Recovering 3D Shapes from Ultra-Fast Motion-Blurred Images

Fei Yu[1], Shudan Guo[1], Shiqing Xin[1], Beibei Wang[2], Haisen Zhao[1†], Wenzheng Chen[3]
[1]Shandong University [2]Nanjing University [3]Peking University

Abstract

We consider the problem of 3D shape recovery from ultra-fast motion-blurred images. While 3D reconstruction from static images has been extensively studied, recovering geometry from extreme motion-blurred images remains challenging. Such scenarios frequently occur in both natural and industrial settings, such as fast-moving objects in sports (e.g.*, balls) or rotating machinery, where rapid motion distorts object appearance and makes traditional 3D reconstruction techniques like Multi-View Stereo (MVS) ineffective.*

In this paper, we propose a novel inverse rendering approach for shape recovery from ultra-fast motion-blurred images. While conventional rendering techniques typically synthesize blur by averaging across multiple frames, we identify a major computational bottleneck in the repeated computation of barycentric weights. To address this, we propose a fast barycentric coordinate solver, which significantly reduces computational overhead and achieves a speedup of up to ***4.57***$\times$*, enabling efficient and photorealistic simulation of high-speed motion. Crucially, our method is fully differentiable, allowing gradients to propagate from rendered images to the underlying 3D shape, thereby facilitating shape recovery through inverse rendering.*

We validate our approach on two representative motion types: rapid translation and rotation. Experimental results demonstrate that our method enables efficient and realistic modeling of ultra-fast moving objects in the forward simulation. Moreover, it successfully recovers 3D shapes from 2D imagery of objects undergoing extreme translational and rotational motion, advancing the boundaries of vision-based 3D reconstruction. Project page can be found at https://maxmilite.github.io/rec-from-ultrafast-blur/.

1. Introduction

Estimating the shape of an object from image collections is crucial for numerous applications, including film production, gaming, and AR/VR. As a long-standing goal in computer vision and graphics, extensive research has leveraged geometric and learning-based priors for object shape recovery [5, 9, 11, 14, 22]. However, most existing methods focus on static objects or those with low-speed motion [4, 23, 43], leaving shape recovery of high-speed objects largely underexplored.

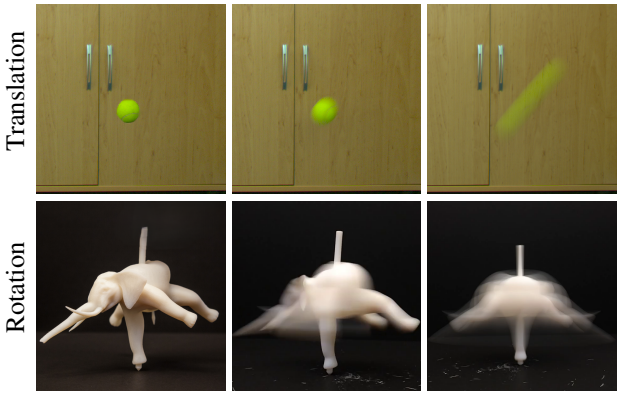


Figure 1. **Ultra-fast motion blur** is common in real-world scenarios. Top: A ball undergoing translational motion [32]. Bottom: A spinning top in rotation [1]. In this paper, our goal is to recover 3D shapes from the high-speed translational and rotational motion.

In this work, we investigate an extremely challenging question: Can we recover the shape of an object undergoing ultra-fast motion? Fast motion is prevalent in real-world scenarios, such as flying balls in sports, rotating machinery, or high-speed robotics. While reducing exposure time can mitigate blur, it often leads to extremely low signal-to-noise ratios in low-light conditions, making motion blur physically unavoidable in many practical scenarios. However, extreme motion blur severely distorts the object's appearance, often obscuring the underlying shape. As shown in Fig. 1, the object's shape is barely perceptible in the captured blurry images, making traditional multi-view geometry-based methods, such as Structure from Motion (SfM) [9, 11], ineffective. SfM-based techniques rely on sharp feature correspondences across views, but when motion blur obscures these features, shape recovery becomes highly challenging.

Alternatively, recovering 3D shape from 2D image collections can be formulated as an inverse problem, where the objective is to optimize the shape so that its render-

†Corresponding author.

ings match the observed images [22]. Leveraging this paradigm, we formulate shape recovery under ultra-fast motion as an inverse rendering problem, where both geometry and appearance are estimated by simulating the blurry process. Typically, motion blur can be approximated by rendering multiple static frames and averaging them [32–34, 39]. However, this method becomes computationally expensive for ultra-fast translational or rotational motions. As shown in Fig. 2, generating realistic motion blur under such conditions requires synthesizing and averaging over 50 individual static frames per blurry image, leading to excessive rendering costs and memory consumption.

We carefully analyze the computational bottleneck in motion blur synthesis. While a single barycentric computation is inexpensive, we identify that the repetitive calculation of these weights required for temporal integration becomes a primary source of inefficiency. This is because barycentric weights must be computed for every pixel with respect to all triangles, and the synthesis of motion blur further amplifies the computational cost by requiring these computations across all sampled frames, leading to a significant overhead. To address this issue, inspired by analytic motion approximation techniques [10], we propose a fast barycentric coordinate solver that significantly reduces computational complexity. By integrating this solver into our differentiable rasterization framework, our approach achieves significant speedup while preserving the accuracy of motion blur simulation. Furthermore, we reformulate the rendering process in a soft, fully differentiable manner, allowing gradients to propagate through motion-blurred images to the underlying 3D shapes.

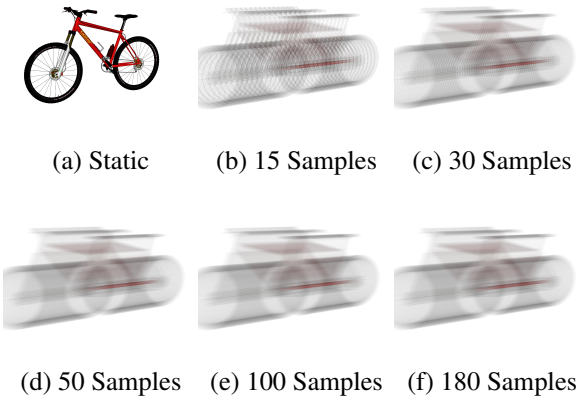
With its differentiable capabilities, our framework enables 3D shape recovery through an inverse rendering pipeline: Beginning with an initial 3D shape, we render motion-blurred images and compare them with the observed ground-truth (GT) images. The shape is then iteratively refined by minimizing the discrepancy between the rendered and GT images. This analysis-by-synthesis approach allows for shape recovery from multi-view blurred images, even under extreme translational and rotational motion.

We evaluate our method on a wide range of testing cases across various shapes and categories. Our method successfully recovers shapes from heavily blurred images caused by ultra-fast motion. Additionally, we demonstrate 3D shape recovery from real-world motion-blurred images, showcasing the effectiveness of our method in challenging real-world scenarios. Our work pushes the boundaries of 3D recovery from ultra-fast motion-blurred images.

2. Related Work

2.1. General Deblurring Methods

Motion blur arises when multiple scene contents are projected onto the same pixel due to motion during image capture [43]. This blur can originate from various sources, including camera motion, object motion, or long exposures in low-light conditions. Typically, motion blur is modeled as a convolution of a clean image with a blur kernel. Numerous methods have been developed to address this issue, leveraging various priors such as total variation (TV) and phase information [27], deep neural networks [12, 13, 26, 36, 44, 45], generative adversarial networks [19, 20, 42], and, more recently, diffusion models [2, 6, 7, 38, 39]. However, these methods primarily focus on low-speed motion, where the blur kernels remain relatively small. Furthermore, most approaches are confined to 2D image space, making them ineffective for handling more complex, non-linear motion patterns, such as rotations.



(a) Static (b) 15 Samples (c) 30 Samples

(d) 50 Samples (e) 100 Samples (f) 180 Samples

Figure 2. Motion blur is typically synthesized by rendering and averaging multiple frames. However, for extreme motion, a large number of frames are required to achieve realistic results. Here, we illustrate a bicycle undergoing extreme translation. Noticeable artifacts appear when using fewer samples, and at least 50 frames are needed to produce a realistic motion-blurred image.

2.2. Shape Recovery from Blurry Images or Videos

Our objective is to recover 3D shapes from blurry images of objects undergoing extremely fast motion [10, 37]. Objects exhibiting motion blur are often categorized as Fast Moving Objects (FMOs) [30]. Prior works have explored reconstructing both shape and motion from images or videos [17, 18, 31]. Rozumnyi et al. [33, 34] pioneered methods to recover 2D and 3D shapes along with motion from blurry images and videos. These approaches effectively leverage neural network-based learned priors, such as the DeFMO network [32], to predict per-timestamp static silhouettes of the object. Combined with differentiable rendering techniques [5, 22], they jointly estimate shape and motion via optimization, enabling robust solutions across a broad range of practical scenarios.

While impressive results are achieved, these methods also largely rely on accurately estimating object motion and static silhouettes, which might struggle with challenging, ultra-fast motion inputs. In this scenario, the resulting blur

introduces an unprecedented level of visual ambiguity, making the accurate recovery of static object silhouettes particularly challenging for [32]. In contrast, our method enables shape recovery under significantly more extreme high-speed motion conditions, demonstrating new capabilities in reconstructing objects undergoing ultra-fast motion.

2.3. Inverse Rendering

Shape recovery through inverse rendering has made rapid progress in recent years. A variety of 3D representations, including meshes, neural radiance fields (NeRF) [24], and Gaussian splatting [15], have been combined with differentiable rendering techniques such as mesh rendering [5, 14, 21, 22], volume rendering [8], and surface rendering [28] to jointly estimate shape, texture, lighting, and material directly from images [25, 40]. However, these methods are generally designed for clean, static images, and their applicability to motion-blurred scenes remains limited. In this paper, we propose a differentiable, rasterization-based renderer specifically designed to handle ultra-fast motion. Our approach extends inverse rendering to extreme motion-blurred conditions, making it a promising solution for high-speed shape recovery.

3. Method

We now describe our method. We first provide the preliminaries of traditional rasterization algorithms in Sec. 3.1 and analyze their computational bottleneck. We then present our solution: a fast barycentric coordinate solver in Sec. 3.2. With this new solver, we detail our differentiable motion-blur rendering algorithm in Sec. 3.3.

3.1. Preliminaries of Rasterization

Rasterization is a fundamental rendering technique that projects 3D triangle meshes onto a 2D image plane. Typically, it operates on a per-pixel basis by computing its barycentric coordinates with respect to each triangle. For a screen pixel $\boldsymbol{p}_i$ and a projected triangle face $\mathbf{F}_j$ with three vertices $\begin{bmatrix}\boldsymbol{v}_0 & \boldsymbol{v}_1 & \boldsymbol{v}_2\end{bmatrix}$, we denote the barycentric coordinates of $\boldsymbol{p}_i$ with respect to $\mathbf{F}_j$ as $\boldsymbol{w} = \begin{bmatrix}w_0 & w_1 & w_2\end{bmatrix}^T$, which satisfies the equation:

$$\boldsymbol{p}_i = w_0\boldsymbol{v}_0 + w_1\boldsymbol{v}_1 + w_2\boldsymbol{v}_2. \tag{1}$$

The vector $\boldsymbol{w}$ is then used to interpolate vertex attributes, such as colors or texture UV mappings.

