, namely ϕ_d for spatial distance and ϕ_z for depth, to enable soft blending in distance d and depth z. We blend the colors of all primitives for a pixel with weights, that are made to sum to 1 by a normalization. To this end, we define the final value at pixel i as follows:

$$C_i = \left(\sum_j s_{\phi_d}(d_j^i) \cdot s_{\phi_z}(z_j^i) \right)^{-1} \cdot \sum_j s_{\phi_d}(d_j^i) \cdot s_{\phi_z}(z_j^i) \cdot C_j^i \quad (4)$$

where C_j^i is the color of pixel i at the j-th primitive.

4. Evaluation

We compare different methods (Sec. 4.1) according to different metrics (Sec. 4.2) on different tasks (Sec. 4.3) leading to the results presented in Sec. 4.5. To keep consistency with our backbone, GenDR, we also use our MLP on the famous single-view 3D reconstruction experiment trained on the ShapeNet dataset (Sec. 4.6).

4.1. Methods

We evaluate five different methods, as summarized in Tab. 1. One method utilizes our meta-learned MLP, while the others employ simpler edge functions with low-dimensional parameters suitable for grid-search optimization. For these existing edge functions, we use parameters that have been previously established in the literature through grid searches. Except for the variations in soft edge handling, all methods employ identical rendering setups, including perspective projection and Phong materials, and operate at the same resolution without super-sampling. We use GenDR as the foundational backbone, into which we integrate various edge functions for comparison.

4.2. Metrics

While image distance serves as our primary loss, we analyze results using two distinct types of metrics as shown in the Metric of Tab. 2. The first is image distance: for the 2D SHAPE and OCCLUSION tasks, this is measured using Mean Squared Error (MSE), while for 3D SHAPE and POSE, we use Intersection over Union (IoU) [EVGW*10].

The second type of metric is parameter distance, which provides insights into the accuracy with which the models can determine underlying geometric or positional parameters. Specifically, for the 3D SHAPE task, we use Chamfer distance; for POSE, we measure angle differences; and for OCCLUSION, we assess depth order. It's important to note that for 2D SHAPE, identifying ground truth parameters for triangle vertices is challenging. Since our optimization efforts aim to uncover these parameters, this additional metric offers valuable insight into each method's performance.

Furthermore, it's worth noting that we did not utilize these parameters during the learning phase, and they're only used for comparison in evaluation. Our focus does not extend to optimizing light or materials, as they are out of the scope of rasterizers. When presenting results, whether through charts or mean value, we ensure reliability by averaging data across 30 to 300 runs of the inner optimizer, depending on the task, in line with practices from GenDR.

Table 1: *Methods in our comparison experiments.*

Methods	Backbone	Function	Params	Tuning
`MLP`	GenDR	$\texttt{MLP}(d;\phi)$	56	Meta-learned
`Gauss`	GenDR	$\mathcal{N}(d \cdot \phi)$	1	Grid-searched
`Log`	GenDR	$1/(1+\exp(-d \cdot \phi))$	1	Grid-searched
`Exp`	GenDR	$\exp(d;\phi)$	2	Grid-searched
`Gamma`	GenDR	$\mathcal{G}(d;\phi;p=0.5)$	3	Grid-searched

4.3. Tasks

We explore four inverse rendering tasks for optimizing 2D and 3D shapes, camera pose, and occlusion as illustrated in Fig. 3. For each task, we study two variants: the *original* version and the *transfer* one. In the *original* setting, the rasterizer is meta-learned on the task with a designated input, say we find the best edge softness to make one particular logo. In the *transfer* instance, the trained rasterizer is used on the same task but with different input and initial parameters, e. g., a new logo with 4000 initial triangles instead of 800. The initial parameters are pointed out in Tab. 2, under the "Original" and "Trans." columns. For some problems, we dynamically vary the size of spatial and depth distances. For other problems, we use preset values, following GenDR's decay protocol where applicable (column "Dec." in Tab. 2). The "Occ." column shows the use of soft functions for color aggregation and depth tuning, which is only utilized for OCCLUSION task.

