Differentiable Rendering with Perturbed Optimizers

Quentin Le Lidec, Ivan Laptev, Cordelia Schmid, Justin Carpentier

Differentiable Rendering with Perturbed Optimizers

Quentin Le Lidec*, **Ivan Laptev***, **Cordelia Schmid***, **Justin Carpentier***

Abstract

Reasoning about 3D scenes from their 2D image projections is one of the core problems in computer vision. Solutions to this inverse and ill-posed problem typically involve a search for models that best explain observed image data. Notably, images depend both on the properties of observed scenes and on the process of image formation. Hence, if optimization techniques should be used to explain images, it is crucial to design differentiable functions for the projection of 3D scenes into images, also known as differentiable rendering. Previous approaches to differentiable rendering typically replace non-differentiable operations by smooth approximations, impacting the subsequent 3D estimation. In this paper, we take a more general approach and study differentiable renderers through the prism of randomized optimization and the related notion of perturbed optimizers. In particular, our work highlights the link between some well-known differentiable renderer formulations and randomly smoothed optimizers, and introduces *differentiable perturbed renderers*. We also propose a variance reduction mechanism to alleviate the computational burden inherent to perturbed optimizers and introduce an adaptive scheme to automatically adjust the smoothing parameters of the rendering process. We apply our method to 3D scene reconstruction and demonstrate its advantages on the tasks of 6D pose estimation and 3D mesh reconstruction. By providing informative gradients that can be used as a strong supervisory signal, we demonstrate the benefits of perturbed renderers to obtain more accurate solutions when compared to the state-of-the-art alternatives using smooth gradient approximations.

1 Introduction

Many common tasks in computer vision such as 3D shape modelling [5, 15, 34, 36, 37] or 6D pose estimation [18, 22, 25, 30, 32] aim at inferring 3D information directly from 2D images. Most of the recent approaches rely on (deep) neural networks and thus require large training datasets with 3D shapes along with well-chosen priors on these shapes. Render & compare methods [18, 22] circumvent the non-differentiability of the rendering process by learning gradients steps from large datasets. Using a more structured strategy would allow to alleviate the need for such a strong supervision. In this respect, differentiable rendering intends to model the effective image generation process to compute a gradient related to the task to solve. This approach has the benefit of containing the prior knowledge of the rendering process while being interpretable. This makes it possible to provide a supervision for neural networks in a weakly-supervised manner [10, 16, 23]. The main challenge of differentiable rendering lies in the non-smoothness of the classical rendering process. In a renderer, both the rasterization steps, which consist in evaluating the pixel values by discretizing the 2D projected color-maps, and the aggregation steps, which merge the color-maps of several objects along the depth dimension by using a Z-buffering operation, are non-differentiable operations (Fig. 1). Intuitively, these steps imply discontinuities in the final rendered image with respect to the 3D positions of the scene objects. For example, if an object moves on a plane parallel to the

*The authors are with Inria and Département d'Informatique de l'Ecole Normale Supérieure, PSL Research University, Paris, France. firstname.lastname@inria.fr

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

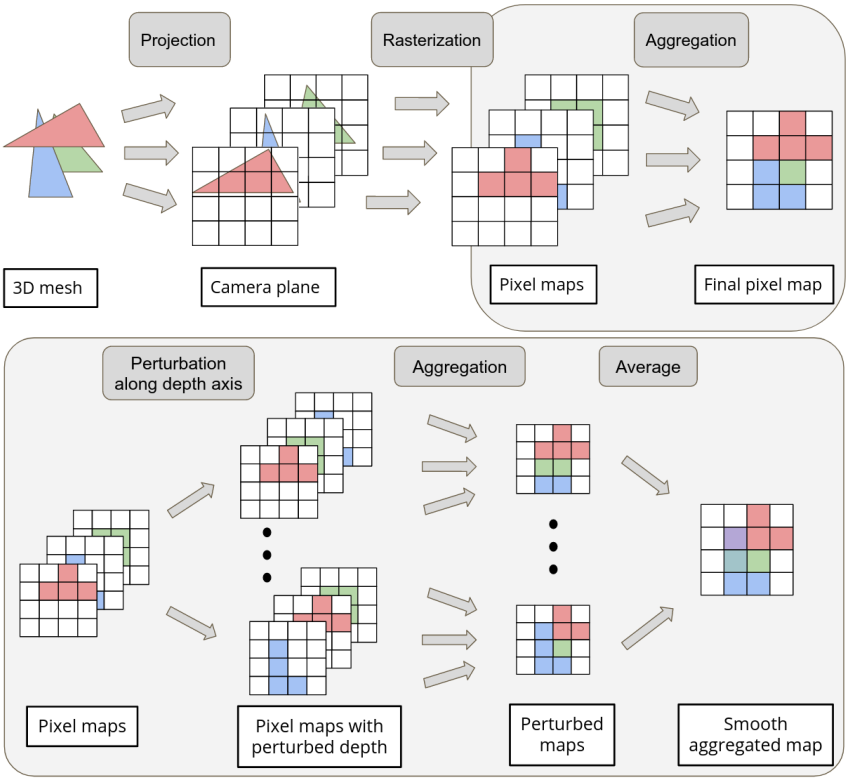

Figure 1: **Top**: Overview of the rendering process: both rasterization and aggregation steps induce non-smoothness in the computational flow. **Bottom:** Illustration of the differentiable perturbed aggregation process. The rasterization step is made differentiable in a similar way.

camera, some pixels will immediately change color at the moment the object enters the camera view or becomes unocluded by another object.

In this paper, we propose to exploit randomized smoothing techniques within the context of differentiable rendering to automatically soften the non-smooth rendering operations, making them naturally differentiable. The generality of our approach offers a theoretical understanding of some of the existing differentiable renderers while its flexibility leads to competitive or even state-of-the-art results in practice. We make the following contributions:

- ↪ We formulate the non-smooth operations occurring in the rendering process as solutions of optimization problems and, based on recent work [4], we propose a natural way of smoothing them using random perturbations. We highlight the versatility of this smoothing formulation and show how it offers a theoretical understanding of several existing differentiable renderers.
- ↪ We propose a general way to use control variate methods to reduce variance when estimating the gradients of the perturbed optimizers, which allows for sharp gradients even in the case of weaker perturbations.
- ↪ We introduce an adaptive scheme to automatically adjust the smoothing parameters by relying on sensitivity analysis of differentiable perturbed renderers, leading to a robust and adaptive behavior during the optimization process.
- ↪ We demonstrate through experiments on pose optimization and 3D mesh reconstruction that the resulting gradients combined to the adaptive smoothing provide a strong signal, leading to more accurate solutions when compared to state-of-the-art alternatives based on smooth gradient approximations [23].

2 Background

In this section, we review the fundamental aspects behind image rendering and recall the notion of perturbed optimizers which are at the core of the proposed approach.

Rasterizer and "aggregater" as optimizers. We consider the rendering process of an RGB image of height h_I and width w_I, from a scene composed with meshes represented by a set of m triangles. We assume that every triangle has already been projected onto the camera plane and their associated color maps $C_j \in \mathbb{R}^{h_I \times w_I \times 3}$, $j \in [1 \,..\, m]$, have been computed using a chosen illumination model (Phong [29], Gouraud [13] etc.) and interpolating local properties of the mesh. We denote by I the occupancy map such that I^i_j is equal to 1 if the center of the i^{th} pixel is inside the 2D projection of the j^{th} triangle on the camera plane and 0 otherwise. By denoting $d(i,j)$ the Euclidean distance from the center of the i^{th} pixel to the projection of the j^{th} triangle, the rasterization step (which actually consists in computing the occupancy map I) can be written as:

$$I^i_j = H(d(i,j)), \tag{1}$$

where H corresponds to the Heaviside function defined by:

$$H(x) = \begin{cases} 0 \text{ if } x \leq 0 \\ 1 \text{ otherwise} \end{cases} = \underset{0 \leq y \leq 1}{\text{argmax}}\; y\, x. \tag{2}$$

We call $R \in \mathbb{R}^{h_I \times w_I \times 3}$ the final rendered image which is obtained by aggregating the color map of each triangle. In classical renderers, this step is done by using a Z-buffer so that only foreground objects are visible. This corresponds to:

$$R^i = \sum_j w^i_j(z) C^i_j, \;\; \text{with} \;\; w^i(z) = \underset{y \text{ s.t. } \|y\|_1 = 1, y \geq 0,\; y_j = 0 \text{ if } I^i_j = 0}{\text{argmax}} \langle z, y \rangle, \tag{3}$$

where $w, z \in \mathbb{R}^{m+1}$ and for $1 \leq j \leq m$, z_j is the inverse depth of the j^{th} triangle and the $(m+1)^{th}$ coordinate of z and w account for the background. The inverse depth of the background is fixed to $z_{m+1} = z_{\min}$. From Eq. (2) and (3), it appears that argmax operations play a central role in renderers and are typically non-differentiable functions with respect to their input arguments, as discussed next.