Traditional differentiable rasterizers (*e.g.*, [5, 22]) compute $\boldsymbol{w}$ by solving a linear system. For example, if we define $\boldsymbol{p}_i = \begin{bmatrix}u & v & 1\end{bmatrix}^T$ and each vertex $\boldsymbol{v} = \begin{bmatrix}x & y & 1\end{bmatrix}^T$, the triangle $\mathbf{F}_j$ can be expressed as:

$$\mathbf{F}_j = \begin{bmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{bmatrix}. \tag{2}$$

The barycentric coordinates can then be obtained by solving:

$$\mathbf{F}_j\boldsymbol{w} = \boldsymbol{p}_i \Rightarrow \boldsymbol{w} = \mathbf{F}_j^{-1}\boldsymbol{p}_i. \tag{3}$$

If all w_0, w_1, w_2 fall within the range $[0, 1]$, the pixel $\boldsymbol{p}_i$ is covered by the triangle $\mathbf{F}_j$. Its final color is then determined using the Z-buffer algorithm, which selects the closest surface among all overlapping triangles.

Discussion We observe that barycentric coordinate computation constitutes a significant computational bottleneck in the context of differentiable motion-blur rasterization. Since barycentric weights for every pixel must be computed with respect to relevant triangles, the cost scales linearly with the number of temporal samples required to generate smooth, realistic blur effects. To overcome this limitation, we propose a fast barycentric coordinate solver that drastically reduces computational complexity and significantly accelerates rendering speed.

3.2. Fast Barycentric Coordinate Solver

Traditional rasterization methods [5, 22] synthesize motion blur by rendering multiple K frames and averaging them. However, by assuming that each triangle moves *linearly* in time, we propose a fast barycentric coordinate solver that avoids the heavy cost of repeated K barycentric computation.

Consider a 3D mesh object M moving linearly from time $T = 0$ to $T = 1$, where our goal is to render K frames at time steps $T = \frac{0}{K-1}, \frac{1}{K-1}, \frac{2}{K-1}, \ldots, \frac{K-1}{K-1}$. For a triangle $\mathbf{F}_j$ moving from $T = 0$ to $T = 1$, at a specific time $T = t$, its time-dependent vertex positions can be defined as:

$$\mathbf{F}_j(t) = \begin{bmatrix}\boldsymbol{v}_0(t) & \boldsymbol{v}_1(t) & \boldsymbol{v}_2(t)\end{bmatrix}. \tag{4}$$

Since we assume linear motion, each vertex position follows linear interpolation between the starting point $\boldsymbol{v}(0)$ and ending point $\boldsymbol{v}(1)$:

$$\boldsymbol{v}(t) = (1-t)\boldsymbol{v}(0) + t\boldsymbol{v}(1). \tag{5}$$

Thus, the matrix representation of $\mathbf{F}_j(t)$ can be represented as:

$$\mathbf{F}_j(t) = \begin{bmatrix} x_0(t) & x_1(t) & x_2(t) \\ y_0(t) & y_1(t) & y_2(t) \\ 1 & 1 & 1 \end{bmatrix}. \tag{6}$$

We then compute the barycentric weights $\boldsymbol{w}(t)$ as:

$$\boldsymbol{w}(t) = \mathbf{F}_j(t)^{-1}\boldsymbol{p}_i = \frac{\mathrm{adj}(\mathbf{F}_j(t))}{\det(\mathbf{F}_j(t))}\,\boldsymbol{p}_i, \tag{7}$$

where $\mathrm{adj}(\mathbf{F}_j(t))$ and $\det(\mathbf{F}_j(t))$ are the adjugate matrix and determinant of $\mathbf{F}_j(t)$.

Moreover, with the assumption of linear motion, they could be written as quadratic functions of t:

$$\boldsymbol{w}(t) = \frac{\boldsymbol{A}_1 t^2 + \boldsymbol{A}_2 t + \boldsymbol{A}_3}{a_1 t^2 + a_2 t + a_3}, \tag{8}$$

where $\boldsymbol{A}_1, \boldsymbol{A}_2, \boldsymbol{A}_3, a_1, a_2, a_3$ are precomputed 3×1 vectors and values that are independent of t and depend solely on $\mathbf{F}_j(0), \mathbf{F}_j(1)$ and $\boldsymbol{p}_i$. Consequently, for the total K frames, these coefficients $(\boldsymbol{A}_1, \boldsymbol{A}_2, \boldsymbol{A}_3, a_1, a_2, a_3)$ can be computed *only once*, and the barycentric coordinate $\boldsymbol{w}(t)$ can then be evaluated using Eq. (8). This allows for efficient barycentric computation without per-frame solving barycentric linear equations. The full derivation is provided in Section B.

3.3. Differentiable Rasterization

With our fast solver, we now describe our differentiable motion-blur rasterization, which is built on prior state-of-the-art differentiable rasterization works SoftRas [22] and DIB-R [5].

We first decompose the entire motion into several segments, assuming that inside each segment, all faces move linearly, which is a common assumption in previous motion-blur simulation work [10, 29, 37]. Note that our method can support complex motions, *e.g.*, a motion composed of rotation and translation, as long as it can be divided into linear motion segments (see Sec. L.1). For linear motions, such as translation, larger segments can be used, whereas for non-linear motions, like rotation, smaller segments are employed. In our experiments, we find that even for the extreme rotational motion, we can divide the full rotation into 12 segments and render smooth results.

Next, for each segment, we treat its start and end as keyframes and render intermediate frames with our fast solver. These rendered frames are averaged to generate the segment blurry image. Subsequently, all segment images are further averaged to produce the final blurry image.

Following DIB-R [5], we also separately process *foreground* pixels (covered by one or more faces) and *background* pixels (not covered by any faces) attributes.

Foreground Pixels For foreground pixels, we perform barycentric interpolation for each frame at time $T = t$ on the closest covering face using the Z-buffer:

$$I(t) = w_0(t)c_0 + w_1(t)c_1 + w_2(t)c_2, \tag{9}$$

where c represents vertex attributes (*e.g.*, vertex colors or texture UV coordinates).

Background Pixels If a pixel is not covered by any triangle, in differentiable rasterization it is assumed that it could be influenced by all triangles. Similarly, we extend the probability of a triangle $\mathbf{F}_j$ influencing a pixel $\boldsymbol{p}_i$ to a time-dependent version:

$$A_i^j(t) = \exp\left(-\frac{d\left(\boldsymbol{p}_i, \mathbf{F}_j(t)\right)}{\delta}\right), \tag{10}$$

where $A_i^j(t)$ is the time-dependent probability, δ is a hyperparameter [22], and $d\left(\boldsymbol{p}_i, \mathbf{F}_j(t)\right)$ is the squared Euclidean distance, which can be defined as

$$d\left(\boldsymbol{p}_i, \mathbf{F}_j(t)\right) = \min_{\boldsymbol{p}\in\mathbf{F}_j(t)} ||\boldsymbol{p}_i - \boldsymbol{p}||_2^2. \tag{11}$$

The core of the Euclidean distance calculation lies in finding $\boldsymbol{p} \in \mathbf{F}_j(t)$ that is closest to $\boldsymbol{p}_i$. By replacing $\boldsymbol{p}$ with another form $\boldsymbol{p} = \mathbf{F}_j(t)\hat{\boldsymbol{w}}$, where we constrain $\hat{\boldsymbol{w}} \in [0,1]^3$ to ensure $\boldsymbol{p} \in \mathbf{F}_j(t)$, finding the closest $\boldsymbol{p}$ is equivalent to finding $\hat{\boldsymbol{w}}^*$, which can be written as:

$$\hat{\boldsymbol{w}}^* = \arg\min_{\hat{\boldsymbol{w}}\in[0,1]^3} \|\mathbf{F}_j(t)\boldsymbol{w}(t) - \mathbf{F}_j(t)\hat{\boldsymbol{w}}\|_2^2, \tag{12}$$

where we also replace $\boldsymbol{p}_i = \mathbf{F}_j(t)\boldsymbol{w}(t)$.

However, we find that evaluating Equation (12) requires additional computational resources for computing $\mathbf{F}_j(t)$ across frames. To further accelerate the computation, we approximate $\mathbf{F}_j(t)$ with either $\mathbf{F}_j(0)$ or $\mathbf{F}_j(1)$, depending on whether t is closer to the start or the end:

$$\hat{\boldsymbol{w}}^* = \arg\min_{\hat{\boldsymbol{w}}\in[0,1]^3} \|\mathbf{F}_j(X)\boldsymbol{w}(t) - \mathbf{F}_j(X)\hat{\boldsymbol{w}}\|_2^2,\ X = \begin{cases} 0 & t \le 0.5 \\ 1 & t > 0.5 \end{cases} \tag{13}$$

Eq. (13) requires only the evaluation of $\mathbf{F}_j^{(0)}, \mathbf{F}_j^{(1)}$, which significantly reduces computational cost while introducing only minor approximation errors. The full derivation is provided in Sec. B. Finally, with the computed $\hat{\boldsymbol{w}}^*$, we obtain

$$d\left(\boldsymbol{p}_i, \mathbf{F}_j(t)\right) = \|\mathbf{F}_j(t)\boldsymbol{w}(t) - \mathbf{F}_j(t)\hat{\boldsymbol{w}}^*)\|_2^2. \tag{14}$$

We then combine the probabilistic influence of all triangle faces on a particular pixel as

$$A_i(t) = 1 - \prod_j \left(1 - A_i^j(t)\right). \tag{15}$$

Gradient Computation Equations (9) and (15) are fully differentiable [5, 22]. Therefore, our method supports backpropagation by propagating gradients through each time-dependent intermediate frame, and ultimately into the keyframes, ensuring efficient optimization in inverse rendering tasks.

4. Analysis

In this section, we evaluate the effectiveness of our method through extensive synthetic experiments. We implemented our method based on SoftRas [22] but split the pixels into foreground and background, following DIB-R [5]. We provide the implementation details in Section C.

We first analyze the ultra-fast motion blur synthesis effect in Sec. 4.1, including both forward rendering and backward gradients. Then, in Sec. 4.2, we present the computation speed, demonstrating significant acceleration over prior methods.

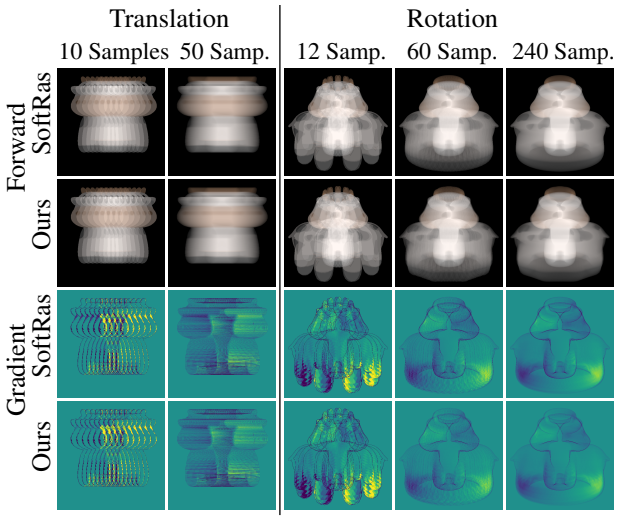


Figure 3. Forward rendering & backward gradient visualization for ultra-fast motion-blur synthesis. Our rendered images and gradients exhibit a high degree of similarity to those generated by SoftRas across various motion cases and sample numbers. Scene settings and render details are provided in Sections E and I.

4.1. Qualitative Validation

We first present the synthesis of ultra-fast motion blur effects, including both translational and rotational movements. We show forward rendering results and their backward gradients. As a reference, we also apply SoftRas [22] to render with the same settings (*e.g.*, the same number of sampled frames) and compute the corresponding gradients. The default hyperparameter values provided in SoftRas are used for this comparison.

The comparison results are shown in Fig. 3. Across all test cases, our rendered images and gradients exhibit a high degree of similarity to those generated by SoftRas, which demonstrates the effectiveness of our method in realistic and differentiable motion blur synthesis. In theory, under linear motion, our foreground pixel renderings should be identical to those of SoftRas, whereas our background pixel computation shows slight discrepancies, primarily due to our Euclidean distance approximation (as detailed in Eqs. (12) and (13)). However, we show that these minor discrepancies have negligible impact on gradient computation of our method (Fig. 3 Bottom).

