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.

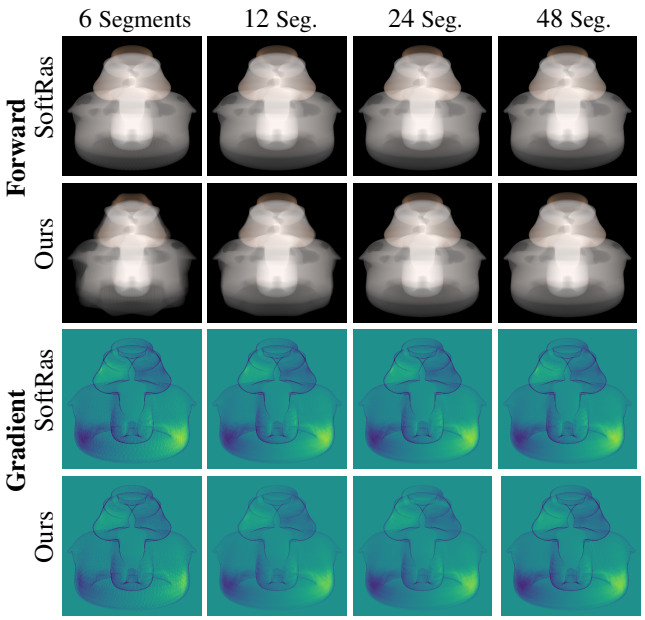


Figure A1. Impact of Segment Count on Rendering and Gradient Quality. Forward rendering and backward gradient visualization demonstrating the effect of different segment counts on motion-blur synthesis. As illustrated, using fewer than 12 segments (e.g., 6 segments) introduces severe artifacts in forward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

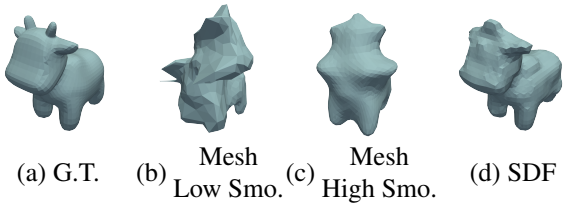


Figure A2. The results of rotational optimization. Mesh representation fails to recover a well-shaped Spot cow, no matter how smooth it is. Instead, SDF representation recovers a significantly better Spot cow.

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.

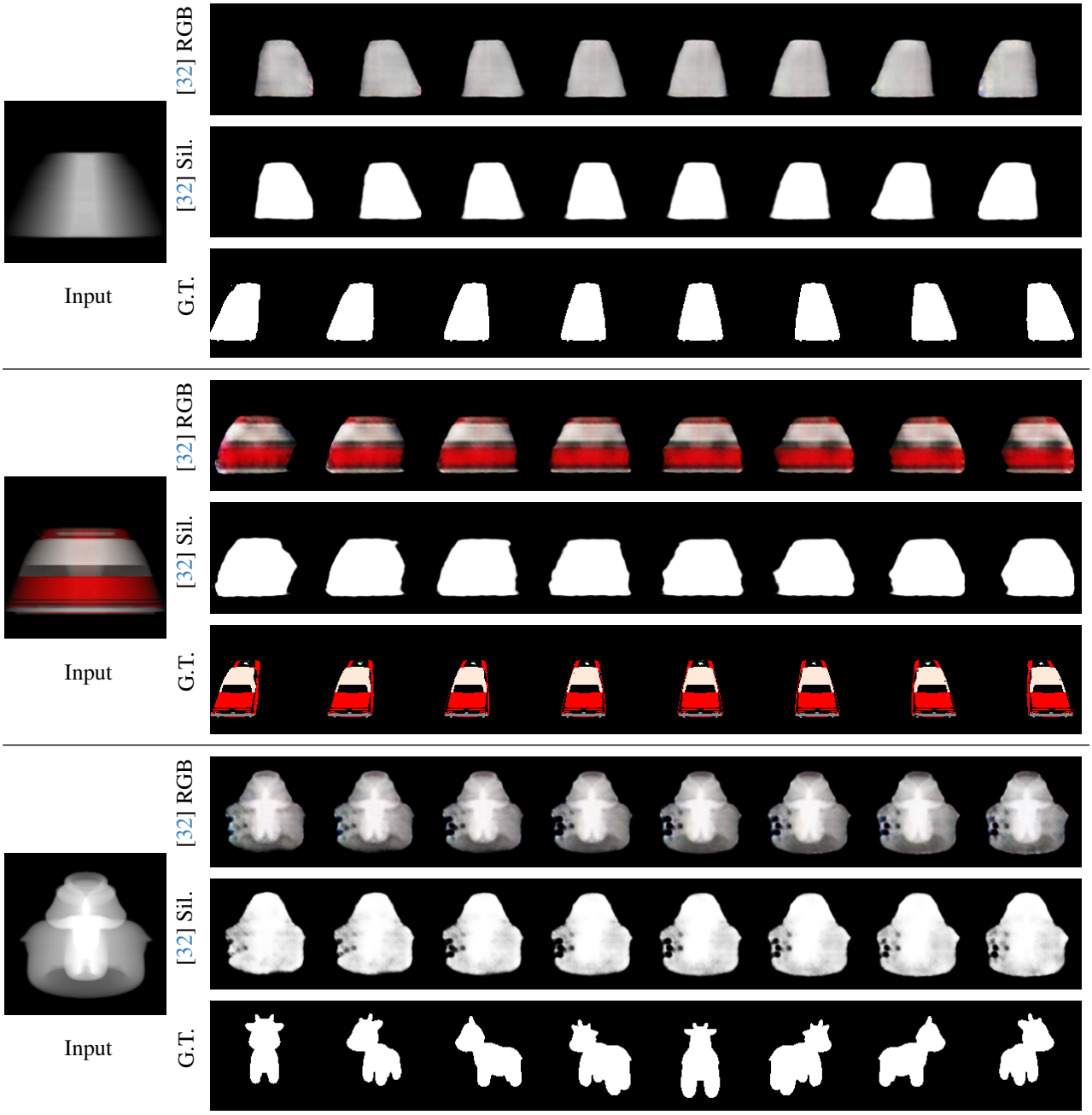
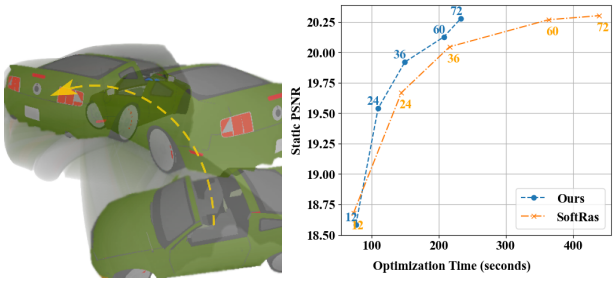


Figure A4. **Failures of DeFMO [32] in Extreme Motion Blur.** Each group displays: **Left.** Input motion-blurred image. **Right.** Three rows presenting results from DeFMO: **Top Row.** RGB images predicted by DeFMO at various timestamps. **Middle Row.** Corresponding static masks (silhouettes) predicted 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.