Perturbed optimizers. In [4], Berthet et al. introduce a generic approach to handle non-smooth problems of the form $y^*(\theta) = \underset{y \in \mathcal{C}}{\text{argmax}} \langle \theta, y \rangle$ with $\mathcal{C}$ a convex polytope, and make these problems differentiable by randomly perturbing them. More precisely, $y^*(\theta)$ necessarily lies on a vertex of the convex set $\mathcal{C}$. Thus, when θ is only slightly modified, y^* remains on the same vertex, but when the perturbation grows up to a certain level, y^* jumps onto another vertex. Concretely, this means that $y^*(\theta)$ is piece-wise constant with respect to θ: the Jacobian $J_\theta y^*$ is null almost everywhere and undefined otherwise. This is why the rasterization (2) and aggregation (3) steps make renderers non-differentiable. Following [4], $y^*(\theta)$ can be approximated by the perturbed optimizer:

$$y^*_\epsilon(\theta) = \mathbb{E}_Z[y^*(\theta + \epsilon Z)] \tag{4}$$

$$= \mathbb{E}_Z \left[\underset{y \in \mathcal{C}}{\text{argmax}} \langle \theta + \epsilon Z, y \rangle \right], \tag{5}$$

where Z is a random noise following a distribution of the form $\mu(z) \propto \exp(-\nu(z))$. Then, we have $y^*_\epsilon(\theta) \xrightarrow{\epsilon \to 0} y^*(\theta)$, $y^*_\epsilon(\theta)$ is differentiable and its gradients are non-null everywhere. Intuitively, one can see from (5) that perturbed optimizers y^*_ϵ are actually a convolved version of the rigid optimizer y^*. Moreover, their Jacobian with respect to θ is given by:

$$J_\theta y^*_\epsilon(\theta) = \mathbb{E}_Z \left[\frac{y^*(\theta + \epsilon Z)}{\epsilon} \nabla \nu(Z)^\top \right]. \tag{6}$$

It is worth noticing at this stage that, when $\epsilon = 0$, one recovers the standard formulation of the rigid optimizer. In general, $y^*_\epsilon(\theta)$ and $J_\theta y^*_\epsilon(\theta)$ do not admit closed-form expressions. To overcome this issue, Monte-Carlo estimators are exploited to compute an estimate of these quantities, as recalled in Sec. 4.

3 Related work

Our work builds on results in differentiable rendering, differentiable optimization and randomized smoothing.

Differentiable rendering. Some of the earliest differentiable renderers rely on the rigid rasterization-based rendering process recalled in the previous section, while exploiting some gradient approximations of the non-smooth operations in the backward. OpenDR [24] uses a first-order Taylor expansion to estimate the gradients of a classical renderer, resulting in gradients concentrated around the edges of the rendered images. In the same vein, NMR [16] proposes to avoid the issue of local gradients by doing manual interpolation during the backward pass. For both OpenDR and NMR, the discrepancy between the function evaluated in the forward pass and the gradients computed in the backward pass may lead to unknown or undesired behaviour when optimizing over these approximated gradients. More closely related to our work, SoftRas [23] directly approximates the forward pass of usual renderers to make it naturally differentiable. Similarly, DIB-R [8] introduces the analytical derivatives of the faces' barycentric coordinates to get a smooth rendering of foreground pixels during the forward pass, and thus, gets gradients naturally relying on local properties of meshes. Additionally, a soft aggregation step is required to backpropagate gradients towards background pixels in a similar way to SoftRas. In a parallel line of work, physically realistic renderers are made differentiable by introducing a stochastic estimation of the derivatives of the ray tracing integral [21, 28]. This research direction seems promising as it allows to generate images with global illumination effects. However, this class of renderers remains computationally expensive when compared to rasterization-based algorithms, which makes them currently difficult to exploit within classic computer vision or robotics contexts, where low computation timings matter.

Differentiable optimization and randomized smoothing. More generally, providing ways to differentiate solutions of constrained optimization problems has been part of the recent growing effort to introduce more structured operations in the differentiable programming paradigm [33], going beyond classic neural networks. A first approach consists in automatically unrolling the optimization process used to solve the problem [11, 26]. However, because the computational graph grows with the number of optimization steps, implicit differentiation, which relies on the differentiation of optimality conditions, should be preferred when available [2, 3, 20]. These methods compute *exact* gradients whenever the solution is differentiable with respect to the parameters of the problem. In this paper, we show how differentiating through a classic renderer relates to computing derivatives of a Linear Programming (LP) problem. In this particular case and as recalled in the Sec. 2, solutions of the optimization problem vary in a heavily non-smooth way as gradients are null almost-everywhere and non-definite otherwise, thus, making them hard to exploit within classic optimization algorithms. To overcome this issue, a first approach consists in introducing a gradient proxy by approximating the solution with a piecewise-linear interpolation [35] which can also lead to null gradient. Another approach leverages random perturbations to replace the original problem by a smoother approximation [1, 4]. For our differentiable renderer, we use this later method as it requires only little effort to transform non-smooth optimizers into differentiable ones, while guaranteeing non-null gradients everywhere. Moreover, randomized smoothing has been shown to facilitate optimization of non-smooth problems [12]. Solving a smooth approximation leads to improved performance when compared to methods dealing with the original rigid problem. In addition to its smoothing effect, the addition of noise also acts as an implicit regularization during the training process which is valuable for globalization aspects [4, 9], i.e. converging towards better local optima.

4 Approximate differentiable rendering via perturbed optimizers

In this section, we detail the main contributions of the paper. We first introduce the reformulation of the rasterization and aggregation steps as perturbed optimizers, and propose a variance reduction mechanism to alleviate the computational burden inherent to these kinds of stochastic estimators. Additionally, we introduce an adaptive scheme to automatically adjust the smoothing parameters inherent to the rendering process.

4.1 Perturbed renderer: a general approximate and differentiable renderer

As detailed in Sec. 2, the rasterization step (2) can be directly written as the argmax of an LP. Similarly, the aggregation (3) step can be slightly modified to reformulate it as an argmax of an LP. Indeed, when $y \geq 0$, the hard constraint $y_j = 0$ if $I_j^i = 0$ is equivalent to $I_j^{i^{y_j}} > 0$, which can be

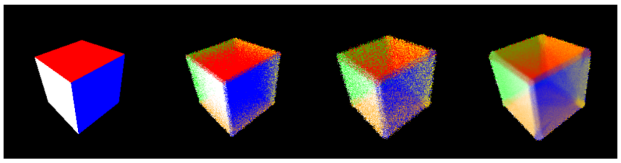

Figure 2: Examples of images obtained from a perturbed differentiable rendering process with a Gaussian noise. With smoothing parameters set to 0, we retrieve the rigid renderer, while adding noise makes pixels from the background appear on the foreground and vice versa.

approximated by adding a logarithmic barrier $y_j \ln I_i^j$ in the objective function as done in classical interior point methods [6]:

$$w_\alpha^i(z) = \underset{y \ s.t. \ \|y\|_1=1, y\geq 0}{\text{argmax}} \langle z + \frac{1}{\alpha}\ln(I^i), y\rangle, \tag{7}$$

because $\ln I_j^i \xrightarrow{I_j^i \to 0} -\infty$, enforcing $w_j^i \xrightarrow{I_j^i \to 0} 0$. $\alpha \to +\infty$ approximates the hard constraint and allows to retrieve the classical formulation (3) of w. Using this formulation, it is possible to introduce a differentiable approximation of the rasterization and aggregation steps:

$$\hat{I}_j^i = H_\sigma(d(i,j)), \text{ with } H_\sigma(x) = \mathbb{E}_X[H(x+\sigma X)] = \mathbb{E}_X[\underset{0\leq y\leq 1}{\text{argmax}} \langle y, x+\sigma X\rangle], \tag{8}$$

$$\hat{R}^i = \sum_j w_{\alpha,\gamma_j}^i(z) C_j^i, \text{ with } w_{\alpha,\gamma}^i(z) = \mathbb{E}_Z[w_\alpha(z+\gamma Z)] \tag{9}$$

$$= \mathbb{E}_Z[\underset{y \ s.t. \ \|y\|_1=1, y\geq 0}{\text{argmax}} \langle y + \frac{1}{\alpha}\ln(\hat{I}^i), z+\gamma Z\rangle]. \tag{10}$$

By proceeding this way, we get a rendering process for which every pixel is influenced by every triangle of meshes (Fig. 1,2), similarly to SoftRas [23]. More concretely, as shown in [4], the gradients obtained with the randomized smoothing are guaranteed to be non-null everywhere unlike some existing methods [8, 16, 24]. This makes it possible to use the resulting gradients as a strong supervision signal for optimizing directly at the pixel level, as shown through the experiments in Sec. 5. At this stage, it is worth noting that the prior distribution on the noise Z used for smoothing is not fixed. One can choose various distributions as a prior thus leading to different smoothing patterns, which makes the approach versatile. Indeed, as a different noise prior induces different smoothing, some specific choices allow to retrieve some of the existing differentiable renderers [16, 23, 24]. In particular, using a Logistic and a Gumbel prior respectively on X and Z leads to SoftRas[23]. Further details are included in Appendix A.

More practically, the smoothing parameters σ, γ appearing in differentiable renderers such as SoftRas [23] or DIB-R [8] can be hard to set. In contrast, in the proposed approach, the smoothing is naturally interpretable as a noise acting directly on positions of mesh triangles. Thus, the smoothing parameters representing the noise intensity can be scaled automatically according to the object dimensions. In Sec. 4.3, we notably propose a generic approach to automatically adapt them along the optimization process.

4.2 Exploiting the noise inside a perturbed renderer

In practice, (5) and (6) are useful even in the cases when the choice of prior on Z does not induce any analytical expression for y_ϵ^* and $J_\theta y_\epsilon^*$. Indeed, Monte-Carlo estimators of these two quantities can be directly obtained from these two expressions:

$$y_\epsilon^M(x) = \frac{1}{M}\sum_{i=1}^M y^*(\theta + \epsilon Z^{(i)}) \tag{11}$$

$$J_\theta \, y_\epsilon^M(z) = \frac{1}{M}\sum_{i=1}^M y^*(\theta+\epsilon Z^{(i)})\frac{1}{\epsilon}\nabla\nu(Z^{(i)})^\top, \tag{12}$$

Table 1: Time and memory complexity of our perturbed renderer for the forward and backward computation during the pose optimization task (5).