4.2. Speed Comparison

Our method is significantly more efficient than traditional blur synthesis methods, *e.g.*, applying SoftRas to render and average multiple frames. To evaluate the running speed, we randomly select 50 models from ShapeNet [3], each containing an average of 5,536 faces. We apply random rotations to each model, render 128×128 front-view motion-blurred silhouette and color images, and measure the time required for both forward rendering and gradient computation in a single pass. We render objects undergoing linear translation with a varying number of samples. The evaluation considers the average time required for forward rendering and backward gradient computation across all models. All experiments are conducted on a single NVIDIA RTX 4090 GPU with 24GB of memory. More details are provided in Section I.

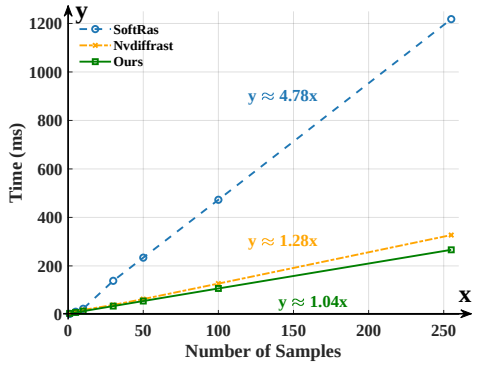


Figure 4. Forward + gradient computation timing results. The slope represents the average time of sampling once. The lower the better. Our method achieves speedups of up to 4.57× and 1.23× compared to SoftRas and Nvdiffrast, respectively.

The timing results are summarized in Fig. 4. Our method achieves speedups of up to 4.57× over SoftRas and 1.23× over Nvdiffrast. Regarding Nvdiffrast [21], it is a highly performant OpenGL-optimized rasterization library, whereas our implementation is built on the SoftRas implementation. Nevertheless, our method is still faster than Nvdiffrast, and incorporating our method into Nvdiffrast would likely bring further acceleration. Moreover, we observe that Nvdiffrast produces weak gradient signals, as its gradients are computed only near edges. This limitation can lead to slower convergence or even failure in extreme shape recovery tasks. In contrast, our method enables global gradient propagation across all triangle primitives, facilitating smoother optimization. Further discussion is provided in Sec. 6.4.

5. Shape Optimization from Blurred Images

With our differentiable rendering pipeline, we now present inverse rendering applications, *i.e.*, recovering 3D shapes from ultra-fast motion-blurred images. Thanks to our efficient framework, our method supports rendering more samples, resulting in smoother forward rendering effects and better backward gradients across the optimization process. We demonstrate the effectiveness and advantages of our method through two challenging tasks, where the goals are to recover the 3D shape of ultra-fast moving objects from two representative types of motion blur: multi-view translational and rotational blurred images.

Similar to other inverse rendering tasks, we assume known rendering parameters for each image, including its camera viewpoint and blur settings (translation or rotation speed).

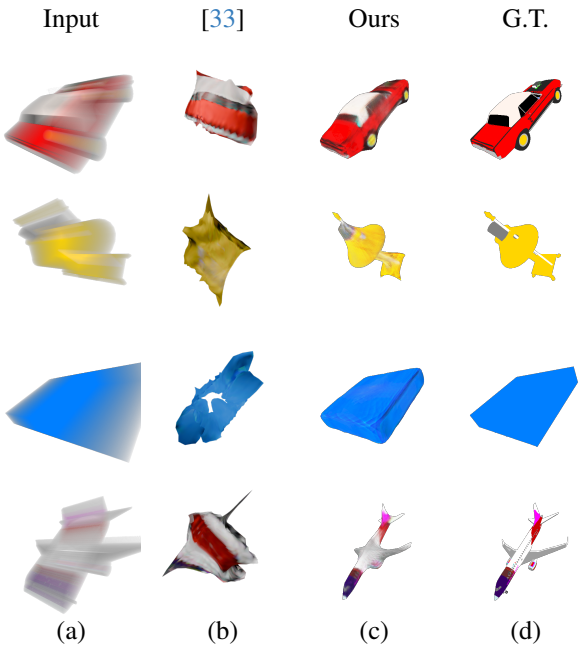


Figure 5. Qualitative results on geometry and color optimization. (a) One of blurred input images. (b) State of the Art [33] result. (c) Our optimization result. (d) Ground-truth object. Our method yields significant superior results than the state-of-the-art work. Note that the object trajectories are synthesized to test the solver's robustness to diverse motion vectors, rather than simulating realistic physical dynamics.

5.1. Translational Recovery

We begin with the task of optimizing a 3D shape undergoing linear translation. Given multi-view RGBA translational blurred images (consisting of RGB color I and transparency α) as input, our method optimizes a mesh M with vertex positions V and a texture map C such that the rendered images, $\hat{I}, \hat{\alpha} = R(V, C)$, match the input.

The optimization of V and C is performed by minimizing the image loss and regularization terms. We adopt the $\mathcal{L}_1$ loss for both color $I, \hat{I}$ and transparency $\alpha, \hat{\alpha}$, formulated as:

$$\mathcal{L}_{\text{img}} = \left\| I - \hat{I} \right\|_1 + \left\| \alpha - \hat{\alpha} \right\|_1 . \tag{16}$$

Similar to [5, 22], we incorporate a smoothness loss $\mathcal{L}_{\text{s}}$ and a Laplacian loss $\mathcal{L}_{\text{L}}$ to regularize the deformation of V (details provided in Section G). The final loss function is defined as:

$$\mathcal{L} = \mathcal{L}_{\text{img}} + \lambda_{\text{s}}\mathcal{L}_{\text{s}} + \lambda_{\text{L}}\mathcal{L}_{\text{L}}. \tag{17}$$

5.2. Rotational Recovery

We further apply our method to a more challenging task: reconstructing fast-rotating objects from motion-blurred images. Similarly, given multi-view images I as input, our method reconstructs a 3D mesh M corresponding to the observed object.

Method	Translation		Rotation
	3D IoU ↑	Static PSNR ↑	Blurred PSNR ↑
[33]	0.152	11.63	13.58
Ours	0.679	19.20	31.89

Table 1. Quantitative comparison for shape optimization from blurred images. We compare the best performance between our method and the state-of-the-art [33]. Our method achieves significantly superior performance compared to [33].

Due to the highly non-convex nature of the rotation optimization problem, we observe that directly optimizing mesh vertices rarely yields well-shaped objects (illustrated in Sec. F). To mitigate this issue, we optimize a Signed Distance Function (SDF) representation instead. We construct an SDF field $\mathcal{S}$ following [41], and extract a mesh M using FlexiCubes [35] in a differentiable manner. Our differentiable renderer R is then employed to generate corresponding rotation images $\hat{I}$, which are subsequently used to optimize $\mathcal{S}$ via loss functions. Given the complexity of the SDF representation, we render grayscale images in this setting and focus on shape recovery.

We retain the same $\mathcal{L}_{\text{img}}$ loss (from Eq. (17)) for image consistency. For SDF regularization, we utilize the loss terms $\mathcal{L}_{\text{crit}}$ and $\mathcal{L}_{\text{reg}}$ from [41] and [35], respectively (details in Section G). The final loss function is defined as:

$$\mathcal{L} = \mathcal{L}_{\text{img}} + \lambda_{\text{crit}}\mathcal{L}_{\text{crit}} + \lambda_{\text{reg}}\mathcal{L}_{\text{reg}}. \tag{18}$$

6. Experiments

In this section, we present qualitative and quantitative experiments to evaluate the effectiveness and efficiency of our method for shape recovery from ultra-fast motion-blurred images. In Secs. 6.1 and 6.2, we present translational and rotational blurred shape recovery, respectively. Furthermore, we validate our method's practical applicability through real-world motion blur data in Sec. 6.3. Finally, we conduct an ablation study in Sec. 6.4, to demonstrate the benefits of our method compared to the widely adopted codebase SoftRas [22] and Nvdiffrast [21] under extreme motion blur conditions. All hyperparameter settings and more results are provided in Sections I, J and L.

6.1. Translational Recovery

In this experiment, we present our method's 3D shape reconstruction performance for objects undergoing translational motion, specifically benchmarking against the state-of-the-art [33]. We perform optimization on 25 selected shapes from ShapeNet, and evaluate both the geometry and color recovery. For geometry quantitative evaluation, we voxelize the predicted and ground truth meshes into 32^3 volumes, and compute the 3D IoU. For color quantitative evaluation, we compare the PSNR of multi-view static novel-view-synthesis (NVS) of the objects. Quantitative evalua-

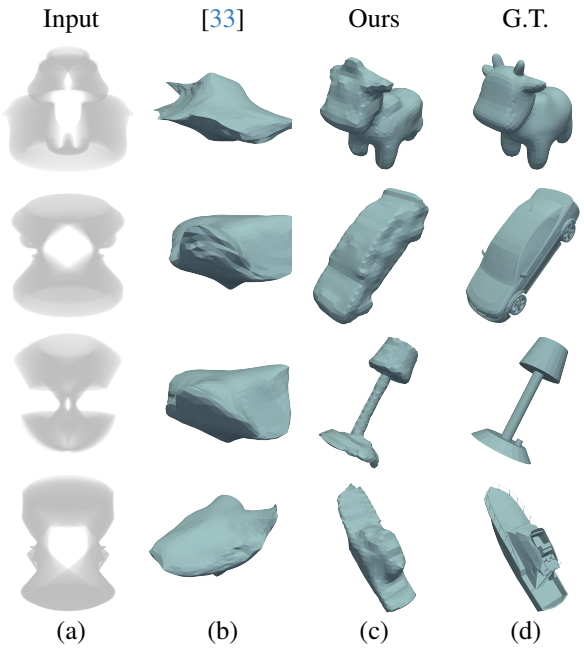


Figure 6. Qualitative results for optimization on rotating objects. (a) One of blurred input images. (b) State of the Art [33] result. (c) Our optimization result. (d) Ground-truth object. Our method also yields a significant superior result than the state-of-the-art work.

tion results are presented in Tab. 1, while qualitative results are illustrated in Fig. 5.

We assess our method's fundamental capability in 3D shape recovery from highly motion-blurred images by benchmarking against the state-of-the-art work [33]. As presented in Tab. 1 and Fig. 5, our method demonstrates a significant advantage in translational recovery. Notably, [33]'s reliance on the learning prior [32] to predict static silhouettes fundamentally limits its performance in the challenging blurry input. In contrast, our method successfully recovers meaningful 3D shape and appearance. More details and analysis are provided in Section K.1.

6.2. Rotational Recovery

In this experiment, we evaluate shape recovery for objects undergoing ultra-fast rotational motion, benchmarking against the state-of-the-art [33]. Here, we observe that multiple feasible 3D shape solutions may correspond to the same blurred image; a detailed analysis is provided in Section H. Consequently, traditional 3D evaluation metrics (*e.g.*, 3D IoU, Chamfer Distance) are not suitable. Therefore, we assess the similarity between the rotational-blurred images rendered from the reconstructed objects and the ground truth, quantified using PSNR.

Similar to Sec. 6.1, for rotational motion, a similar trend of superior performance is observed for our method compared to [33]. As shown in Tab. 1 and Fig. 6, our method achieves a significantly superior performance and succeeds in a high-quality 3D reconstructions even under extreme rotational scenarios.