2D Shape: In this task, we determine the positions of hundreds of 2D triangle vertices to recreate a specific target image. The initial positions of these triangles are randomized. For the *original* variant, an EGSR logo serves as the target image during both the meta-learning and optimization phases. The challenge is to accurately reconstruct this logo. In the *transfer* variant, we apply both

Table 2: *Properties of all tasks we study.*

Dim.	Task	DOF		Dec.	Occ.	Metric	
		Original	Trans.			Im.	Para.
2D	2D SHAPE	800	4,000	×	×	MSE	—
3D	3D SHAPE	3,840	3,840	×	×	IoU	Chamf.
3D	POSE	4	4	✓	×	IoU	Angle
1D	OCCLUSION	3	8	×	✓	MSE	Depth

meta-learned and grid-searched softness to a different EGSR logo, requiring optimization of additional initial triangles to reshape the target.

3D Shape: We begin by optimizing a mesh sphere to match the silhouette of an airplane within 100 steps. To evaluate this, we render the airplane from five different elevation angles $(([-60°, -30°, 0°, 30°, 60°]))$ and sample 24 azimuthal views at each elevation, averaging the loss. We compare the performance of our MLP with meta-learned parameters against CDFs with grid-searched softness. While the *original* variant uses an airplane, the *transfer* variant tests the method on a chair.

Pose: In this scenario, the geometry remains fixed while we optimize the camera's pose; the up and look-at directions are set constants. The camera position is represented in spherical coordinates. We randomly initialize the viewing angle between $[10°, 30°]$ and distance from $[2, 8]$. Azimuth and elevation are normally randomized from $[15°, 75°]$ and segmented into three parts for evaluation. For each segment, we calculate the average loss across 200 pairs of reference and initial images. An Adam optimizer ($\beta_1 = 0.9, \beta_2 = 0.99, \epsilon = 10^{-8}$) with different learning rates (0.03 and 0.3) is employed for optimizing both the meta parameters and the camera pose. The *original* variant involves a cow [CPS13], and the *transfer* variant focuses on a dragon.

Occlusion: Initially, we set three overlapping quadrilaterals with distinct colors and textures, and optimize their depths to achieve the correct occlusion order. Depths are uniformly randomized within $z \in [0.5, 1.5]$. In the *transfer* version, we move the focus of our task from ordering the quadrilaterals to correctly determining the closest face to the scene. The challenge in this version would be having more initial quadrilaterals with complex textures (eight overlapping ones with depths ranging from $z \in [-1.5, 2]$), and finding the correct top face in fewer iterations. The MLP outperforms all the distributions by modifying the order correctly in just 10 iterations.

This setup tests each method's ability to accurately process the anteroposterior relationship, especially when quadrilaterals fully overlap, challenging the capabilities of hard z-buffering and silhouette aggregation. Note that in our optimization process, only the z coordinate of each primitive is adjusted, while the x and y coordinates are held fixed.

4.4. Implementation

We implement the meta-learned soft rasterization with PyTorch, leveraging CUDA extensions to compute gradients precisely for high efficiency. All the experiments run on a NVIDIA V100 SXM2 16GB GPU and we use a meta-learning technique that is similar to traditional MAML to learn our meta parameters, which represent the softness of different distributions both in 2D and depth space. The full algorithm is outlined in Algo. 1.

Note that the optimization is not supervised by the ground truth parameters at any point.

In the SHAPE and POSE tasks, we apply a hard z-buffer. In the OCCLUSION task, we use the same edge function for all variants of depth, to solely focus on their performance as soft depth functions. In all tasks, we use probabilistic sum as silhouette aggregation [LLCL19] for all methods.

4.5. Results

We report first quantitative and later qualitative results of our approach.

Quantitative: Quantitative results are seen in Fig. 4 and Tab. 3, where we analyze all tasks according to all metrics using all methods. In Fig. 4, a successful method will —according to both metrics— have a graph that quickly and reliably goes to a low error value; and also stays down. In Tab. 3, lower is better.

We see that across the tasks, and consistent between metrics, our `MLP` performs best (blue). This is true both for the endpoint (right in each plot and values in Tab. 3), as for most (interruptible) in-between iterations as well. For some iterations (horizontal axis in each plot), all methods perform similarly in most tasks and for both metrics, but eventually, meta-learned methods take the lead while others plateau. In some examples, existing methods could not solve the task with the published default values at all, while ours can adapt to any task on any scale. We also see, that while our work optimizes the image error, the unseen parameter also converges to its lowest, which is the ultimate objective in reverse rendering tasks.

For OCCLUSION, MLP performs consistently better compared to other distributions. While a convergence seems possible in the original version based on the image, the parameter error shows otherwise, meaning that the image error is not enough for these distributions to learn depth.