# of samples	1	2	8	32	64	SoftRas	Hard renderer
Forward (ms)	30 ($\pm$3)	30 ($\pm$3)	31 ($\pm$3)	31 ($\pm$3)	31 ($\pm$3)	29 ($\pm$3)	29 ($\pm$3)
Backward (ms)	19 ($\pm$1)	19 ($\pm$1)	19 ($\pm$2)	20 ($\pm$1)	22 ($\pm$1)	18 ($\pm$1)	N/A
Max memory (Mb)	217.6	217.6	217.6	303.1	440.0	180.3	178.5

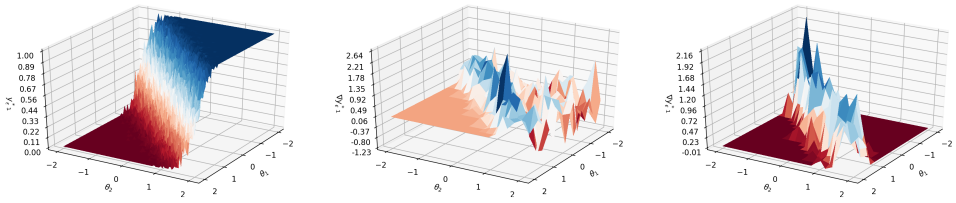

Figure 3: Control variates methods [19] allow to reduce the variance of the gradient of the smooth argmax. **Left**: Gaussian-perturbed argmax operator, **Middle**: estimator of gradient **Right**: variance-reduced estimator of gradient.

where M is the total number of samples used to compute the approximations.

Approximately, we have $\text{Var}\left[J_\theta y_\epsilon^M(z)\right] \propto \frac{\text{Var}\left[y^*(\theta+\epsilon Z)\nabla\nu(Z^{(i)})^\top\right]}{M\epsilon^2}$. When decreasing ϵ, the variance of the Monte-Carlo estimator increases, which can make the gradients very noisy and difficult to exploit in practice. For this reason, we use the control variates method [19] in order to reduce the variance of our estimators. Because the quantity $y^*(\theta)\nabla\nu(Z)^\top$ has a null expectation when Z has a symmetric distribution and is positively correlated with $y^*(\theta+\epsilon Z)\nabla\nu(Z)^\top$, we can rewrite (6) as:

$$J_\theta y_\epsilon^*(\theta) = \mathbb{E}_Z\left[\left(y^*(\theta+\epsilon Z) - y^*(\theta)\right)\nabla\nu(Z)^\top/\epsilon\right], \tag{13}$$

which naturally leads to a variance-reduced Monte-Carlo estimator of the Jacobian:

$$J_\theta y_\epsilon^M(\theta) = \frac{1}{M}\sum_{i=1}^{M}\left(y^*(\theta+\epsilon Z^{(i)}) - y^*(\theta)\right)\frac{1}{\epsilon}\nabla\nu(Z^{(i)})^\top. \tag{14}$$

The computation of y_ϵ^M requires the solution of M perturbed problems instead of only one in the case of classical rigid optimizer. Fortunately, this computation is naturally parallelizable, leading to a constant computation time at the cost of an increased memory footprint. Using our variance-reduced estimator alleviates the need for a high number of Monte Carlo samples, hence reducing the computational burden inherent to perturbed optimizers (Fig. 3). As shown in Sec. 5 (Fig. 5, left), $M = 8$ samples is already sufficient to get stable and accurate results. In addition, computations from the forward pass can be reused in the backward pass and time and memory complexities to evaluate these passes are comparable to classical differentiable renderers (Tab. 1). Consequently, evaluating gradients does not require any extra computation. It is worth mentioning at this stage that this variance reduction mechanism in fact applies to any perturbed optimizer. This variance reduction technique can be interpreted as estimating a sub-gradient with the finite differences in a random direction and is inspired by the field of random optimization [27].

4.3 Making the smoothing adaptive with sensitivity analysis

In a way similar to (6), it is possible to formulate the sensitivity of the perturbed optimizer with respect to the smoothing parameter ϵ:

$$J_\epsilon y_\epsilon^*(\theta) = \mathbb{E}_Z\left[y^*(\theta+\epsilon Z)\left(\nabla\nu(Z)^\top Z - 1\right)/\epsilon\right] \tag{15}$$

and, as previously done, we can build a variance-reduced Monte-Carlo estimator of this quantity:

$$J_\epsilon y_\epsilon^M(\theta) = \frac{1}{M}\sum_{i=1}^{M}\left(y^*(\theta+\epsilon Z^{(i)}) - y^*(\theta)\right)\frac{\nabla\nu(Z^{(i)})^\top Z^{(i)} - 1}{\epsilon} \tag{16}$$

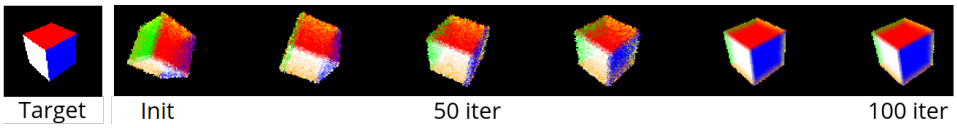

Figure 4: Perturbed differentiable renderer and adaptive smoothing lead to precise 6D pose estimation.

In the case of differentiable rendering, a notable property is the positivity of the sensitivity of any locally convex loss $\mathcal{L}$ (which is valid for the RGB loss) with respect to the γ smoothing parameter when approaching the solution. Indeed, in the neighborhood of the solution, we have $\theta \approx \theta_t$ and a first-order Taylor expansion gives:

$$\frac{\partial \mathcal{L}}{\partial \gamma}(\theta) \approx J_\gamma w_\gamma(z)^\top C^\top \nabla^2 \mathcal{L}(R(\theta_t)) C J_\gamma w_\gamma(z) \geq 0, \tag{17}$$

because for a locally convex loss, $\nabla^2 \mathcal{L}(R(\theta_t))$ is positive definite near a local optimum. We exploit this property to gradually reduce the smoothing when needed. To do so, we track an exponential moving average of the sensitivity during the optimization process:

$$v_\gamma^t = \beta_\gamma v_\gamma^{t-1} + (1 - \beta_\gamma) \frac{\partial \mathcal{L}}{\partial \gamma}(\theta_t) \tag{18}$$

where β_γ is a scalar parameter. Whenever v_γ^t is positive, the smoothing (σ, γ) is decreased at a constant rate (Fig. 4). Note that the adaptive algorithm does not require any further computation as sensitivity can be obtained from backpropagation and also beneficially applies to other differentiable renderers [23] (see 5).

5 Results

In this section, we explore applications of the proposed differentiable rendering pipeline to standard computer vision tasks: single-view 3D pose estimation and 3D mesh reconstruction. Our implementation is based on Pytorch3d [31] and will be publicly released upon publication. It corresponds to a modular differentiable renderer where switching between different type of noise distributions (Gaussian, Cauchy, etc.) is possible. We notably compare our renderer against the open source SoftRas implementation available in Pytorch3d and DIB-R from Kaolin library. The results for these renderers are obtained by running the experiments in our setup as we were not able to get access to the original implementations of the pose optimization and shape reconstruction problems.

Single-view 3D pose estimation is concerned with the retrieving the 3D rotation $\theta \in SO(3)$ of an object from a reference image. Similarly to [23], we first consider the case of a colored cube and fit the rendered image of this cube to a single view of the true pose $R(\theta_t)$ (Fig. 4). The problem can be formulated as a regression problem of the form:

$$\min_\theta \ \mathcal{L}_{RGB}(\theta) = \frac{1}{2} \| R(\theta_t) - \hat{R}(\theta) \|_2^2, \tag{19}$$

where $\hat{R}$ is the smooth differentiable rendered image. We aim at analysing the sensitivity of our perturbed differentiable renderer with respect to random initial guesses of various amplitudes from the real pose. Thus, the initial θ is randomly perturbed from the true pose with various intensity for the perturbation (exploiting an angle-axis representation of the rotation, with a randomized rotation axis). The intensity of the angular perturbation is taken between 20 and 80 degrees. Consequently, the value of the average final error has an increased variance and is often perturbed by local minima (Tab. 2). To avoid these caveats, we rely on another metric: the percentage of solved tasks (see Tab. 2 and Fig. 5 and 6), accounting for the amount of final errors which are below a given threshold (10 degrees for Tab. 2 and bottom row of Fig. 6). For this task, we compare our method with two different distributions for the smoothing noise (Gaussian and Cauchy priors) to SoftRas. We use Adam [17] with parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$ and operate with 128×128 RGB images, each optimization problem taking about 1 minute to solve on a Nvidia RTX6000 GPU.

Table 2: Results of pose optimization from single-view image using perturbed differentiable renderer with variance reduction and adaptive smoothing.