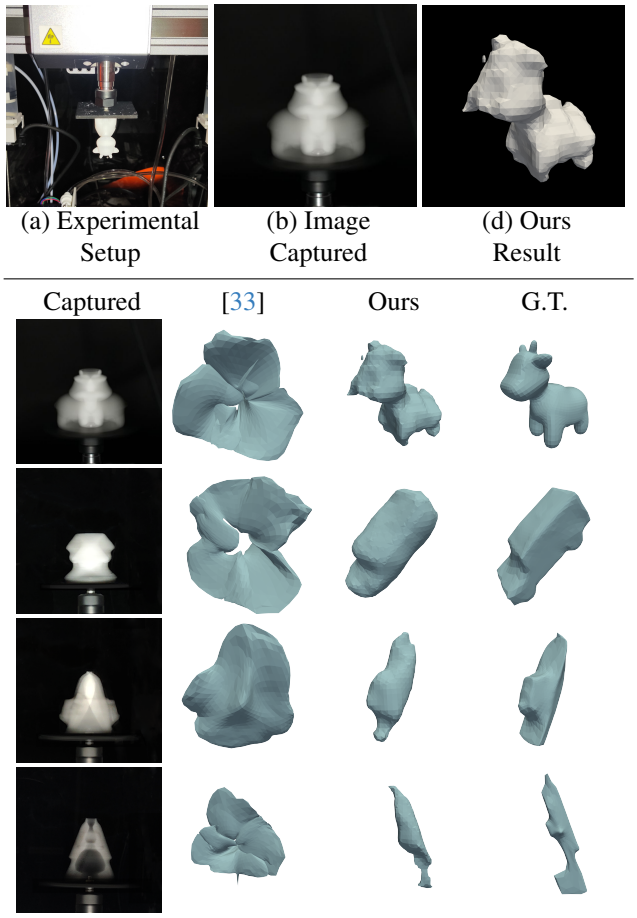


Figure 7. Real-world experiment. (a) Our experimental setup. (b) The original image captured. (c) Ours optimization result. The remaining rows: More real-world examples. Quantitatively, our method achieves a superior blurred PSNR score of 24.52 dB compared to 12.51 dB for [33]. Our method is capable of reconstructing 3D objects from real-world motion-blurred images.

6.3. Real-World Results

Next, we evaluate our method on real-world images. We capture a front view of 3D-printed rotating objects at 100 Hz with a camera exposure time of $1/100$ s. Since the data was captured in a controlled studio environment with a black background, we extract the object alpha masks based on pixel intensity thresholds to serve as supervision signals. After preprocessing, including cropping and brightness correction, we perform rotational shape optimization using the same settings as described in Sec. 5.2.

Results and evaluations are presented in Fig. 7. The 3D print technique allows us to establish ground truth for evaluation. As summarized, our method achieves a superior blurred PSNR score of 24.52 dB compared to 12.51 dB for [33], demonstrating a significant improvement. Qualitatively, due to the imperfect pose and noise introduced in real data, the recovered shape exhibits slight artifacts compared

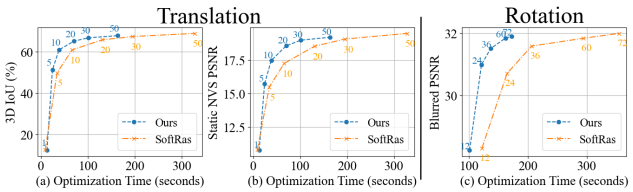


Figure 8. Optimization results for objects undergoing translation and rotation. We draw optimization time v.s. performance curves for our method and SoftRas, where each point indicate different number of samples.

to the synthetic recovery results. Nevertheless, our method successfully recovers reasonable shapes, demonstrating its capability to handle real-world data effectively.

6.4. Ablation Study

Finally, we conduct an ablation study to validate our key design choices. Specifically, we analyze the efficiency improvements of our method compared to the codebase SoftRas, and assess the robustness and gradient quality of our method against high-performance Nvdiffrast under the extreme motion blur scenarios.

Comparison with SoftRas Our method is built upon the SoftRas framework. Therefore, we compare our method against SoftRas in terms of optimization time and reconstruction quality. As shown in Fig. 8, several interesting points can be observed. First, increasing the number of samples improves performance but also increases optimization time. This is reasonable as more samples result in better blur simulation, yielding better shape recovery performance and longer time. Second, our method exhibits strong efficiency over SoftRas. Given the same number of samples, our method achieves significantly faster optimization time. Conversely, for the same optimization time, our method yields superior reconstruction quality.

Comparison with Nvdiffrast Next, we present a comparison of our method's gradient quality and robustness against Nvdiffrast. We first quantitatively assess the convergence behaviors of both methods as a function of the number of iterations. As illustrated in Fig. 9, our method demonstrates a significantly faster convergence rate. This is attributable to the stronger and more stable gradients generated by our approach, in contrast to the comparatively weaker gradients produced by Nvdiffrast.

Beyond convergence speed, we observe that Nvdiffrast [21] frequently encounters catastrophic failures, leading to the inability to reconstruct valid meshes. For example, in our experiments on rotational recovery, Nvdiffrast failed to successfully complete the process (manifested as program crashes) in any of the 10 repeated attempts within 8 out of 25 test data cases, which even precluded a quantitative comparison with our method. Furthermore, even in cases where Nvdiffrast manages to complete the reconstruction, its output quality often exhibits lower fidelity compared to ours,

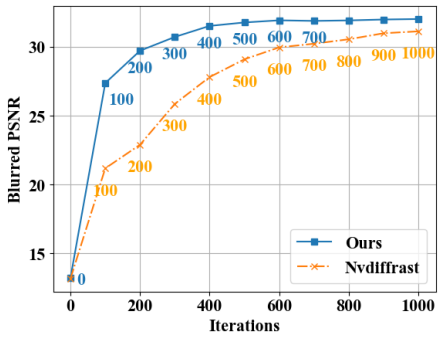
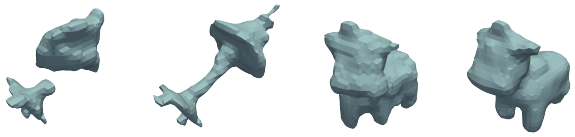


Figure 9. Comparison of convergence rates between our method and Nvdiffrast w.r.t. number of iterations. Labels denote numbers of iterations. The convergence rate of Nvdiffrast is significantly slower than that of our method, which we attribute to its weaker pixel-wise gradients.



(a) Nvdiffrast (b) Ours (c) Nvdiffrast (d) Ours

Figure 10. Failure cases of Nvdiffrast. (a, c) Nvdiffrast optimization results. (b, d) Ours optimization results. In this task, our method can successfully recover a well-shaped object in most scenarios, whereas Nvdiffrast frequently fails, or producing distorted and uneven objects.

as illustrated in Fig. 10. In stark contrast, our method consistently achieves successful, well-shaped, and high-fidelity reconstructions. More analysis is provided in Section K.2.

7. Discussion

Limitations Our method currently relies on known camera poses and motion parameters. Additionally, our physical formation model assumes linear motion segments and a linear, noise-free photometric response. While effective, these assumptions may deviate from in-the-wild scenarios characterized by complex non-linear motion, camera response functions (tone mapping), or sensor noise. We provide a comprehensive discussion on these limitations and future works in Section M.

Conclusion In this paper, we propose a novel inverse rendering approach for 3D shape recovery from ultra-fast motion-blurred images. Our fast barycentric coordinate solver accelerates rendering while preserving accuracy, enabling efficient and fully differentiable shape reconstruction. Experimental results validate the effectiveness of our method on both synthetic and real-world data, advancing 3D reconstruction under ultra-fast motion blur.

8. Acknowledgements

This work is supported in part by grants from National Key Research and Development Program of China (Grant No. 2024YFB3309500), National Natural Science Foundation of China (Grant No. U23A20312, 62472257).

References

[1] Moritz Bächer, Emily Whiting, Bernd Bickel, and Olga Sorkine-Hornung. Spin-it: Optimizing moment of inertia for spinnable objects. *ACM Transactions on Graphics (TOG)*, 33(4):1–10, 2014. 1

[2] Weimin Bai, Siyi Chen, Wenzheng Chen, and He Sun. Blind inversion using latent diffusion priors. *arXiv preprint arXiv:2407.01027*, 2024. 2

[3] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 5

[4] Wenbo Chen and Ligang Liu. Deblur-gs: 3d gaussian splatting from camera motion blurred images. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 7 (1):1–15, 2024. 1

[5] Wenzheng Chen, Huan Ling, Jun Gao, Edward Smith, Jaako Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. In *NeurIPS*, 2019. 1, 2, 3, 4, 6

[6] Hyungjin Chung, Jeongsol Kim, Michael T Mccann, Marc L Klasky, and Jong Chul Ye. Diffusion posterior sampling for general noisy inverse problems. *arXiv preprint arXiv:2209.14687*, 2022. 2

[7] Hyungjin Chung, Jeongsol Kim, Sehui Kim, and Jong Chul Ye. Parallel diffusion models of operator and image for blind inverse problems. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6059–6069, 2023. 2

[8] Robert A Drebin, Loren Carpenter, and Pat Hanrahan. Volume rendering. *ACM Siggraph Computer Graphics*, 22(4): 65–74, 1988. 3

[9] Yasutaka Furukawa, Carlos Hernández, et al. Multi-view stereo: A tutorial. *Foundations and Trends® in Computer Graphics and Vision*, 9(1-2):1–148, 2015. 1

[10] Carl Johan Gribel, Michael C Doggett, and Tomas Akenine-Möller. Analytical motion blur rasterization with compression. *High Performance Graphics*, 10, 2010. 2, 4

[11] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003. 1

[12] Meiguang Jin, Givi Meishvili, and Paolo Favaro. Learning to extract a video sequence from a single motion-blurred image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6334–6342, 2018. 2

[13] Meiguang Jin, Zhe Hu, and Paolo Favaro. Learning to extract flawless slow motion from blurry videos. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8112–8121, 2019. 2

[14] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3907–3916, 2018. 1, 3

[15] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (ToG)*, 42(4):1–14, 2023. 3

[16] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5

[17] Jan Kotera, Denys Rozumnyi, Filip Sroubek, and Jiri Matas. Intra-frame object tracking by deblatting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019. 2

[18] Jan Kotera, Jiří Matas, and Filip Šroubek. Restoration of fast moving objects. *IEEE Transactions on Image Processing*, 29:8577–8589, 2020. 2

[19] Orest Kupyn, Volodymyr Budzan, Mykola Mykhailych, Dmytro Mishkin, and Jiří Matas. Deblurgan: Blind motion deblurring using conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8183–8192, 2018. 2

[20] Orest Kupyn, Tetiana Martyniuk, Junru Wu, and Zhangyang Wang. Deblurgan-v2: Deblurring (orders-of-magnitude) faster and better. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019. 2

[21] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics (ToG)*, 39(6):1–14, 2020. 3, 5, 6, 8

[22] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. *The IEEE International Conference on Computer Vision (ICCV)*, 2019. 1, 2, 3, 4, 5, 6

[23] Yiren Lu, Yunlai Zhou, Disheng Liu, Tuo Liang, and Yu Yin. Bard-gs: Blur-aware reconstruction of dynamic scenes via gaussian splatting. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 16532–16542, 2025. 1

[24] B Mildenhall, PP Srinivasan, M Tancik, JT Barron, R Ramamoorthi, and R Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, 2020. 3

[25] Jacob Munkberg, Jon Hasselgren, Tianchang Shen, Jun Gao, Wenzheng Chen, Alex Evans, Thomas Müller, and Sanja Fidler. Extracting Triangular 3D Models, Materials, and Lighting From Images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8280–8290, 2022. 3

[26] Jinshan Pan, Haoran Bai, and Jinhui Tang. Cascaded deep video deblurring using temporal sharpness prior. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3043–3051, 2020. 2

[27] Liyuan Pan, Richard Hartley, Miaomiao Liu, and Yuchao Dai. Phase-only image based kernel estimation for single image blind deblurring. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6034–6043, 2019. 2