In all experiments, the transfer from learning on one task and testing on another can succeed, as deduced from comparing the first and the second pair of rows, in which the first task class is seen in meta-training, while the second one of the pair is not. This shows the potential to save computational resources on optimizing complex tasks by training on related simpler ones.

Qualitative: Similarly, the qualitative results of the same tasks are illustrated in Fig. 5, and Fig. 6, comparing the renderings of the final iteration from Fig. 4. As shown in the Error results, `MLP` performs best. For SHAPE and POSE tasks, our results match the reference better than the previous methods starting from the same initialization. Additionally, the successful transfer of meta-parameters from one task to another is evident in each pair of columns. Despite the substantial differences in 3D shapes and discrepancies in DOF, the meta-learned softness effectively facilitates a transfer and achieves the lowest error.

Note that for POSE task, GenDR gives unsatisfactory results, which are faithfully obtained by directly running the code released by them. The reason why their optimization ended up so bad is that they manually set the range of dynamic softness from 10^{-1} to 10^{-7}. As shown in the supplementary materials, their method achieves optima when the softness decays to about 10^{-5}, but it finally gets worse as the softness keeps decaying and becomes improper.

This is one of the drawbacks of manually designed softness, even if GenDR sets higher softness at the early stage and decreases it as the solution converges, it's still unknown when and where the softness reaches the optima. However, our meta-learned softness does not have such problems and limitations but also shows a coarse-to-fine transition in practice (if set to higher initially).

Table 3: *Image and parameter error for different tasks (columns) and different methods (rows). The best method is shown in* ***bold font.***

	2D Shape				3D Shape				Pose				Occlusion			
	Original		Transfer		Original		Transfer		Original		Transfer		Original		Transfer	
	Im.	Para.	Im.	Para.	Im.	Para.	Im.	Para.	Im.	Para.	Im.	Para.	Im.	Para.	Im.	Para.
MLP	**0.0038**	—	**0.0038**	—	**19.4**	**3.2**	**69.0**	4.72	**1.1**	**0.01**	**2.17**	0.14	**0.041**	**0.244**	**0.040**	1.63
Gauss	0.0046	—	0.0187	—	31.2	11.9	79.0	4.49	15.3	0.19	15.98	0.14	0.049	0.962	0.058	7.63
Log	0.0056	—	0.0179	—	29.8	10.1	97.0	4.54	16.2	0.22	16.63	**0.12**	0.048	0.916	0.055	7.51
Exp	0.0054	—	0.0148	—	32.0	11.3	87.0	**4.35**	15.9	0.18	14.98	0.15	0.049	0.627	0.046	**1.40**
Gamma	0.0056	—	0.0204	—	28.7	10.0	86.0	4.58	14.5	0.24	15.14	0.24	0.049	0.627	0.046	**1.40**