Initial error	20°			50°			80°		
Diff. renderer	SoftRas	Cauchy smoothing	Gaussian smoothing	SoftRas	Cauchy smoothing	Gaussian smoothing	SoftRas	Cauchy smoothing	Gaussian smoothing
Avg. error (°)	5.5	6.1	**3.8**	14.6	20.9	**11.6**	33.5	41.8	**25.3**
Std. error (°)	4.9	5.1	4.4	26.6	21.0	22.5	35.3	34.0	34.9
Task solved (%)	84 ±4.0	82 ±2.3	**93** ±0.4	75 ±5.3	71 ±2.5	**84** ±1.3	55 ±4.8	55 ±5.5	**67** ±2.6

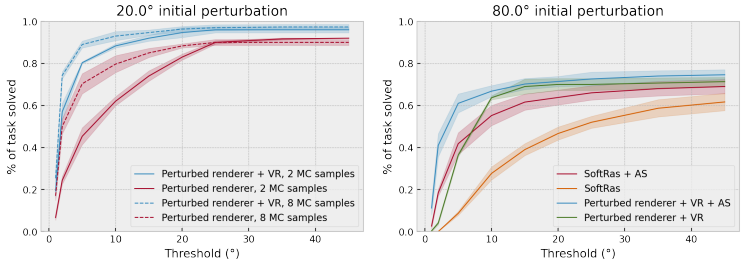

Figure 5: **Left**: Variance reduction (VR) drastically reduces the number of Monte-Carlo samples required to retrieve the true pose from a 20° perturbation. **Right**: Adaptive smoothing (AS) improves resolution of precise pose optimization for a 80° perturbation.

We provide results of 100 random perturbations for various magnitude of initial perturbations in Tab. 2 and Fig. 5,6. Error bars provide standard deviations while running experiments with different random seeds. Fig. 5 (left) demonstrates the advantage of our variance reduction method to estimate perturbed optimizers and precisely retrieve the true pose at a lower computational cost. Indeed, the variance-reduced perturbed renderer is able to perform precise optimization with only 2 samples (80% of final errors are under 5° while it is less than 50% without the use of control variates). Additionally, our adaptive smoothing significantly improves the number of solved tasks for both SoftRas and our perturbed renderer (Fig. 5, right).

Additional results in Tab. 2, Fig. 6 and Fig. 11 in Appendix demonstrate that our perturbed renderers with a Gaussian smoothing, outperforms SoftRas and can achieve state-of-the-art results for pose optimization tasks on various objects. As shown in Appendix (Prop. 4), SoftRas is equivalent to using a Gumbel smoothing which is an asymmetric prior. However, no direction should be preferred when smoothing the rendering process which explain the better performances of the Gaussian prior.

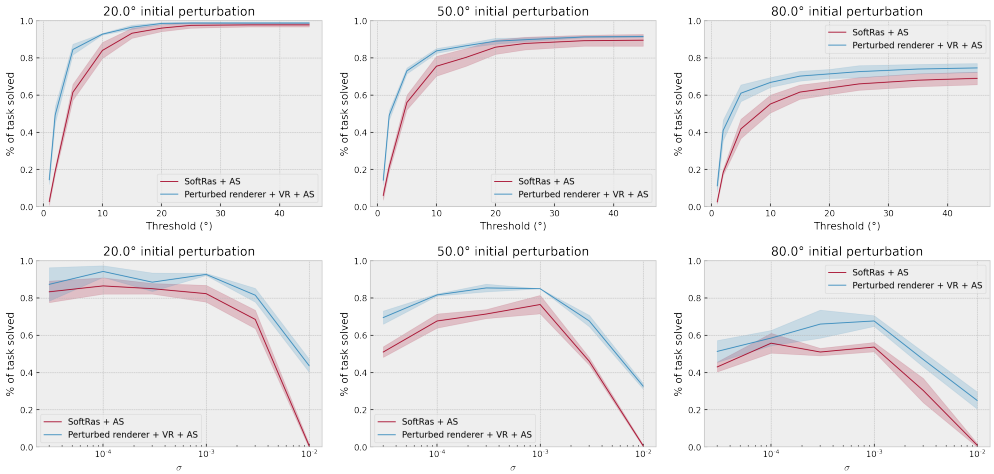

Figure 6: **Top row**: perturbed renderer combined with variance reduction and adaptive scheme improves SoftRas results on pose optimization. **Bottom row**: The method is robust w.r.t. initial smoothing values. Higher is better.

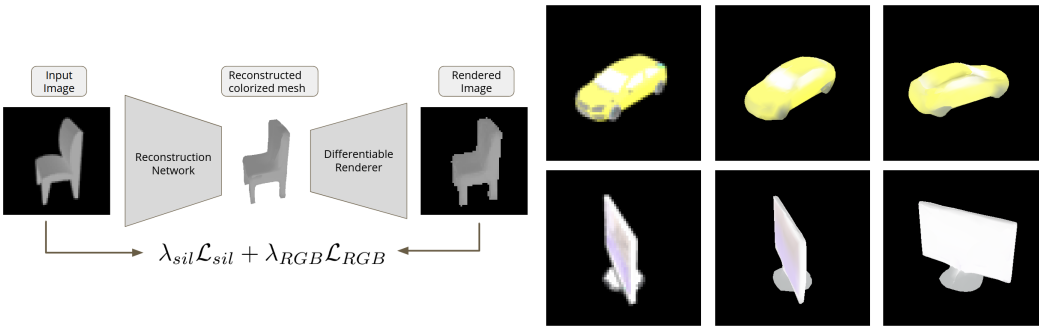

Figure 7: **Left**: A neural network is trained with a self-supervision signal from a differentiable rendering process. **Right**: Qualitative results from self-supervised 3D mesh reconstruction using a perturbed differentiable renderer. $\mathbf{1^{st}}$ **column**: Input image. $\mathbf{2^{nd}}$ **and** $\mathbf{3^{rd}}$ **columns**: reconstructed mesh viewed from different angles.

Table 3: Results of mesh reconstruction on the ShapeNet dataset [7] reported with 3D IoU (%)

	Airplane	Bench	Dresser	Car	Chair	Display	Lamp	Speaker	Rifle	Sofa	Table	Phone	Vessel	Mean
NMR [16]	58.5	45.7	74.1	71.3	41.4	55.5	36.7	67.4	55.7	60.2	39.1	76.2	59.4	57.0
SoftRas [23] [2]	62.0	47.55	66.2	69.4	49.4	60.0	43.3	62.6	61.4	60.4	43.6	76.4	59.9	58.6
DIB-R [8] [2]	59.7	**50.9**	66.2	**72.6**	**52.0**	57.9	43.8	**63.8**	61.0	65.4	**50.5**	76.3	58.6	59.9
Ours	**63.5** (±0.15)	49.4 (±0.08)	**67.1** (±0.32)	72.3 (±0.15)	51.6 (±0.11)	**60.4** (±0.14)	**44.3** (±0.31)	62.8 (±0.24)	**65.7** (±0.09)	**66.7** (±0.12)	49.2 (±0.21)	**80.2** (±0.33)	**60.0** (±0.07)	61.0 (±0.11)

Bottom row of Fig. 6 shows that our method is robust to the choice of initial values of smoothing parameters. During optimization, the stochasticity of the gradient due to the noise from the perturbed renderer acts as an implicit regularization of the objective function. This helps to avoid sharp local minima or saddle points and converge to more stable minimal regions, leading to better optima. As mentioned earlier, results are limited by the inherent non-convexity of the rendering process and differentiable renderers are not able to recover from a bad initial guess. Increasing the noise intensity to further smoothen the objective function would not help in this case, as it would also lead to a critical loss of information on the rendering process.

Self-supervised 3D mesh reconstruction from a single image. In our second experiment, we demonstrate the ability of our perturbed differentiable renderer to provide a supervisory signal for training a neural network. We use the renderer to self-supervise the reconstruction of a 3D mesh and the corresponding colors from a single image (see Fig. 7). To do so, we use the network architecture proposed in [23] and the subset of the Shapenet dataset [7] from [16]. The network is composed of a convolutional encoder followed by two decoders: one for the mesh reconstruction and another one for color retrieving (more details in Appendix B).

Using the same setup as described in [8, 16, 23] , the training is done by minimizing the following loss:

$$\mathcal{L} = \lambda_{sil}\mathcal{L}_{sil} + \lambda_{RGB}\mathcal{L}_{RGB} + \lambda_{lap}\mathcal{L}_{lap}, \tag{20}$$

where $\mathcal{L}_{sil}$ is the negative IoU between silhouettes I and $\hat{I}$, $\mathcal{L}_{RGB}$ is the ℓ_1 RGB loss between R and $\hat{R}$, $\mathcal{L}_{lap}$ is a Laplacian regularization term penalizing the relative change in position between neighbour vertices (detailed description in Appendix B). For training, we use Adam algorithm with a learning rate of 10^{-4} and parameters $\beta_1 = 0.9, \beta_2 = 0.999$. Even if the memory consumption of the perturbed renderer may limit the maximum size of batches, its flexibility makes it possible to switch to a deterministic approximation such as [23] to use larger batches in order to finetune the neural network, if needed.

We analyze the quality of obtained 3D reconstruction through a rendering pipeline by providing IoU measured on 13 different classes of the test set (see Tab. 3). We observe that our perturbed

[2]These numbers were obtained by running the renderer in our own setup (cameras, lightning, training parameters etc.) in order to have comparable results. This may explain the slight difference with the numbers from the original publications.

renderer obtains state-of-the-art accuracies on this task. Error bars (representing the standard deviation obtained by retraining the model with different random seeds) confirm the stability of our method. Additionally, we illustrate qualitative results for colorized 3D mesh reconstruction from a single image in Fig. 7(right) confirming the ability of perturbed renderers to provide strong visual supervision signals.