[28] Hanspeter Pfister, Matthias Zwicker, Jeroen Van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 335–342, 2000. 3

[29] Mads JL Rønnow, Ulf Assarsson, and Marco Fratarcangeli. Fast analytical motion blur with transparency. *Computers & Graphics*, 95:36–46, 2021. 4

[30] Denys Rozumnyi, Jan Kotera, Filip Sroubek, Lukas Novotny, and Jiri Matas. The world of fast moving objects. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5203–5211, 2017. 2

[31] Denys Rozumnyi, Jan Kotera, Filip Sroubek, and Jiri Matas. Sub-frame appearance and 6d pose estimation of fast moving objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6778–6786, 2020. 2

[32] Denys Rozumnyi, Martin R Oswald, Vittorio Ferrari, Jiri Matas, and Marc Pollefeys. Defmo: Deblurring and shape recovery of fast moving objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3456–3465, 2021. 1, 2, 3, 7, 5, 6

[33] Denys Rozumnyi, Martin R Oswald, Vittorio Ferrari, and Marc Pollefeys. Shape from blur: Recovering textured 3d shape and motion of fast moving objects. *Advances in Neural Information Processing Systems*, 34:29972–29983, 2021. 2, 6, 7, 5, 8

[34] Denys Rozumnyi, Martin R Oswald, Vittorio Ferrari, and Marc Pollefeys. Motion-from-blur: 3d shape and motion estimation of motion-blurred objects in videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15990–15999, 2022. 2

[35] Tianchang Shen, Jacob Munkberg, Jon Hasselgren, Kangxue Yin, Zian Wang, Wenzheng Chen, Zan Gojcic, Sanja Fidler, Nicholas Sharp, and Jun Gao. Flexible isosurface extraction for gradient-based mesh optimization. *ACM Trans. Graph.*, 42(4), 2023. 6, 3, 4, 5

[36] Wang Shen, Wenbo Bao, Guangtao Zhai, Li Chen, Xiongkuo Min, and Zhiyong Gao. Blurry video frame interpolation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5114–5123, 2020. 2

[37] Konstantin Shkurko, Cem Yuksel, Daniel Kopta, Ian Mallett, and Erik Brunvand. Time interval ray tracing for motion blur. *IEEE transactions on visualization and computer graphics*, 24(12):3225–3238, 2017. 2, 4

[38] Bowen Song, Soo Min Kwon, Zecheng Zhang, Xinyu Hu, Qing Qu, and Liyue Shen. Solving inverse problems with latent diffusion models via hard data consistency. *arXiv preprint arXiv:2307.08123*, 2023. 2

[39] Radim Spetlik, Denys Rozumnyi, and Jiří Matas. Single-image deblurring, trajectory and shape recovery of fast moving objects with denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 6857–6866, 2024. 2

[40] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *NeurIPS*, 2021. 3

[41] Zixiong Wang, Yunxiao Zhang, Rui Xu, Fan Zhang, Peng-Shuai Wang, Shuangmin Chen, Shiqing Xin, Wenping Wang, and Changhe Tu. Neural-singular-hessian: Implicit neural representation of unoriented point clouds by enforcing singular hessian. *ACM Transactions on Graphics (TOG)*, 42 (6):1–14, 2023. 6, 4

[42] Kaihao Zhang, Wenhan Luo, Yiran Zhong, Lin Ma, Bjorn Stenger, Wei Liu, and Hongdong Li. Deblurring by realistic blurring. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2737–2746, 2020. 2

[43] Kaihao Zhang, Wenqi Ren, Wenhan Luo, Wei-Sheng Lai, Björn Stenger, Ming-Hsuan Yang, and Hongdong Li. Deep image deblurring: A survey. *International Journal of Computer Vision*, 130(9):2103–2130, 2022. 1, 2

[44] Zhihang Zhong, Ye Gao, Yinqiang Zheng, and Bo Zheng. Efficient spatio-temporal recurrent neural network for video deblurring. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VI 16*, pages 191–207. Springer, 2020. 2

[45] Shangchen Zhou, Jiawei Zhang, Jinshan Pan, Haozhe Xie, Wangmeng Zuo, and Jimmy Ren. Spatio-temporal filter adaptive network for video deblurring. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2482–2491, 2019. 2

Recovering 3D Shapes from Ultra-Fast Motion-Blurred Images

Supplementary Material

A. Appendix Overview

In this appendix, we provide a comprehensive explanation of the technical details and additional experimental results of our work.

We start with the derivation of our method in Section B, followed by implementation details in Section C. Section D discusses segmentation analysis, and Section E covers visualization details. Scene settings are outlined in Section I. We then address limitations with a failure case of rotational optimization in Section F and detail all loss terms in Section G. Section H analyzes the suitability of 3D losses for evaluation. Finally, Section J provides hyperparameter settings, Sec. K provides further analysis of baselines, and Section L presents more results.

B. Derivation of Our Method

In this section, we show the derivation of our method.

Barycentric Coordinate Solver We detail the derivation of $\boldsymbol{A}_1$, $\boldsymbol{A}_2$, $\boldsymbol{A}_3$, a_1, a_2, a_3.

First we introduce the definition of $\mathrm{adj}(\mathbf{F}_j(t))$ and $\det(\mathbf{F}_j(t))$. For a 3×3 matrix $\mathbf{A}$, given by:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, \tag{A1}$$

its determinant $\det(\mathbf{A})$ is computed as:

$$\det(\mathbf{A}) = a_{11}\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12}\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13}\begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} \tag{A2}$$

where each 2×2 determinant (called a minor) is computed as:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc. \tag{A3}$$

For the matrix $\mathbf{A}$, its adjugate matrix $\mathrm{adj}(\mathbf{A})$ can be defined by:

$$\mathrm{adj}(\mathbf{A}) = \begin{bmatrix} C_{11} & C_{21} & C_{31} \\ C_{12} & C_{22} & C_{32} \\ C_{13} & C_{23} & C_{33} \end{bmatrix}, \tag{A4}$$

where C_{ij} are defined as:

$$C_{ij} = (-1)^{i+j} M_{ij}$$

and M_{ij} is the determinant of the minor matrix obtained by deleting the i-th row and j-th column of $\mathbf{A}$.

If $\mathbf{A}$ is invertible (*i.e.*, $\det(\mathbf{A}) \neq 0$), the inverse of $\mathbf{A}$ can be expressed in terms of its adjugate matrix:

$$\mathbf{A}^{-1} = \frac{\mathrm{adj}(\mathbf{A})}{\det(\mathbf{A})}. \tag{A5}$$

Now we consider derivation on a triangle. For a triangle matrix $\mathbf{F}_j(t)$ represented as:

$$\mathbf{F}_j(t) = \begin{bmatrix} x_0(t) & x_1(t) & x_2(t) \\ y_0(t) & y_1(t) & y_2(t) \\ 1 & 1 & 1 \end{bmatrix}, \tag{A6}$$

where the vertex position $\boldsymbol{v}(t)$ can be defined by:

$$\boldsymbol{v}(t) = (1-t)\boldsymbol{v}(0) + t\boldsymbol{v}(1), \tag{A7}$$

its determinant $\det(\mathbf{F}_j(t))$ is

$$\begin{aligned} \det(\mathbf{F}_j(t)) = {} & x_0(t)\big(y_1(t) - y_2(t)\big) \\ & - x_1(t)\big(y_0(t) - y_2(t)\big) \\ & + x_2(t)\big(y_0(t) - y_1(t)\big), \end{aligned} \tag{A8}$$

and the adjugate matrix $\mathrm{adj}(\mathbf{A})$ is

$$\mathrm{adj}(\mathbf{F}_j(t)) = \begin{bmatrix} y_1(t) - y_2(t) & x_2(t) - x_1(t) & x_1(t)y_2(t) - x_2(t)y_1(t) \\ y_2(t) - y_0(t) & x_0(t) - x_2(t) & x_2(t)y_0(t) - x_0(t)y_2(t) \\ y_0(t) - y_1(t) & x_1(t) - x_0(t) & x_0(t)y_1(t) - x_1(t)y_0(t) \end{bmatrix}. \tag{A9}$$

Given a pixel $\boldsymbol{p}_i = \begin{bmatrix} u & v & 1 \end{bmatrix}^T$, we have

$$\begin{aligned} \boldsymbol{w}(t) = \mathbf{F}_j(t)^{-1}\boldsymbol{p}_i &= \frac{\mathrm{adj}(\mathbf{F}_j(t)) \times \boldsymbol{p}_i}{\det(\mathbf{F}_j(t))} \\ &= \frac{\boldsymbol{A}_1 t^2 + \boldsymbol{A}_2 t + \boldsymbol{A}_3}{a_1 t^2 + a_2 t + a_3} \end{aligned} \tag{A10}$$

By Eqs. (A6) to (A10), we can represent $\boldsymbol{A}_1$, $\boldsymbol{A}_2$, $\boldsymbol{A}_3$, a_1, a_2, a_3 using $x_i(0/1), y_i(0/1), u, v$:

$$\boldsymbol{A}_1 = \begin{bmatrix} -(x_2(0) - x_2(1))(y_1(0) - y_1(1)) \\ +(x_1(0) - x_1(1))(y_2(0) - y_2(1)) \\ \\ ((x_2(0) - x_2(1))(y_0(0) - y_0(1)) \\ -(x_0(0) - x_0(1))(y_2(0) - y_2(1))) \\ \\ (-((x_1(0) - x_1(1))(y_0(0) - y_0(1))) \\ +(x_0(0) - x_0(1))(y_1(0) - y_1(1))) \end{bmatrix}, \tag{A11}$$

$$\boldsymbol{A}_2 = \begin{bmatrix} u(-y_1(0)+y_2(0)+y_1(1)-y_2(1)) \\ +v(x_1(0)-x_2(0)-x_1(1)+x_2(1)) \\ +(y_2(0)x_1(1)-y_1(0)x_2(1)+ \\ x_2(0)(2y_1(0)-y_1(1))+ \\ x_1(0)(-2y_2(0)+y_2(1))) \\ \\ u(y_0(0)-y_2(0)-y_0(1)+y_2(1)) \\ +v(-x_0(0)+x_2(0)+x_0(1)-x_2(1)) \\ +(-(y_2(0)x_0(1))+y_0(0)x_2(1)+ \\ x_2(0)(-2y_0(0)+y_0(1))+ \\ x_0(0)(2y_2(0)-y_2(1))) \\ \\ u(-y_0(0)+y_1(0)+y_0(1)-y_1(1)) \\ +v(x_0(0)-x_1(0)-x_0(1)+x_1(1)) \\ +(y_1(0)x_0(1)-y_0(0)x_1(1)+ \\ x_1(0)(2y_0(0)-y_0(1))+ \\ x_0(0)(-2y_1(0)+y_1(1))) \end{bmatrix}, \tag{A12}$$

$$\boldsymbol{A}_3 = \begin{bmatrix} u(y_1(0)-y_2(0))+v(-x_1(0)+x_2(0)) \\ +(-(x_2(0)y_1(0))+x_1(0)y_2(0)) \\ \\ u(-y_0(0)+y_2(0))+v(x_0(0)-x_2(0)) \\ +(x_2(0)y_0(0)-x_0(0)y_2(0)) \\ \\ u(y_0(0)-y_1(0))+v(-x_0(0)+x_1(0)) \\ +(-(x_1(0)y_0(0))+x_0(0)y_1(0)) \end{bmatrix}, \tag{A13}$$

$$\begin{gathered} a_1 = \\ -(y_1(0)x_0(1))+y_2(0)x_0(1) \\ -y_2(0)x_1(1)+y_0(0)(x_1(1)-x_2(1)) \\ +y_1(0)x_2(1)-x_1(1)y_0(1) \\ +x_2(1)y_0(1)+x_0(1)y_1(1)-x_2(1)y_1(1) \\ +x_2(0)(y_0(0)-y_1(0)-y_0(1)+y_1(1)) \\ +x_1(0)(-y_0(0)+y_2(0)+y_0(1)-y_2(1)) \\ -x_0(1)y_2(1)+x_1(1)y_2(1) \\ +x_0(0)(y_1(0)-y_2(0)-y_1(1)+y_2(1)), \end{gathered} \tag{A14}$$

$$\begin{gathered} a_2 = \\ y_1(0)x_0(1)-y_2(0)x_0(1) \\ +y_2(0)x_1(1)-y_1(0)x_2(1) \\ +y_0(0)(-x_1(1)+x_2(1)) \\ +x_2(0)(-2y_0(0)+2y_1(0)+y_0(1)-y_1(1)) \\ +x_0(0)(-2y_1(0)+2y_2(0)+y_1(1)-y_2(1)) \\ +x_1(0)(2y_0(0)-2y_2(0)-y_0(1)+y_2(1)), \end{gathered} \tag{A15}$$

$$\begin{gathered} a_3 = \\ x_2(0)(y_0(0)-y_1(0)) \\ +x_0(0)(y_1(0)-y_2(0)) \\ +x_1(0)(-y_0(0)+y_2(0)). \end{gathered} \tag{A16}$$

Euclidean Distance Approximation In this section, we detail the derivation of $\hat{w}^*$.