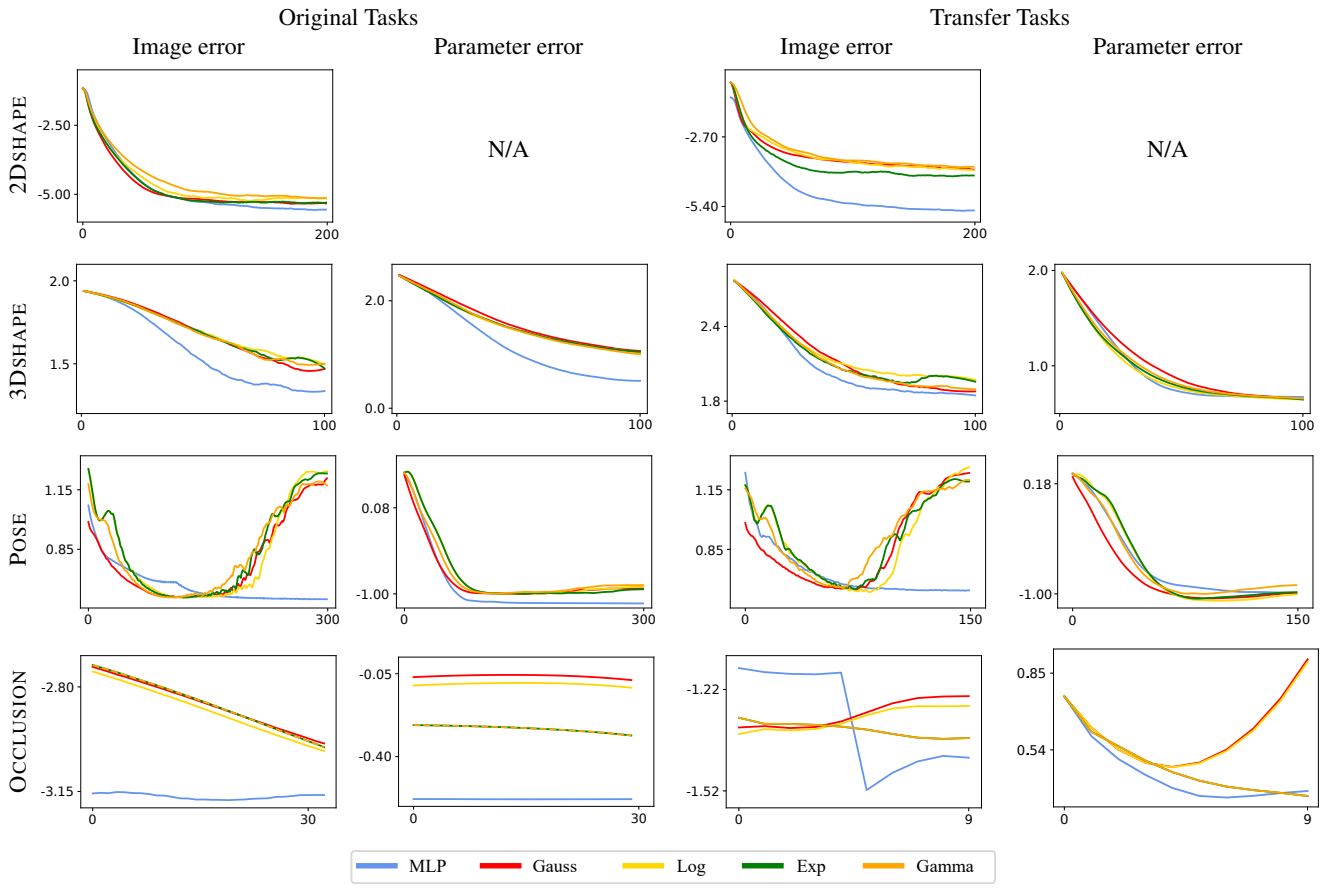


Figure 4: *Every subplot shows the* ***convergence of one inverse rendering task*** *according to one metric where different colors represent different methods. Within each subplot the vertical axis is loss, so less is better (log scale). The horizontal axis is optimization iterations. The first two columns show a training variant, the last two columns show a transfer condition. In each horizontal pair, the first plot is the image-based metric, the second one is the parameter error.*

4.6. Single-View 3D Mesh Reconstruction

For this final experiment, we follow the same auto-encoder structure, as proposed in all preceding studies ([KUH18, LLCL19, CLG*19, PBKD21, PGBD22]), and constrict the number of renderers to one with the distribution function using MLP and probabilistic T-conorm. Since the implementation of nested loops is not time-efficient due to the task's complexity, we optimize the parameters of our MLP with the same Adam optimizer used for the model's parameters. While we have the disadvantage of not reaching convergence for our MLP, in Tab. 4, we show that our results are comparable with the best results reported in GenDR, which is searched over a large set of distribution functions and T-Conorms.

Table 4: ***Comparison of 3D IoU for* SINGLE-VIEW 3D MESH RECONSTRUCTION *in GenDR and our method.*** *The results from GenDR are the best results reported in their paper, comparing a total of 30 renderers, while ours represent a single instance of renderer with MLP as the distribution function.*

Method	Airplane	Bench	Dresser	Car	Chair	Display	Lamp	Speaker	Rifle	Sofa	Table	Phone	Vessel	Mean
GenDR [PGBD22] (best results)	0.6473	0.5026	0.7175	0.7696	0.5297	0.6147	0.4665	0.6673	0.6773	0.6879	0.4961	0.8189	0.6006	0.6232
MLP + Probabilistic	0.6385	0.4755	**0.7216**	0.6849	0.5086	0.6033	0.4594	0.6671	0.6613	0.6449	0.4617	0.7890	0.5829	0.6076
Difference	0.0088	0.0271	-0.0041	0.0847	0.0211	0.0114	0.0071	0.0002	0.0160	0.0430	0.0344	0.0299	0.0177	0.0156

4.7. Time Performance

For the time performance of training, taking the 3D SHAPE task as an example, GenDR employs coarse-to-fine grid-searching in 28 iterations. Our method meta-learns an MLP in 50 iterations but achieves better softness and results without manual range or fineness presumptions.