6 Conclusion

In this paper, we introduced a novel approach to differentiable rendering leveraging recent contributions on perturbed optimizers. We demonstrated the flexibility of our approach grounded in its underlying theoretical formulation. The combination of our proposed perturbed renderers with variance-reduction mechanisms and sensitivity analysis enables for robust application of our rendering approach in practice. Our results notably show that perturbed differentiable rendering can reach state-of-the-art performance on pose optimization and is also competitive for self-supervised 3D mesh reconstruction. Additionally, perturbed differentiable rendering can be easily implemented on top of existing implementations without major modifications, while generalizing existing renderers such as SoftRas. As future work, we plan to embed our generic differentiable renderer within a differentiable simulation pipeline in order to learn physical models, by using visual data as strong supervision signals to learn physics.

Acknowledgments and Disclosure of Funding

We warmly thank Francis Bach for useful discussions. This work was supported in part by the HPC resources from GENCI-IDRIS(Grant AD011012215), the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute), and Louis Vuitton ENS Chair on Artificial Intelligence.

References

[1] J. Abernethy. 1 perturbation techniques in online learning and optimization. 2016.

[2] A. Agrawal, S. Barratt, S. Boyd, E. Busseti, and W. Moursi. Differentiating through a cone program. *Journal of Applied and Numerical Optimization*, 1(2), 2019.

[3] B. Amos and J. Z. Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*. PMLR, 2017.

[4] Q. Berthet, M. Blondel, O. Teboul, M. Cuturi, J.-P. Vert, and F. Bach. Learning with differentiable perturbed optimizers, 2020.

[5] V. Blanz and T. Vetter. A morphable model for the synthesis of 3d faces. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 187–194, 1999.

[6] S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[7] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[8] W. Chen, H. Ling, J. Gao, E. Smith, J. Lehtinen, A. Jacobson, and S. Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[9] J. Cohen, E. Rosenfeld, and Z. Kolter. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning*, pages 1310–1320. PMLR, 2019.

[10] X. Deng, Y. Xiang, A. Mousavian, C. Eppner, T. Bretl, and D. Fox. Self-supervised 6d object pose estimation for robot manipulation. In *International Conference on Robotics and Automation (ICRA)*, 2020.

[11] J. Domke. Generic methods for optimization-based modeling. In N. D. Lawrence and M. Girolami, editors, *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, volume 22 of *Proceedings of Machine Learning Research*, pages 318–326, La Palma, Canary Islands, 21–23 Apr 2012. PMLR.

[12] J. C. Duchi, P. L. Bartlett, and M. J. Wainwright. Randomized smoothing for stochastic optimization. *SIAM Journal on Optimization*, 22(2):674–701, 2012.

[13] H. Gouraud. *Computer Display of Curved Surfaces*. PhD thesis, 1971. AAI7127878.

[14] E. J. Gumbel. *Statistical theory of extreme values and some practical applications: a series of lectures*, volume 33. US Government Printing Office, 1954.

[15] D. Hoiem, A. A. Efros, and M. Hebert. Recovering surface layout from an image. *International Journal of Computer Vision*, 75(1):151–172, 2007.

[16] H. Kato, Y. Ushiku, and T. Harada. Neural 3d mesh renderer, 2017.

[17] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[18] Y. Labbe, J. Carpentier, M. Aubry, and J. Sivic. Cosypose: Consistent multi-view multi-object 6d pose estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[19] A. M. Law, W. D. Kelton, and W. D. Kelton. *Simulation modeling and analysis*, volume 3. McGraw-Hill New York, 2000.

[20] Q. Le Lidec, I. Kalevatykh, I. Laptev, C. Schmid, and J. Carpentier. Differentiable simulation for physical system identification. *IEEE Robotics and Automation Letters*, Feb. 2021.

[21] T.-M. Li, M. Aittala, F. Durand, and J. Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 37(6):222:1–222:11, 2018.

[22] Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox. Deepim: Deep iterative matching for 6d pose estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 683–698, 2018.

[23] S. Liu, T. Li, W. Chen, and H. Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning, 2019.

[24] M. Loper and M. Black. Opendr: An approximate differentiable renderer. 09 2014.

[25] D. G. Lowe. Three-dimensional object recognition from single two-dimensional images. *Artificial intelligence*, 31(3):355–395, 1987.

[26] D. Maclaurin, D. Duvenaud, and R. Adams. Gradient-based hyperparameter optimization through reversible learning. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2113–2122, Lille, France, 07–09 Jul 2015. PMLR.

[27] Y. Nesterov and V. Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566, 2017.

[28] M. Nimier-David, D. Vicini, T. Zeltner, and W. Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *ACM Transactions on Graphics (TOG)*, 38(6):1–17, 2019.

[29] B. T. Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317, June 1975.

[30] M. Rad and V. Lepetit. Bb8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3828–3836, 2017.

[31] N. Ravi, J. Reizenstein, D. Novotny, T. Gordon, W.-Y. Lo, J. Johnson, and G. Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv preprint arXiv:2007.08501*, 2020.

[32] L. G. Roberts. *Machine perception of three-dimensional solids*. PhD thesis, Massachusetts Institute of Technology, 1963.

[33] V. Roulet and Z. Harchaoui. Differentiable programming à la moreau. *arXiv preprint arXiv:2012.15458*, 2020.

[34] A. Saxena, M. Sun, and A. Y. Ng. Learning 3-d scene structure from a single still image. In *2007 IEEE 11th international conference on computer vision*, pages 1–8. IEEE, 2007.

[35] M. Vlastelica, A. Paulus, V. Musil, G. Martius, and M. Rolínek. Differentiation of blackbox combinatorial solvers, 2020.

[36] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 52–67, 2018.

[37] X. Yan, J. Yang, E. Yumer, Y. Guo, and H. Lee. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

Appendices

A Differentiable perturbed renderer

This section describes some possible choice of priors for the smoothing noise of our differentiable perturbed renderers and discuss the eventual links with already existing renderers. In addition, we provide justification of the variance reduction and adaptive smoothing methods for different priors on the noise.

A.1 Perturbed renderers

The following propositions discuss some possible choices of noise prior for the perturbed renderer, exhibiting the links with already existing differentiable renderers [24, 16, 23].

Proposition 1. *The sigmoid function corresponds to the perturbed Heaviside function with a logistic prior on noise:*

$$sigmoid(x) = \mathrm{E}[H(x+Z)] \tag{21}$$

where $Z \sim$ *Logistic*$(0,1)$

Proof. First, we note that the sigmoid function correspond to the first weight from the softmax applied to the R^2 vector $\widetilde{x} = \begin{pmatrix} x \\ 0 \end{pmatrix}$. Indeed, we have :

$$\sigma(x) = \frac{1}{1+\exp(-x)} = \frac{\exp(x)}{\exp(x)+\exp(0)} \tag{22}$$

$$= softmax\left(\widetilde{x}\right)^T \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{23}$$

From the previous proposition, with Z a random variable in R^2 distributed with a Gumbel distribution, we have:

$$softmax(\widetilde{x}) = \mathrm{E}[\underset{y \ s.t. \ \|y\|_1=1}{\operatorname{argmax}} \langle y, \widetilde{x} + \epsilon Z \rangle] \tag{24}$$

$$= \mathrm{E}[\underset{y \ s.t. \ \|y\|_1=1}{\operatorname{argmax}} y^T \begin{pmatrix} x+\epsilon Z_1 \\ \epsilon Z_2 \end{pmatrix}] \tag{25}$$

$$= \mathrm{E}[\underset{y \ s.t. \ \|y\|_1=1}{\operatorname{argmax}} y^T \begin{pmatrix} x+\epsilon(Z_1-Z_2) \\ 0 \end{pmatrix}] \tag{26}$$

because $x + \epsilon(Z_1 - Z_2) > 0$ is equivalent to $x + \epsilon Z_1 > \epsilon Z_2$.

Finally, we can re-write:

$$\sigma(x) = \mathrm{E}[\underset{y \ s.t. \ \|y\|_1=1}{\operatorname{argmax}} y^T \begin{pmatrix} x+\epsilon\widetilde{Z} \\ 0 \end{pmatrix}]^T \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{27}$$

$$= \mathrm{E}[\underset{0 \le y \le 1}{\operatorname{argmax}} y(x+\epsilon\widetilde{Z})] \tag{28}$$

$$= \mathrm{E}[H(x+\epsilon\widetilde{Z})] \tag{29}$$

where $\widetilde{Z}$ follows a logistic law.

Which allows to conclude that the sigmoid corresponds to the Heaviside function perturbed with a logistic noise. We could use the fact that the cumulative distribution function of the logistic distribution is the sigmoid function to make an alternative proof. □

Proposition 2. *The affine approximation of the step function corresponds to the perturbed Heaviside function with an uniform prior on noise:*

$$H_{aff}(x) = \mathrm{E}[H(x+Z)] \tag{30}$$

where $Z \sim \mathcal{U}(-\frac{1}{2}, \frac{1}{2})$

Proof. We have :

$$\begin{aligned}
\mathrm{E}\left[H(x+Z)\right] &= \mathrm{P}(x+Z>0) && (31)\\
&= \mathrm{P}(Z>-x) && (32)\\
&= \begin{cases} 0 & if\ x \leq -\frac{1}{2} \\ x+\frac{1}{2} & if\ -\frac{1}{2} < x < \frac{1}{2} \\ 1 & if\ x \geq \frac{1}{2} \end{cases} && (33)\\
&= H_{aff}(x) && (34)
\end{aligned}$$

□

Proposition 3. *The arctan approximation of the step function corresponds to the perturbed Heaviside function with a Cauchy prior on noise:*

$$\frac{1}{2} + \frac{1}{\pi} arctan(x) = \mathrm{E}[H(x+Z)] \tag{35}$$

where $Z \sim Cauchy(0,1)$

Proof. We have :

$$\begin{aligned}
\mathrm{E}\left[H(x+Z)\right] &= \mathrm{P}(x+Z>0) && (36)\\
&= \mathrm{P}(Z>-x) && (37)\\
&= \int_{-x}^{\infty} \frac{1}{\pi(1+x^2)} \mathrm{d}z && (38)\\
&= \frac{1}{2} + \frac{1}{\pi} arctan(x) && (39)
\end{aligned}$$