Given a triangle $\mathbf{F} = \begin{bmatrix} \boldsymbol{v_0} & \boldsymbol{v_1} & \boldsymbol{v_2} \end{bmatrix} = \begin{bmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{bmatrix}$ and pixel barycentric coordinates $\boldsymbol{w} = \begin{bmatrix} w_0 & w_1 & w_2 \end{bmatrix}$, we consider finding $\hat{w}^* = \begin{bmatrix} w_0^* & w_1^* & w_2^* \end{bmatrix}$ such that

$$\hat{\boldsymbol{w}}^* = \underset{\hat{\boldsymbol{w}} \in [0,1]^3}{\arg\min} ||\mathbf{F}\boldsymbol{w} - \mathbf{F}\hat{\boldsymbol{w}}||_2^2 . \tag{A17}$$

If the pixel is inside the triangle, it's obvious that $\hat{\boldsymbol{w}}^* = \boldsymbol{w}$, so we only consider the scenario where the pixel is outside the triangle.

First we calculate the pixel position $\boldsymbol{p} = \begin{bmatrix} u & v & 1 \end{bmatrix}^T = \mathbf{F}\boldsymbol{w}$. If $\boldsymbol{p}$ is outside the triangle $\mathbf{F}$, the closest point $\boldsymbol{p}^*$ must lie on one of the triangle's edges. Therefore, we need to compute the closest point from $\boldsymbol{p}$ to each of the 3 edges of the triangle and select the one with the minimum distance.

For each edge $\boldsymbol{v}_i\boldsymbol{v}_j$, we first compute the parameter t such that the projection (closest) point $\boldsymbol{p}^{'} = \boldsymbol{v}_i + t(\boldsymbol{v}_j - \boldsymbol{v}_i)$. We have

$$t = \frac{(u-x_i)(x_j-x_i)+(v-y_i)(y_j-y_i)}{(x_j-x_i)^2+(y_j-y_i)^2}. \tag{A18}$$

If $0 \leq t \leq 1$, the projection point lies on the edge, and the barycentric coordinates of $\boldsymbol{p}^{'}$ can be represented as $\boldsymbol{w}^{'} = \begin{bmatrix} w_0^{'} & w_1^{'} & w_2^{'} \end{bmatrix}$, where $w_i^{'} = 1 - t^{'}, w_j^{'} = t$, the rest one $= 0$. If $t < 0$, then $\boldsymbol{p}^{'} = \boldsymbol{v}_i$, $w_i^{'} = 1$, the rest $= 0$. If $t > 1$, then $\boldsymbol{p}^{'} = \boldsymbol{v}_j$, $w_j^{'} = 1$, the rest $= 0$.

Perform these computations for the three edges $\boldsymbol{v}_0\boldsymbol{v}_1$, $\boldsymbol{v}_1\boldsymbol{v}_2$, $\boldsymbol{v}_2\boldsymbol{v}_0$, then choose the $\boldsymbol{p}^{'}$ with the smallest distance. Its barycentric coordinates $\boldsymbol{w}^{'}$ are the desired solution $\hat{w}^*$.

C. Implementation Details

In this section, we present implementation details of our method. Our codebase is built on SoftRas. However, we follow DIB-R [5] and separately compute the foreground and background pixels. Moreover, in the original Softras implementation, the probability map A_j^i is defined as:

$$A_i^j = \text{sigmoid}\left(-\frac{d(\boldsymbol{p}_i, \mathbf{F}_j)}{\delta}\right). \tag{A19}$$

We change it to exponential function for smoother gradients [5]:

$$A_i^j = \exp\left(-\frac{d(\boldsymbol{p}_i, \mathbf{F}_j)}{\delta}\right). \tag{A20}$$

In addition, we find that enabling *Aggregate Function* in SoftRas [22] results in a total reconstruction failure in the optimization task, so we disable it in all of our experiments.

D. Segmentation Analysis

In the main paper, we decompose the entire rotation into 12 segments. In this section, we will illustrate the quality of forward rendered images and backward gradients with respect to the number of segments used.

Results are illustrated in Fig. A1. If using fewer than 12 segments (e.g., 6 segments) leads to severe artifacts in the forward rendering. Conversely, employing more than 12 segments increases the computational cost significantly, with only marginal improvement in rendering quality.

Therefore, as a trade-off between rendering quality and computational efficiency, 12 segments are chosen in our experiments.

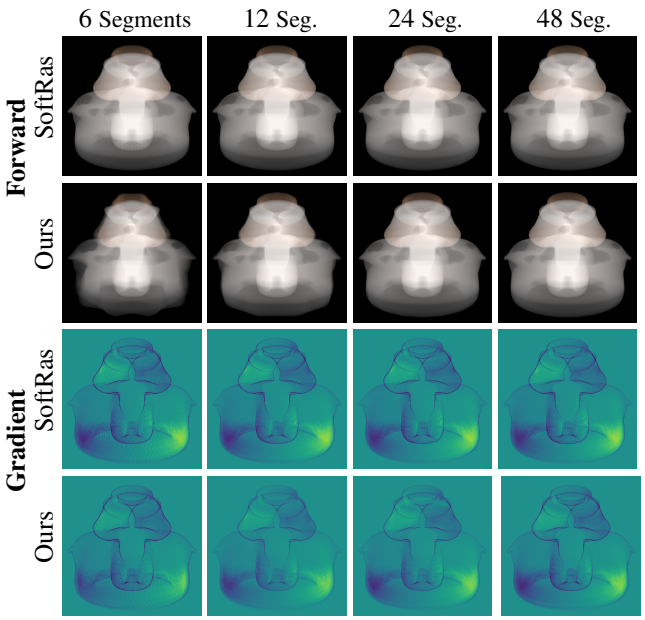


Figure A1. Impact of Segment Count on Rendering and Gradient Quality. Forward rendering and backward gradient visualization demonstrating the effect of different segment counts on motion-blur synthesis. As illustrated, using fewer than 12 segments (e.g., 6 segments) introduces severe artifacts in forward rendering and compromises gradient quality. Increasing the segment count to 12 significantly improves both rendering smoothness and gradient accuracy.

E. Visualization Details

In this section, we provide detailed explanations of the rendering process for the images presented in Fig. 3.

All forward images are rendered using the same camera parameters and blur settings (*i.e.*, translation or rotation speed) as those used in our main experiments. For translational motion, we do not decompose the motion into segments. For rotational motion, the entire rotation circle is always decomposed into 12 segments. Consequently, for rotational motion blurred with a total of 12, 60 or 240 samples, these are respectively distributed as 1, 5 and 20 samples per segment, given our decomposition into 12 segments.

For gradient images, we compute gradients with respect to the X-positions of all vertices. After obtaining these per-vertex gradient scalars, we then render these scalars into single-channel grayscale images, which are subsequently color-mapped using the Viridis color-map for visualization.

All images are rendered at a resolution of 512×512 pixels.

F. Failures of Mesh in Rotational Optimization

In rotational optimization, we observe that directly optimizing mesh vertices fails to recover well-shaped objects. One such failure case is illustrated in Fig. A2. The mesh representation consistently fails to recover a well-shaped object, even when incorporating smoothing regularization during optimization. In contrast, the results obtained with the SDF representation are substantially superior.

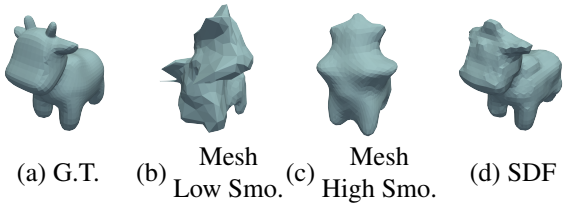


Figure A2. The results of rotational optimization. Mesh representation fails to recover a well-shaped Spot cow, no matter how smooth it is. Instead, SDF representation recovers a significantly better Spot cow.

G. Details of Loss Terms

In this section, we provide detailed definitions of additional loss terms not explicitly covered in our main paper.

Laplacian Loss Our definition of Laplacian loss follows [22]. For each vertex v, let $\mathcal{N}(v)$ be the set of adjacent vertices of v. The Laplacian loss is then defined as:

$$\mathcal{L}_{\text{L}} = \sum_{v} \left\| \delta_v - \frac{1}{|\mathcal{N}(v)|} \sum_{v' \in \mathcal{N}(v)} \delta_{v'} \right\|^2 . \tag{A21}$$

where δ_v denotes the predicted movement of vector v. This Laplacian loss encourages adjacent vertices to move consistently, thereby promoting mesh deformation smoothness.

Smoothness Loss Our definition of Smoothness loss is the same as [22]. For all two neighboring faces sharing the edge e_i, let θ_i be the dihedral angle between the two faces. We have

$$\mathcal{L}_{\text{s}} = \sum_{e_i} \left(\cos(\theta_i) + 1\right)^2 . \tag{A22}$$

This smoothness loss encourages adjacent faces to have similar normal directions, thereby penalizing sharp edges.

Regularization Loss in FlexiCubes We incorporate the regularization loss provided in FlexiCubes [35]. It is defined as:

$$\mathcal{L}_{\text{reg}} = \lambda_{\text{dev}} \mathcal{L}_{\text{dev}} + \lambda_{\text{sign}} \mathcal{L}_{\text{sign}}. \tag{A23}$$

For $\mathcal{L}_{\text{dev}}$, it is defined as:

$$\mathcal{L}_{\text{dev}} = \sum_{v \in V} \text{MAD}\left[\{|v - u_e|_2 : u_e \in \mathcal{N}(v)\}\right], \tag{A24}$$

where V denotes the set of voxel grid vertices, $|\cdot|2$ denotes Euclidean distance, $\text{MAD}(Y) = \frac{1}{|Y|}\sum y \in Y|y - \text{mean}(Y)|$ is the Mean Absolute Deviation, and $\mathcal{N}(v)$ denotes the set of adjacent vertices of v. This term penalizes the variability of distances between a vertex v and its neighbors $u_e \in \mathcal{N}(v)$.

For $\mathcal{L}_{\text{sign}}$, it is defined as:

$$\mathcal{L}_{\text{sign}} = \sum_{(s_a, s_b) \in \mathcal{E}_g} H\left(\sigma(s_a), \text{sign}(s_b)\right), \tag{A25}$$

where $\mathcal{E}_g$ denotes the set of all edges (a, b) where the scalar function values (s_a, s_b) at grid vertices a, b have differing signs (*i.e.*, cross the zero-level set). H and σ denote the cross-entropy and sigmoid functions, respectively. This term discourages the appearance of spurious geometrical structures or internal cavities in regions where explicit shape supervision is absent.