For the time performance of evaluation, our method only replaces a CDF with a small MLP with 56 parameters, which only contributes to less than 1% of the whole computational graph.

These claims can be further proved by comparing the computational costs of training, evaluating, and rendering. The time costs (seconds) corresponding to Tab. 3 are shown in Tab. 5.

From Tab. 5 we can see that the MLP's extra time cost is negligible compared to the whole Evaluate Time, which includes forward rendering, backward pass, loss calculation, etc. As analyzed above, the time cost of MLP compared to GenDR for other tasks is similar. For Sec. 4.6, the extra cost introduced by MLP is less and our method will be faster than grid-searching across 30 different renderers - while being slightly slower than every single renderer.

Table 5: *The time(s) cost of each method in* 3D SHAPE.

Distribution	Train	Evaluate	Render	Image Loss
MLP	229.2	4.36	0.088	69.0
Gauss	74.95	3.24	0.053	79.0
Log	83.9	3.34	0.059	97.0
Exp	87.7	3.51	0.057	87.0
Gamma	94.5	3.63	0.056	86.0

4.8. Cross Evaluation

Tab. 6 shows what happens when a softness meta-trained from one task is used for another task. The rows stand for the tasks on which MLPs are trained, the columns refer to the evaluated tasks.

Table 6: *The loss of using softness meta-trained from each task for other tasks.*

	Train			
Evaluate	2D SHAPE	3D SHAPE	OCCLUSION	POSE
2D SHAPE	**1** × **14.7**	1.3 × 19.4	9.9 × 0.042	1.8 × 2.4
3D SHAPE	1.2 × 14.7	**1** × **19.4**	9.9 × 0.042	1.4 × 2.4
OCCLUSION	53.7 × 14.7	2.1 × 19.4	**1** × **0.042**	1.7 × 2.4
POSE	1.1 × 14.7	1.1 × 19.4	9.9 × 0.042	**1** × **2.4**

The lowest image loss for each task is on the diagonal, which means a softness learned on one task works better on this task than any other task, and this is true for all tasks, respectively, showing that meta-learning is effective. Note that here we use static softness for POSE task.

5. Conclusion and discussion

We introduced the application of meta-learning to acquire knowledge from a continuous space of softening operations. This approach is used to soften edges and occlusion functions, which in turn improves differentiable rendering. Our approach utilizes a tunable MLP for space and depth edge functions, allowing for joint optimization of their parameters alongside scene parameters. This addresses issues related to discontinuities, enables the optimization of shape and pose, and resolves occlusions in depth.

Additionally, we have investigated the generalization capabilities of meta-learned softening operations, demonstrating the potential of our method to tackle complex rendering problems. The results underscore the transformative impact of our approach on complex differentiable rendering techniques.

Furthermore, we have explored adapting softening functions dynamically based on the task, moving away from a one-size-fits-all approach to one that adjusts softness based on specific scene requirements. Compared to manually designing the range and decreasing step size, our method shows more robustness and adaptability. Both static and dynamic strategies of softness can work, but it's a trade-off. Dynamic softness could be more precise but also needs more computations.

In future work, we plan to extend this technique to broader applications where discontinuities in integrands hinder differentiation and optimization, such as in physical differential equations and dynamic optimization. We envision meta-learning different softness for different optimization stages (from earlier to later) to capture different frequency detail levels. This strategy suggests that softening could evolve into a more high-dimensional and complex process as the dynamic transition has not yet been proven to be necessarily monotonic, potentially making it better suited for implementation with an MLP. Similarly, the combination strategy of multiple softness in multi-task optimization is also a good direction to explore.

Moreover, this technique can be further improved by using the neural representation or neural proxy methods to provide heuristic gradients for the rasterization instead of analytical approaches as explored in this work.