□

Proposition 4. *The softmax function corresponds to the perturbed maximal coordinates with a Gumbel prior on the noise:*

$$softmax(z) = \mathrm{E}[\underset{y\ s.t.\ \|y\|_1=1}{\textit{argmax}} \langle y, z+Z\rangle] \tag{40}$$

where $Z \sim$ *Gumbel*$(0,1)$

Proof. The Gumbel-max trick is a classical property of the Gumbel distribution stating: the maximizer of M fixed values z_i which have been perturbed by a noise i.i.d Z_i following a Gumbel distribution, follows an exponentially weighted distribution [14]. This can be written:

$$\mathrm{P}(\max_j z_j + Z_j = z_i + Z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \tag{41}$$

where Z_i are i.i.d. and following a Gumbel distribution. Thus, for a noise Z following a Gumbel distribution, we have :

$$\begin{aligned}
\mathrm{E}[\underset{y\ s.t.\ \|y\|_1=1}{\mathrm{argmax}} \langle y, z+\epsilon Z\rangle] &= \sum_i e_i \mathrm{P}(\underset{y\ s.t.\ \|y\|_1=1}{\mathrm{argmax}} \langle y, z+\epsilon Z\rangle = e_i) && (42)\\
&= \sum_i e_i \mathrm{P}(\max_j z_j + Z_j = z_i + \epsilon Z_i) && (43)\\
&= \sum_i e_i \frac{\exp(z_i)}{\sum_j \exp(z_j)} && (44)\\
&= softmax(z) && (45)
\end{aligned}$$

□

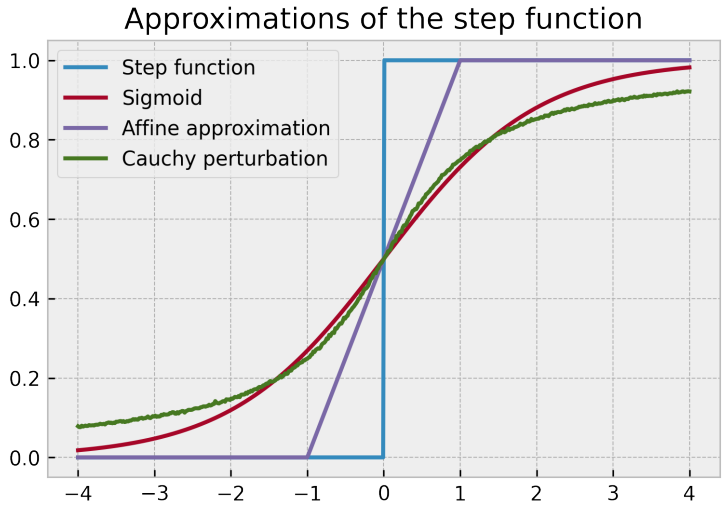

Figure 8: Modifying the prior on the noise leads to different approximations of a non-smooth operator. The sigmoid and affine approximations are obtained by using respectively a Logistic and a Uniform prior.

Following the previous results, one can observe that OpenDR [24] corresponds to a rasterization step where a uniform prior is used for the smoothing noise in order to get gradients during the backward pass. More precisely, the uniform distribution is taken with a support as large as a pixel size which allows to get non null gradients for pixels lying on the border of triangles. This can be linked to antialiasing techniques which are added to classical rendering pipeline to remove high frequencies introduced by the non-smooth rasterization. These techniques make the intensity of a pixel vary linearly with respect to the position of the triangle which can also be seen as an affine approximation of the rasterization. However, one should note that the density of the uniform distribution cannot be written in the form of $\mu(z) \propto \exp(-\nu(z))$ so the properties on differentiability from [4] does not apply. In particular, this approximation is non-differentiable at some points ($x = 1$ or -1) and can have null gradients (for $x < -1$ or $x > 1$) which is inducing localized gradients for OpenDR [24]. For this reason, the approach of NMR [16] actually consists in having a variable range for the distribution support in order to get non-null gradients for a wider region. This approach corresponds to the affine interpolation proposed in [35] and can lead to null gradients. More closely related to our approach, SoftRas smoothen the rasterization by using a sigmoid approximation while Z-buffering is replaced by a softmax during aggregation. As shown by propositions 1 and 4, this approach actually corresponds to a special case of a perturbed renderer where rasterization and aggregation steps were smoothen by using respectively a Logistic and a Gumbel prior for the noise.

In short, this means that some existing renderers [24, 23, 16] can be interpreted in the framework of differentiable perturbed renderers when considering specific priors for the noise used for smoothing. In addition, the generality of the approach allows to consider a continuous range of novel differentiable renderers in between the already existing ones.

A.2 Variance reduction

As introduced in the paper, control variates methods can be used to reduce the noise of the Monte-Carlo estimators of the Jacobian of a perturbed renderer, without inducing any extra-computation. In particular, we prove this in the case of Gaussian and Cauchy priors on the smoothing noise.

Proposition 5. *Jacobian of perturbed renderers can be written as :*

$$J_\theta y_\epsilon^*(\theta) = \mathbb{E}_Z \left[\left(y^*(\theta + \epsilon Z) - y^*(\theta) \right) \nabla \nu(Z)^\top \frac{1}{\epsilon} \right], \tag{46}$$

when Z follows a Gaussian distribution.

Proof. As done in [4, 1], with a change of variable $z' = \theta + \epsilon z$ we have:

$$J_\theta y_\epsilon^*(\theta) = \frac{\partial}{\partial \theta}\left(\int_{-\infty}^{\infty} y^*(\theta + \epsilon z)\mu(z)\mathrm{d}z\right), \tag{47}$$

$$= \int_{-\infty}^{\infty} y^*(z')\frac{\partial}{\partial \theta}\mu\left(\frac{z'-\theta}{\epsilon}\right)\mathrm{d}z', \tag{48}$$

$$= \int_{-\infty}^{\infty} y^*(z')\frac{1}{\epsilon}\nabla\nu\left(\frac{z'-\theta}{\epsilon}\right)^\top \mu\left(\frac{z'-\theta}{\epsilon}\right)\mathrm{d}z' \tag{49}$$

And by doing the inverse change of variable, we get:

$$J_\theta y_\epsilon^*(\theta) = \mathbb{E}_Z\left[\frac{y^*(\theta+\epsilon Z)}{\epsilon}\nabla\nu(Z)^\top\right] \tag{50}$$

In addition, for Z following a standard Gaussian distribution we have $\nu(Z) = \frac{\|Z\|^2}{2}$ and thus:

$$\mathbb{E}_Z\left[y^*(\theta)\nabla\nu(Z)^\top\right] = y^*(\theta)\mathbb{E}_Z\left[Z^\top\right] \tag{51}$$

and because Z is centered, we get:

$$\mathbb{E}_Z\left[y^*(\theta)\nabla\nu(Z)^\top\right] = 0 \tag{52}$$

Finally we get:

$$J_\theta y_\epsilon^*(\theta) = \mathbb{E}_Z\left[\left(y^*(\theta+\epsilon Z) - y^*(\theta)\right)\nabla\nu(Z)^\top\frac{1}{\epsilon}\right]. \tag{53}$$

□

Proposition 6. *Jacobian of perturbed renderers can be written as :*

$$J_\theta y_\epsilon^*(\theta) = \mathbb{E}_Z\left[\left(y^*(\theta+\epsilon Z) - y^*(\theta)\right)\nabla\nu(Z)^\top\frac{1}{\epsilon}\right], \tag{54}$$

when Z follows a Cauchy distribution.

Proof. Proceeding in a similar way to 5, for a Cauchy distribution we have $\nu(z) = \ln(1+z^2)$, thus

$$\mathbb{E}_Z\left[y^*(\theta)\nabla\nu(Z)^\top\right] = y^*(\theta)\mathbb{E}_Z\left[\frac{2Z^\top}{1+\|Z\|^2}\right] \tag{55}$$

$$= 0 \tag{56}$$

and finally we have:

$$J_\theta y_\epsilon^*(\theta) = \mathbb{E}_Z\left[\left(y^*(\theta+\epsilon Z) - y^*(\theta)\right)\nabla\nu(Z)^\top\frac{1}{\epsilon}\right], \tag{57}$$

for Z following a Cauchy distribution □

A.3 Adaptive smoothing

In the same way as we proceeded for Jacobians computation, control variates method can be used to reduce the variance of Monte-Carlo estimators of the sensitivity of perturbed optimizers with respect to smoothing parameter ϵ.

Proposition 7. *The sensitivity of the perturbed optimizer with respect to the smoothing parameter ϵ can be expressed as:*

$$J_\epsilon y_\epsilon^*(\theta) = \mathbb{E}_Z\left[\left(y^*(\theta+\epsilon Z) - y^*(\theta)\right)\frac{\nabla\nu(Z)^\top Z - 1}{\epsilon}\right] \tag{58}$$

for Z a smoothing noise following a Gaussian prior.