We use the same weight parameters $\lambda_{\text{dev}}, \lambda_{\text{sign}}$ as specified in [35].

Regularization Loss in Neural-Singular-Hessian We use the regularization loss provided in Neural-Singular-Hessian [41]. It is defined as:

$$\mathcal{L}_{\text{crit}} = \lambda_{\text{Eikonal}}\mathcal{L}_{\text{Eikonal}} + \lambda_{\text{singularH}}\mathcal{L}_{\text{singularH}}. \tag{A26}$$

The Eikonal loss $\mathcal{L}_{\text{Eikonal}}$ is defined as:

$$\mathcal{L}_{\text{Eikonal}} = \int_{\mathcal{P}} ||(||\nabla f(x)||_2 - 1)||_1 \, \mathrm{d}x, \tag{A27}$$

where $f(\cdot)$ denotes the SDF function and $\mathcal{P}$ denotes the set of sampling points. The Eikonal loss encourages the gradient magnitude of the SDF field to be 1, which is crucial for maintaining global smoothness and a valid SDF property.

The singular Hessian loss $\mathcal{L}_{\text{singularH}}$ is defined as:

$$\mathcal{L}_{\text{singularH}} = \int_{\mathcal{P}_{\text{near}}} ||\det(\mathbf{H}_f(x))||_1 \, \mathrm{d}x, \tag{A28}$$

where $f(\cdot)$ denotes the SDF function, $\mathcal{P}_{\text{near}}$ denotes the set of sampling points located near the zero-level set (surface), and $\det(\mathbf{H}_f(x))$ signifies the determinant of the Hessian matrix $\mathbf{H}f(x)$. The Hessian matrix is defined as the Jacobian of the gradient of f:

$$\mathbf{H}_f(x) = \begin{bmatrix} f_{xx}(x) & f_{xy}(x) & f_{xz}(x) \\ f_{yx}(x) & f_{yy}(x) & f_{yz}(x) \\ f_{zx}(x) & f_{zy}(x) & f_{zz}(x) \end{bmatrix}. \tag{A29}$$

We set the initial weighting parameters as $\lambda_{\text{Eikonal}} = \frac{50}{53}$ and $\lambda_{\text{singularH}} = \frac{3}{53}$. The same decay policy as described in [41] is adopted.

(a) Cylinder (b) Twisted Cylinder (c) Blur Image (d) Ours Result

Figure A3. A cylinder and a twisted cylinder. They share a same rotational blurred image. Given (c) as input, our optimization result is (d).

H. Analysis of 3D Losses in Rotational Optimization

In rotational optimization, we did not employ 3D losses for quantitative evaluation of reconstructed object shapes. The rationale behind this decision is detailed in this section.

In the rotational recovery task, it is common for multiple distinct 3D objects to produce rotational motion-blurred images that are indistinguishable from the input blurred image. An illustrative example is provided in Fig. A3, where the rotational motion-blurred images of both objects in Fig. A3 (a, b) result in the same blurred image shown in Fig. A3 (c).

As demonstrated in Fig. A3 (a, b), the geometric discrepancies among feasible objects can be substantial, and it is unreasonable to designate any one of these feasible objects as the ground truth. Consequently, evaluating the results using 3D losses or static image losses is not suitable.

To the best of our knowledge, the most effective evaluation for this task is to compute the differences between the rotational motion-blurred images, which is adopted in our main paper.

I. Scene Settings

In this section, we detail the specific configurations for our scenes, covering object initialization, motion parameters, and camera extrinsic and intrinsic properties.

I.1. Object Initialization

All 3D objects utilized in our experiments (*e.g.*, for gradient visualization and optimization evaluation) undergo a two-step initialization process. First, Each object is uniformly scaled such that the maximum Euclidean norm of any vertex does not exceed 1. Subsequently, each object is rotated around its local X-axis by a random angle uniformly sampled from the range $[-90°, 90°]$.

I.2. Motion Parameters

Translation For all translational motion, objects undergo a linear translation along the X-axis. The position $P(t) = (x(t), y(t), z(t))$ of a vertex that was initially at $P_0 =$

(x_0, y_0, z_0) is defined by:

$$\begin{cases} x(t) = x_0 + (0.5 - t) \\ y(t) = y_0 \\ z(t) = z_0 \end{cases} \quad \text{for } t \in [0, 1] \tag{A30}$$

This leads to the object translating linearly from an X-coordinate of $x_0 + 0.5$ at $t = 0$ to $x_0 - 0.5$ at $t = 1$.

Rotation For rotational motion, objects are rotated around the Y-axis. The angular displacement is $\theta(t) = 2\pi t$, where $t \in [0, 1]$. The position $P(t) = (x(t), y(t), z(t))$ of a vertex that was initially at $P_0 = (x_0, y_0, z_0)$ is defined by:

$$P(t) = \mathbf{R}_y(2\pi t)P_0, \tag{A31}$$

where $\mathbf{R}_y(\theta)$ is the 3D rotation matrix around the Y-axis by an angle θ:

$$\mathbf{R}_y(\theta) = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix} \tag{A32}$$

I.3. Camera

Following SoftRas, our camera setup employs a standard perspective model. The camera's eye point $\mathbf{E} = (E_x, E_y, E_z)$, from which it observes the scene origin $(0, 0, 0)$, is defined by spherical coordinates: a radial distance d, an elevation angle ϕ, and an azimuth angle θ. The conversion to Cartesian coordinates is given by:

$$\begin{cases} E_x = d\cos(\phi)\cos(\theta) \\ E_y = d\cos(\phi)\sin(\theta) \\ E_z = d\sin(\phi) \end{cases} \tag{A33}$$

For all experiments, $d = 2.232$, and $\phi \in \{-60°, -30°, 0°, 30°, 60°\}$. The azimuth angle θ varies with the motion type: (1) For translational motion, $\theta \in \{-315°, -270°, -225°, -180°, -135°, -90°, -45°, 0°\}$; (2) For rotational motion, $\theta = 0°$.

The camera's intrinsic parameters define a perspective projection with a fixed half-angular field of view $\alpha = 30°$. A 3D point $P = (x, y, z)$ in camera coordinates is projected to an image point $P_p = (x_p, y_p)$ as:

$$x_p = \frac{x}{z \cdot \tan(\alpha)} \quad \text{and} \quad y_p = \frac{y}{z \cdot \tan(\alpha)} \tag{A34}$$

J. Hyperparameter Settings

In this section, we detail the hyperparameter settings used in our experiments.

J.1. Overall Settings

Following [22], we set $\delta = 1 \times 10^{-4}$ in the probability function. Unless otherwise stated, we randomly select 25 objects from ShapeNet [3] for evaluation. The ADAM optimizer [16] is employed for optimization. Each image is rendered at a resolution of 128×128 pixels. All experiments are conducted on a single NVIDIA RTX 4090 GPU with 24GB of memory.

J.2. Translational Optimization

In this experiment, each object is rendered from 40 different viewpoints. We set $\lambda_S = 3 \times 10^{-2}$, $\lambda_L = 3 \times 10^{-4}$, and $\alpha = 0.01$, $\beta_1 = 0.5$, $\beta_2 = 0.99$ (following [22]) for the ADAM optimizer. The batch size for input views is set to 16, and each object is optimized for 1000 iterations. A sphere consisting of 1352 vertices and 2700 faces is utilized as a template mesh for deformation. We use the same method as the official SoftRas implementation for mesh texturing.

J.3. Rotational Optimization

In this experiment, each object is rendered from 5 viewpoints with varying elevations. The ADAM optimizer is configured with $\alpha = 5 \times 10^{-4}$, $\beta_1 = 0.9$, and $\beta_2 = 0.999$. The batch size for input views is set to 5, and each object is optimized for 1000 iterations. We set the initial $\lambda_{\text{crit}} = 3 \times 10^{-3}, \lambda_{\text{reg}} = 1$. The voxel-grid resolution in FlexiCubes [35] is set to 32. In our approach, we decompose the entire rotation into 12 segments, all of which are uniformly sampled.

We first pretrain the SDF field on an inclined ellipsoid (defined by $4x^2 + 2.5y^2 + 2.5z^2 - 3yz = 1$) for 500 iterations, followed by an additional 1000 iterations of optimization.

K. Further Analysis of Baselines

K.1. Shape From Blur [33]

In this section, we provide more details and analysis of our comparative experiments against Shape from Blur (SFB) [33].

Experimental Setup Adaptations for Comparison The problem formulation and experimental setup of SFB differ from our inverse rendering approach. SFB is designed to recover 3D shape and motion parameters directly from a single RGB blurred image, leveraging a pre-trained neural network (DeFMO, [32]) for intermediate guidance. Specifically, SFB takes a single RGB image and an RGB background as input, and does not require or utilize explicit object motion information (*e.g.*, translation or rotation velocities). Its core optimization loop involves:

1. Using DeFMO to predict instance-level static masks (silhouettes) for multiple intermediate timestamps from the input. These masks represent the underlying static appearance of the object at various points along its motion path.
2. Optimizing for mesh deformation (starting from a template mesh) and motion parameters (translation t, Δt, rotation r, Δr) through a single-view, differentiable rendering pipeline.
3. In each optimization iteration, it renders RGB images and silhouette masks for multiple timestamps.
4. The rendered silhouettes are compared against the static masks predicted by DeFMO.
5. All rendered RGB images, masked by their silhouettes and composited with the input background, are averaged to form a synthetic motion-blurred image. This image is then compared against the input RGB image. These losses drive the backward propagation and optimization.

In contrast, our method operates on multi-view RGB images with their corresponding non-binary transparency masks (alpha channels). In addition, our method requires and utilizes object motion information (*e.g.*, trajectory, velocities) as input. During optimization, we render multi-view motion-blurred RGBA images by accumulating contributions from the object along its known motion path, which are then compared against the input images for gradient computation.

Despite these fundamental differences, SFB remains the most relevant benchmark due to the severe scarcity of alternative methods tackling 3D shape recovery from motion blur. To enable a best-effort comparison, we adapted our data for SFB. Specifically, for each input to SFB, we generate a single RGB image by masking our RGB images with corresponding transparency masks and compositing them onto a plain black background, to minimize the influence of the background to the greatest extent possible. This ensures SFB receives input that best aligns with its expected format (RGB image + background) while making our data compatible. We kept SFB's camera parameters consistent with those used in our setup and made no other modifications to SFB's internal configurations or parameters, aiming for the most straightforward comparison.

Why SFB Performs Not So Well in These Extreme Motion Scenarios As demonstrated in the main paper (Tab. 1), our method significantly outperforms SFB for ultra-fast motion blur reconstruction. This disparity, particularly in extreme motion scenarios, primarily stems from a limitation in SFB's pipeline: its heavy reliance on the DeFMO [32] neural network for deriving intermediate static masks.

DeFMO, while generally effective for typical fast motion blur scenarios, fails when confronted with the highly diffused and ambiguous observations generated by ultra-fast motion. In such extreme cases, DeFMO struggles to accurately predict the static masks at timestamps along the motion path. As illustrated in Fig. A4, the masks produced by DeFMO for our ultra-fast motion blurred images are often highly inaccurate and entirely non-representative of the underlying object's true silhouette.

Since the DeFMO-predicted static masks serve as a fundamental guidance signal for SFB's shape and motion recovery, their inaccuracy directly propagates through the entire pipeline. This makes SFB ineffective for the ultra-fast motion blur reconstruction challenge, despite any richness in the input image data provided.