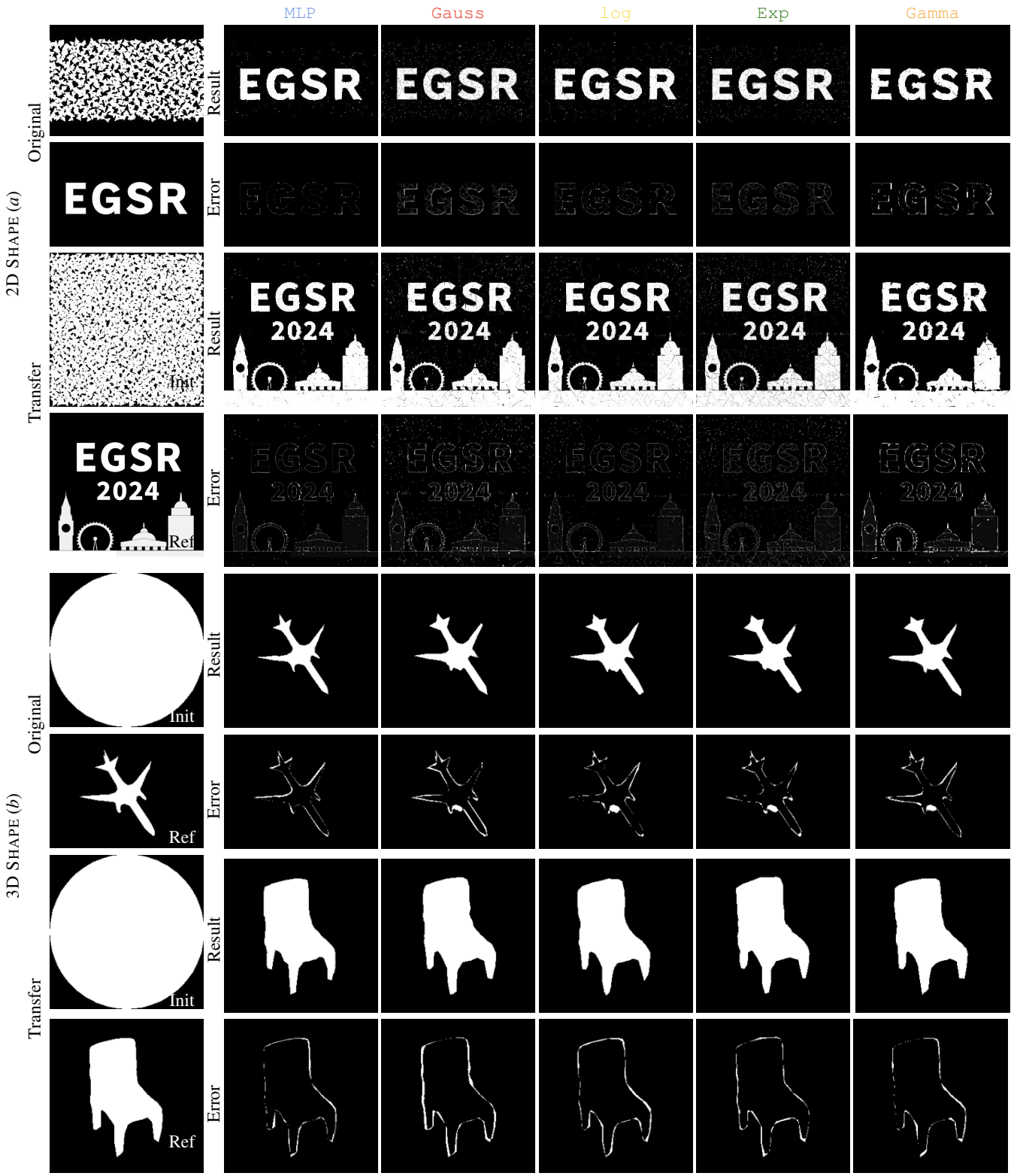


Figure 5: ***Results of different methods for the* 2D SHAPE *and* 3D SHAPE *task.*** *Every pair of rows is one task. The first two pairs are 2D tasks, the second two pairs are 3D tasks. Every even rows show a rendering of the result upon convergence, except for the first column, where we show the initialization (random tris in 2D tasks and a sphere in 3D tasks). Every odd row, shows the error image of that, except for the first column, which shows the reference, the target. A successful optimization would have a black error image and a result that looks similar to the reference. Please refer to the supplementary materials for more analysis.*