Proof. In a similar way to 5, using the same change of variable, we get:

$$J_\theta y_\epsilon^*(\theta) = \frac{\partial}{\partial \epsilon}\left(\int_{-\infty}^{\infty} y^*(\theta+\epsilon z)\mu(z)\mathrm{d}z\right), \tag{59}$$

$$= \int_{-\infty}^{\infty} y^*(z')\frac{\partial}{\partial \epsilon}\mu\left(\frac{z'-\theta}{\epsilon}\right)\mathrm{d}z', \tag{60}$$

$$= \int_{-\infty}^{\infty} y^*(z')\frac{1}{\epsilon^2}\left(\nabla\nu\left(\frac{z'-\theta}{\epsilon}\right)^\top \frac{z'-\theta}{\epsilon} - 1\right)\mu\left(\frac{z'-\theta}{\epsilon}\right)\mathrm{d}z' \tag{61}$$

which yields with the inverse change of variable:

$$J_\epsilon y_\epsilon^*(\theta) = \mathbb{E}_Z\left[y^*(\theta+\epsilon Z)\frac{\nabla\nu(Z)^\top Z - 1}{\epsilon}\right] \tag{62}$$

Then, for Z following a Gaussian distribution, we have:

$$\mathbb{E}_Z\left[y^*(\theta)\frac{\nabla\nu(Z)^\top Z - 1}{\epsilon}\right] = y^*(\theta)\mathbb{E}_Z\left[\frac{\|Z\|^2 - 1}{\epsilon}\right] \tag{63}$$

$$= 0 \tag{64}$$

for Z following a standard Gaussian noise. Finally, we can write:

$$J_\epsilon y_\epsilon^*(\theta) = \mathbb{E}_Z\left[\left(y^*(\theta+\epsilon Z) - y^*(\theta)\right)\frac{\nabla\nu(Z)^\top Z - 1}{\epsilon}\right] \tag{65}$$

□

Proposition 8. *The sensitivity of the perturbed optimizer with respect to the smoothing parameter ϵ can be expressed as:*

$$J_\epsilon y_\epsilon^*(\theta) = \mathbb{E}_Z\left[\left(y^*(\theta+\epsilon Z) - y^*(\theta)\right)\frac{\nabla\nu(Z)^\top Z - 1}{\epsilon}\right] \tag{66}$$

for Z a smoothing noise following a Cauchy prior.

Proof. We adapt the previous proof to the case of a Cauchy distribution for the noise. We observe that:

$$J_\epsilon y_\epsilon^*(\theta) = \mathbb{E}_Z\left[y^*(\theta+\epsilon Z)\frac{\nabla\nu(Z)^\top Z - 1}{\epsilon}\right] \tag{67}$$

remains valid. In addition, we have :

$$\mathbb{E}_Z\left[y^*(\theta)\frac{\nabla\nu(Z)^\top Z - 1}{\epsilon}\right] = y^*(\theta)\frac{1}{\epsilon}\mathbb{E}_Z\left[\frac{2\|Z\|^2}{1+\|Z\|^2} - 1\right] \tag{68}$$

$$\tag{69}$$

for Z following a Cauchy noise. Moreover, an integration by parts gives:

$$\int_{-\infty}^{\infty}\frac{2x^2}{(1+x^2)^2}\mathrm{d}x = \left[\frac{-x}{1+x^2}\right]_{-\infty}^{\infty} + \int_{-\infty}^{\infty}\frac{1}{1+x^2}\mathrm{d}x \tag{70}$$

$$= [\arctan(x)]]_{-\infty}^{\infty} \tag{71}$$

so we have :

$$\mathbb{E}_Z\left[\frac{2\|Z\|^2}{1+\|Z\|^2} - 1\right] = 0 \tag{72}$$

As previously done in Proposition 7, we finally get:

$$J_\epsilon y_\epsilon^*(\theta) = \mathbb{E}_Z\left[\left(y^*(\theta+\epsilon Z) - y^*(\theta)\right)\frac{\nabla\nu(Z)^\top Z - 1}{\epsilon}\right] \tag{73}$$

□

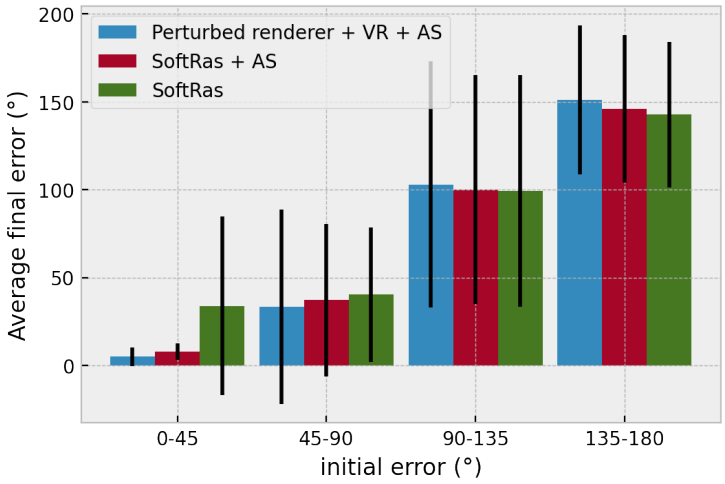

Figure 9: Pose optimization with an initial guess uniformly sampled on the rotation space. Results are reported for our differentiable perturbed renderer with Adaptive Smoothing (AS) and Variance Reduction (VR), and for SoftRas [23] with and without Adaptive Smoothing.

B Experimental results

In this section, we provide details on our experimental setup and some additional results. In addition, we provide the code for our differentiable perturbed renderer based on Pytorch3d [31] and the experiments on pose optimization while additional experiments will be made available upon publication.

B.1 Pose optimization

Trying to solve pose optimization from an initial guess taken uniformly in the rotation space leads to very variable results when measuring the average final error (Fig.9). When the initialization is too far from the initial guess, the optimization process converges towards a local optima. This limitation is inherent to approaches using differentiable rendering to solve pose optimization tasks and could be avoided only by using a combination with other methods, e.g. Render & Compare [18], to get better initial guess for the pose. For this reason, we prefer to consider the ability of our differentiable renderer to retrieve the true pose with an initial guess obtained by perturbing the target and measure the amount of final errors below a given precision threshold (Fig. 10,11).

In addition to the pose optimization task on the cube, we propose to solve the same problem with others objects from Shapenet dataset [7] (Fig. 11).

B.2 Mesh reconstruction

Reconstruction network Network architecture is based on [23] and is represented in Fig. 12. The encoder is composed of 3 convolutional layers (each of them is followed by a batch normalization operations) and 3 fully connected layers. We use two different decoders to retrieve respectively the vertices' position and their color. The positional decoder is composed of 3 fully connected layers. The decoder used for colors retrieval is composed of a branch selecting a palette of N_p colors on the input image while the second branch mixes these colors to colorize each of the N_v vertices.

Training setup We use the same setup as described in [8, 16, 23]. The training is done by minimizing a loss combining silhouette and color information with a regularization term:

$$\mathcal{L} = \lambda_{sil}\mathcal{L}_{sil} + \lambda_{RGB}\mathcal{L}_{RGB} + \lambda_{lap}\mathcal{L}_{lap}, \tag{74}$$

and we set $\lambda_{sil} = 1$, $\lambda_{RGB} = 1$, $\lambda_{lap} = 3.10^{-3}$ during training.

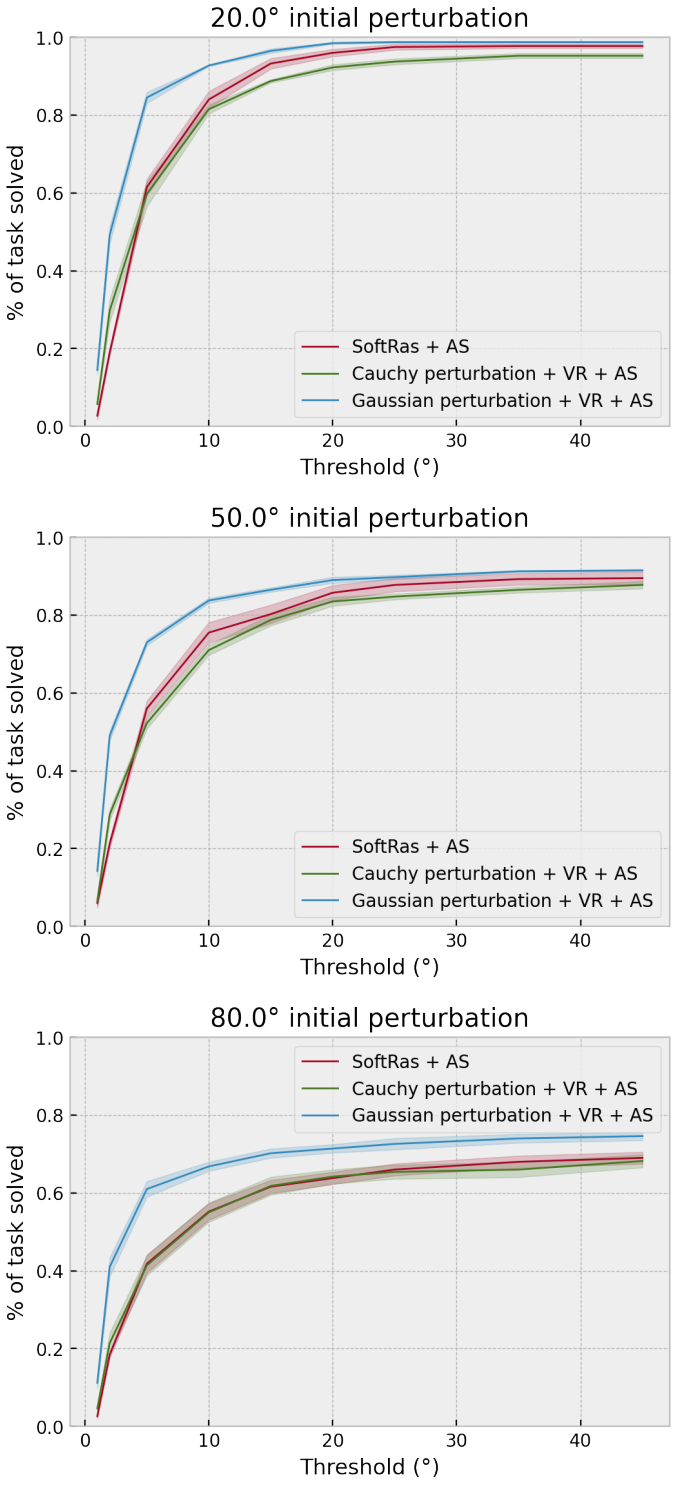

Figure 10: Pose optimization results for a perturbed renderer with Gaussian and Cauchy priors on the smoothing noise, and for SoftRas.