However, we acknowledge that SFB is a pioneering and important work that significantly advances the field of shape-from-blur by introducing a novel, learning-assisted approach to tackle this challenging inverse problem. Our analysis of its limitations merely highlights the unique difficulties posed by extreme motion blur. While our method demonstrates superior performance in this specific setting, the requirement for input transparency masks will be a limitation. We believe that addressing the challenges of extreme motion blur, particularly managing the ambiguity without explicit transparency, presents a significant and fertile ground for future research.

K.2. Analysis of Nvdiffrast's Gradient Computation

In the main paper (Section 6.4), we demonstrated Nvdiffrast's limited performance in reconstructing shapes from extreme motion blur. This might stem from a fundamental difference in how geometry gradients are computed.

Nvdiffrast primarily derives geometry gradients from localized, pixel-wise anti-aliasing signals along triangle edges. This means a vertex's influence on the gradient is concentrated on a few pixels it directly affects. While efficient for rendering, these localized gradients are insufficient for optimizing shape deformations from highly ambiguous, severely blurred input images. It leads to slow convergence or catastrophic failures due to a lack of meaningful gradient signals.

In contrast, our method, built on from SoftRas [22], enables each vertex to influence many pixels across a broader image region, effectively generating global and smoothed gradients. Such gradients provide a more stable signal for shape optimization. This fundamental difference in gradient computation contributes to robust 3D shape recovery in our challenging scenarios.

L. More Results

In this section, we present additional experimental results.

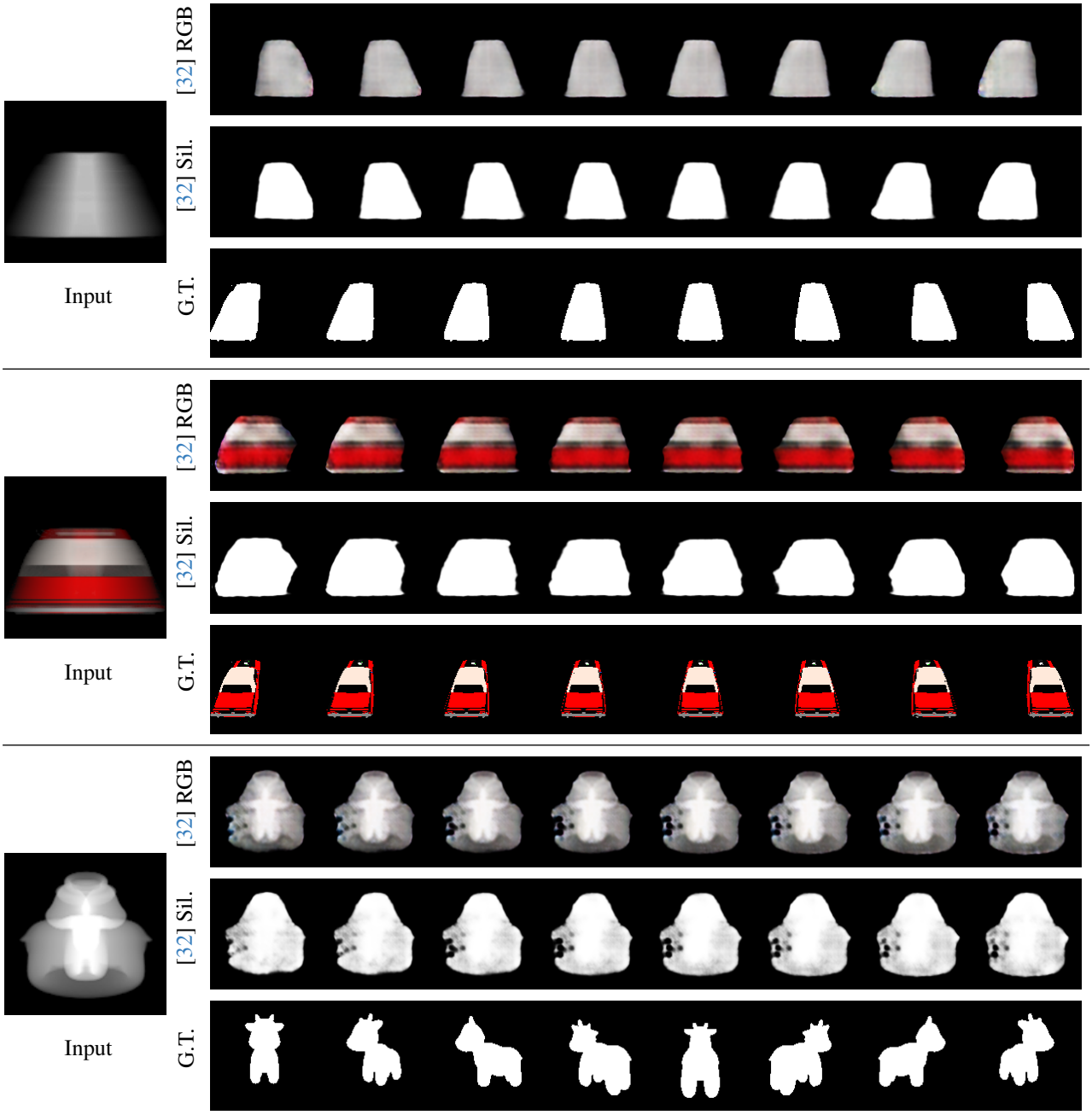
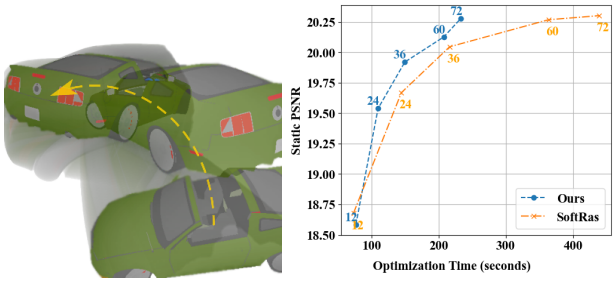


Figure A4. **Failures of DeFMO [32] in Extreme Motion Blur.** Each group displays: **Left.** Input motion-blurred image. **Right.** Three rows presenting results from DeFMO: **Top Row.** RGB images predicted by DeFMO at various timestamps. **Middle Row.** Corresponding static masks (silhouettes) predicted by DeFMO. **Bottom Row.** Ground Truth (G.T.) static masks at the respective timestamps. As illustrated, DeFMO [32] fails to predict accurate static masks under these extreme motion conditions. This fundamental inaccuracy in DeFMO's prior critically undermines the optimization guidance for SFB [33], ultimately leading to its reconstruction failures in the challenging scenarios.

L.1. Parabolic Recovery

We provide an evaluation on a more complex motion type: combined translational and rotational motion along a parabolic trajectory.

In this experiment, each vertex $P_0 = (x_0, y_0, z_0)$ undergoes a two-step transformation to define its motion path over time $t \in [0, 1]$. The vertex is first rotated around the Y-axis by an angle $\theta(t) = \pi t$, where $t \in [0, 1]$. The intermediate rotated position $P_{rot}(t)$ is given by $P_{rot}(t) = \mathbf{R}y(\pi t)P_0$. Specifically, for $P_{rot}(t) =$



(a) Illustration (b) Evaluation

Figure A5. Shape recovery for complex motion trajectories (parabolic translation + rotation). Labels indicate the corresponding number of samples. We achieve better performance than SoftRas.

$(x_{rot}(t), y_{rot}(t), z_{rot}(t))$:

$$\begin{cases} x_{rot}(t) = x_0 \cos(\pi t) + z_0 \sin(\pi t) \\ y_{rot}(t) = y_0 \\ z_{rot}(t) = -x_0 \sin(\pi t) + z_0 \cos(\pi t). \end{cases} \tag{A35}$$

Subsequently, a translation vector $T(t) = (T_x(t), T_y(t), T_z(t))$ is applied to the rotated position $P_{rot}(t)$. Let $s(t) = 0.5 - t$. The components of this translation vector are:

$$\begin{cases} T_x(t) = s(t) \\ T_y(t) = -4s(t)^2 + 0.5 \\ T_z(t) = s(t). \end{cases} \tag{A36}$$

The final position $P(t) = (x(t), y(t), z(t))$ at time t is then $P(t) = P_{rot}(t) + T(t)$. Specifically:

$$\begin{cases} x(t) = x_{rot}(t) + (0.5 - t) \\ y(t) = y_{rot}(t) + (-4(0.5 - t)^2 + 0.5) \\ z(t) = z_{rot}(t) + (0.5 - t). \end{cases} \tag{A37}$$

Illustration and evaluation results are shown in Fig. A5.

L.2. Accelerating Existing Pipelines

We further demonstrate the potential of our method as an accelerator for existing optimization-based inverse rendering pipelines. We integrate our method into [33], replacing its original rendering component. We evaluate its performance by comparing total optimization time and reconstruction quality (TIoU, PSNR, SSIM) against the original [33].

Quantitative comparison results are presented in Tab. A1. Our integration reduces the optimization time while maintaining comparable reconstruction quality. These results demonstrate our method's effectiveness in accelerating existing inverse rendering pipelines, thereby enabling them to tackle complex, real-world motion blur scenarios with greater efficiency.

Moreover, these results also demonstrate that our method can leverage existing pipelines (*e.g.*, [33]) to handle diverse real-world scenarios.

Method	Falling Objects Dataset			Time (s)
	TIoU ↑	PSNR ↑	SSIM ↑	
[33]	0.678	26.133	0.736	60.663
[33] + Ours	0.678	26.010	0.731	47.227

Table A1. Evaluation on the FMO real-world benchmark. Note time contains both rendering and data processing steps. Our solver can be integrated into [33]'s pipeline, providing faster optimization with comparable performance. "+ Ours" denotes replacing the Kaolin DIB-R rasterizer with ours but retaining the texture mapping module. Since the time cost for per template mesh remains similar, as reported in [33], we follow the best settings but use the Voronoi sphere as the template mesh only, and split the trajectory into 8 segments in our method. We have tried our best to make reproduction (the top row) but small discrepancy in performance still exists, which might impact little on our time-oriented evaluation. Results show that with the complement of out method (the bottom row), a speedup can be achieved without significant losses of performance. In addition, Kaolin DIB-R is a highly-optimized CUDA renderer, while our method is lack of low level CUDA optimization. We believe that with more such optimization, our method can achieve a more significant acceleration.

M. Detailed Limitations and Future Work

In this section, we provide a detailed discussion on the limitations of our method and potential directions for future research.

Dependency on Known Motion and Poses Similar to many inverse rendering approaches, our current optimization pipeline requires known camera intrinsics, poses, and motion information. In unconstrained settings, obtaining these parameters can be challenging. A promising direction is to integrate our differentiable renderer with motion estimation modules (*e.g.*, [33]) to jointly estimate motion trajectories and shape from the input image. We present a preliminary trial of this integration in Sec. L.2.

Motion Linearity Assumption Our fast barycentric solver assumes that motion within each time segment is linear. While this approximation holds for short exposure times, highly complex non-linear motions may introduce errors. Addressing this would require modeling higher-order motion trajectories or employing finer temporal segmentation, which we leave for future optimization.

Photometric Assumptions Our rendering model assumes a linear photometric relationship between the scene

radiance and pixel intensity. However, real-world camera ISPs (Image Signal Processors) typically apply non-linear tone mapping curves (*e.g.*, Gamma correction) to compress high dynamic range data for display. Furthermore, high-speed photography often necessitates high ISO settings to compensate for short exposure times (if not blurring intentionally), or operates in low-light conditions where the signal-to-noise ratio is low. Our current model does not explicitly account for non-linear camera response functions or sensor noise. Future work could incorporate learnable camera response functions (CRFs) and noise modeling to enhance reconstruction robustness in raw, in-the-wild footage.