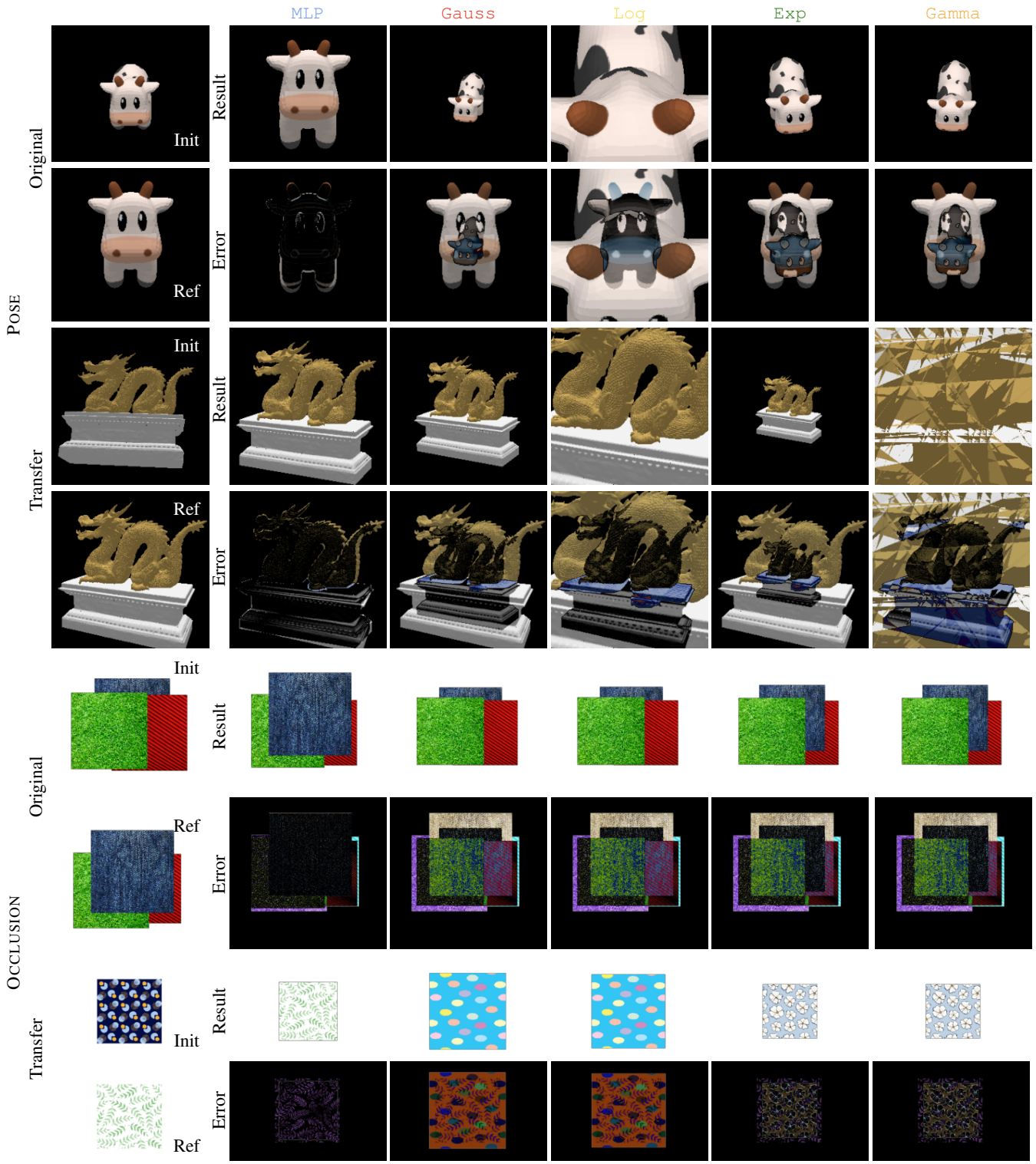


Figure 6: ***Results of different method for the* POSE *and* OCCLUSION *task.*** *Every pair of rows is one task. The first two pairs are* POSE *tasks, the second two pairs are occlusion tasks. Every even rows show a rendering of the result upon convergence, except for the first column, where we show the initialization (random poses in the* POSE *task and random quads in the* OCCLUSION *task). Every odd row, shows the error image of that, except for the first column, which shows the reference, the target. A successful optimization would have a black error image and a result that looks similar to the reference. Please refer to the supplementary materials for more analysis.*

References

[BLD20] BANGARU S., LI T.-M., DURAND F.: Unbiased warped-area sampling for differentiable rendering. *ACM Trans. Graph. 39*, 6 (2020), 245:1–245:18. 2

[CLG*19] CHEN W., LING H., GAO J., SMITH E., LEHTINEN J., JACOBSON A., FIDLER S.: Learning to predict 3d objects with an interpolation-based differentiable renderer. In *NeurIPS* (2019). 2, 7