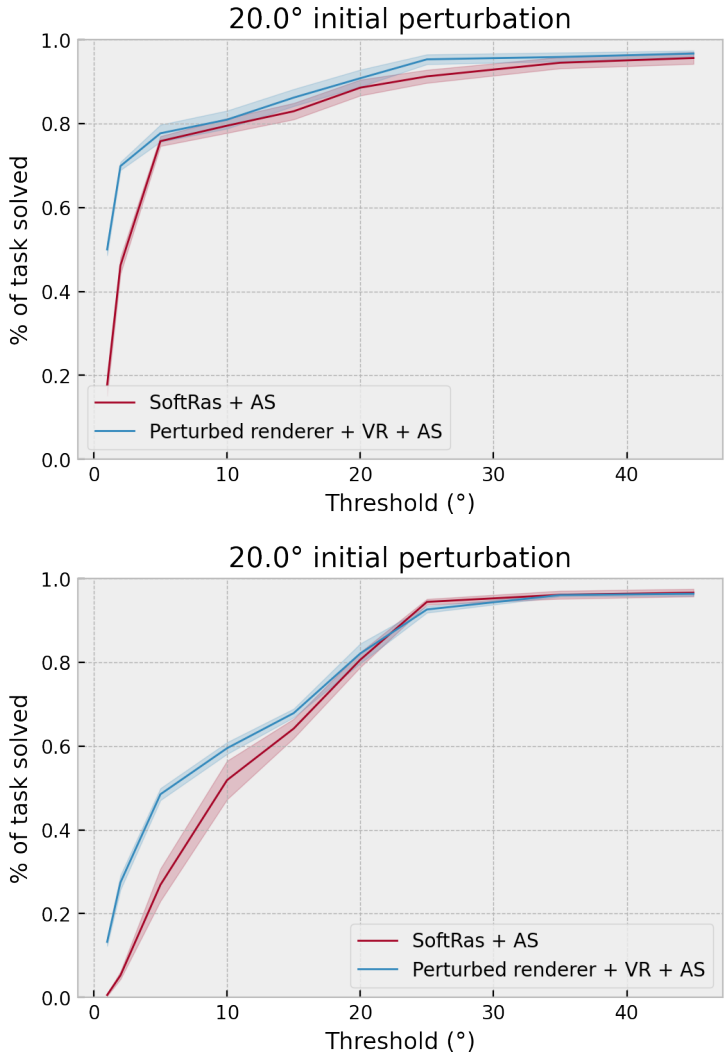

Figure 11: Pose optimization on various objects: a bag (**Top**) and a mailbox (**Bottom**)

In more details, $\mathcal{L}_{sil}$ is the negative IoU between silhouettes I and $\hat{I}$ which can be expressed as:

$$\mathcal{L}_{sil} = \mathbb{E}\left[1 - \frac{\|I \odot \hat{I}\|_1}{\|I + \hat{I} - I \odot \hat{I}\|_1}\right] \tag{75}$$

$\mathcal{L}_{RGB}$ is the ℓ_1 RGB loss between R and $\hat{R}$:

$$\mathcal{L}_{RGB} = \|R - \hat{R}\|_1 \tag{76}$$

$\mathcal{L}_{lap}$ is a Laplacian regularization term penalizing the relative change in position between a vertices v and its neighbours $\mathcal{N}(v)$ which can be written:

$$\mathcal{L}_{lap} = \sum_v \left(v - \frac{1}{|\mathcal{N}(v)|} \sum_{v' \in \mathcal{N}(v)} v'\right)^2 \tag{77}$$

Additional results We provide additional qualitative results for colorized mesh reconstruction (Fig. 13).

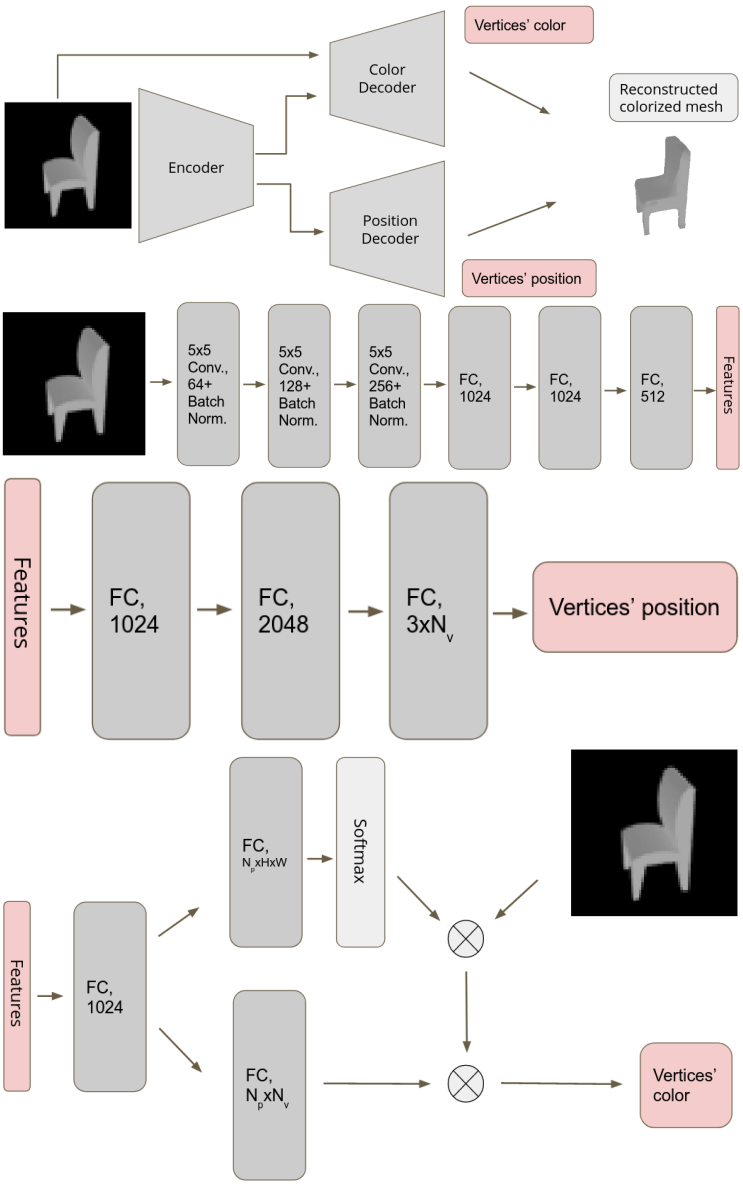

Figure 12: Learning architecture for the task of colorized 3D mesh reconstruction. **First row**: Overview of reconstruction network. **Second row**: Encoder network. **Third row**: Positional decoder. **Fourth row**: Color decoder.

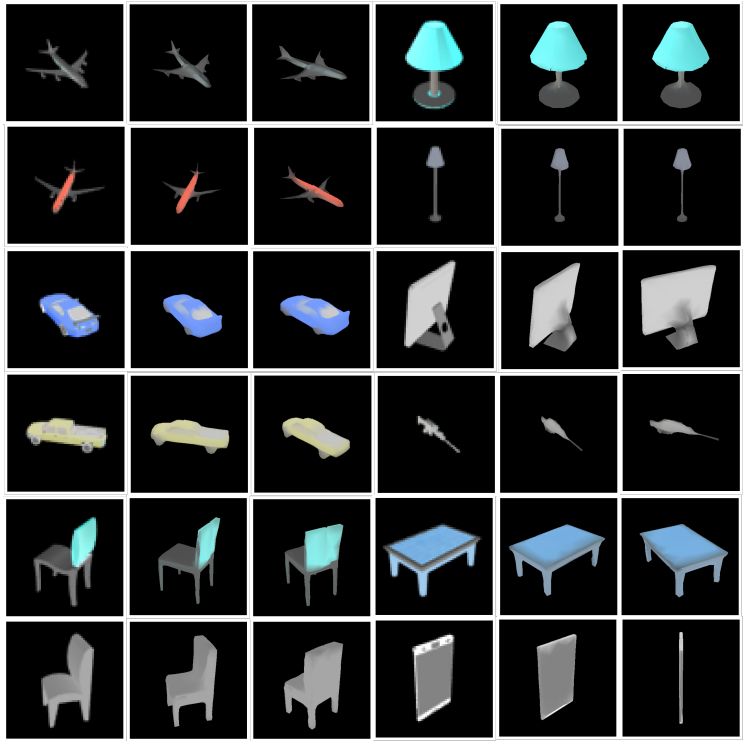

Figure 13: Results from unsupervised 3D mesh and color reconstruction. $\mathbf{1^{st}}$ **and** $\mathbf{4^{th}}$ **column:** Input image. **Other columns:** Reconstructed 3D mesh from different angles.