[CLG*21] CHEN W., LITALIEN J., GAO J., WANG Z., TSANG C. F., KHAMIS S., LITANY O., FIDLER S.: DIB-R++: Learning to predict lighting and material with a hybrid differentiable renderer, 2021. 1

[CPS13] CRANE K., PINKALL U., SCHRÖDER P.: Robust fairing via conformal curvature flow. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 1–10. 6

[EVGW*10] EVERINGHAM M., VAN GOOL L., WILLIAMS C. K. I., WINN J., ZISSERMAN A.: The pascal visual object classes (voc) challenge. *International Journal of Computer Vision 88*, 2 (2010), 303–338. 5

[FAL17] FINN C., ABBEEL P., LEVINE S.: Model-agnostic meta-learning for fast adaptation of deep networks. In *ICLR* (2017), pp. 1126–1135. 3

[FR23] FISCHER M., RITSCHEL T.: Plateau-reduced differentiable path tracing. In *CVPR* (2023). 1

[GMCG20] GUPTA A., MEDHI J., CHATTOPADHYAY A., GUPTA V.: End-to-end differentiable 6DoF object pose estimation with local and global constraints, 2020. 1

[KBM*20] KATO H., BEKER D., MORARIU M., ANDO T., MATSUOKA T., KEHL W., GAIDON A.: Differentiable rendering: A survey, 2020. 3

[KUH18] KATO H., USHIKU Y., HARADA T.: Neural 3D mesh renderer. In *CVPR* (2018), pp. 3907–3916. 1, 7

[LADL18] LI T.-M., AITTALA M., DURAND F., LEHTINEN J.: Differentiable monte carlo ray tracing through edge sampling. *ACM Trans. Graph. (Proc. SIGGRAPH) 37*, 6 (2018). 1, 2

[LB14] LOPER M. M., BLACK M. J.: Opendr: An approximate differentiable renderer. In *ECCV* (2014), pp. 154–169. 1

[LHJ19] LOUBET G., HOLZSCHUCH N., JAKOB W.: Reparameterizing discontinuous integrands for differentiable rendering. *Transactions on Graphics (Proceedings of SIGGRAPH Asia) 38*, 6 (Dec. 2019). doi:10.1145/3355089.3356510. 2

[LHK*20] LAINE S., HELLSTEN J., KARRAS T., SEOL Y., LEHTINEN J., AILA T.: Modular primitives for high-performance differentiable rendering. *ACM Trans. Graph. 39*, 6 (nov 2020). URL: https://doi.org/10.1145/3414685.3417861, doi:10.1145/3414685.3417861. 3

[LLCL19] LIU S., LI T., CHEN W., LI H.: Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *CVPR* (2019), pp. 7708–7717. 1, 2, 3, 4, 6, 7

[NDDJK21] NIMIER-DAVID M., DONG Z., JAKOB W., KAPLANYAN A.: Material and lighting reconstruction for complex indoor scenes with texture-space differentiable rendering. In *Proc. EGSR* (2021), pp. 73–84. 1

[PBDC19] PETERSEN F., BERMANO A. H., DEUSSEN O., COHEN-OR D.: Pix2vex: Image-to-geometry reconstruction using a smooth differentiable renderer. *CoRR abs/1903.11149* (2019). arXiv:1903.11149. 2

[PBKD21] PETERSEN F., BORGELT C., KUEHNE H., DEUSSEN O.: Learning with algorithmic supervision via continuous relaxations. In *NeurIPS* (2021). 1, 7

[PGBD22] PETERSEN F., GOLDLUECKE B., BORGELT C., DEUSSEN O.: GenDR: A generalized differentiable renderer. In *CVPR* (June 2022), pp. 4002–4011. 3, 4, 7, 8

[Pin88] PINEDA J.: A parallel algorithm for polygon rasterization. *SIGGRAPH* (1988). 2

[RRR*15] RHODIN H., ROBERTINI N., RICHARDT C., SEIDEL H.-P., THEOBALT C.: A versatile scene model with differentiable visibility applied to generative pose estimation. *ICCV* (2015), 765–773. 1, 2

[YBAF22] YANG Y., BARNES C., ADAMS A., FINKELSTEIN A.: Aδ: Autodiff for discontinuous programs – applied to shaders. In *ACM SIGGRAPH, to appear* (Aug. 2022). 2

[ZMY*20] ZHANG C., MILLER B., YAN K., GKIOULEKAS I., ZHAO S.: Path-space differentiable rendering. *ACM Trans. Graph. 39*, 4 (2020). 2