

Splatter Layout: Geometry-embedded 3D Reconstruction via Surface Unfolding

Supplementary Material

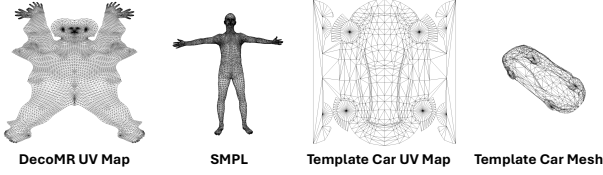


Figure 1. Initial template Mesh of Human (SMPL) and Car Domain (Custom Made).

A. Details of Unfolded Layout Generation

In this section, we present additional information on the unfolded layout generation process. The pipeline is divided into three main stages: preprocessing, optimization, and post-processing.

A.1. Pre-processing

Given an image-aligned template mesh (SMPL or car template model) $\mathcal{M}_{3D}(\mathbf{v}, \mathbf{f})$, we preprocess the data through normalization and topology alignment. For normalization, we uniformly scale the mesh so that the longest axis of its 3D bounding box fits within $[-1, 1]$, preserving the aspect ratio across all axes, and translate it so that its center aligns with the origin $[0, 0, 0]$. For topology alignment, we remap each mesh to establish a one-to-one vertex-wise correspondence with the template UV map. Specifically, we adopt the topology from DecoMR for human [17] and a custom-designed template mesh for cars (See Figure 1). This remapping is achieved by reordering the vertices of the 3D mesh and duplicating those lying on UV seam boundaries. This is necessary due to discontinuities in the template UV map caused by seams, which break surface connectivity. By duplicating seam vertices and updating the face indices accordingly, we obtain a one-to-one mapping between 3D mesh vertices \mathbf{v} and their corresponding UV layout coordinates \mathbf{t} .

A.2. Optimization Procedure

Our optimization procedure is guided by an unfolding network inspired by FoldingNet [15], whose ability to model point-wise correspondences makes it well-suited for geometry-aware optimization. We begin by encoding the preprocessed 3D template mesh vertices \mathbf{v} using the FoldingNet encoder, modified to remove batch normalization. The resulting latent feature is concatenated with the template mesh vertex texture coordinate \mathbf{t} , and the combined vector is passed through a two-block MLP decoder:

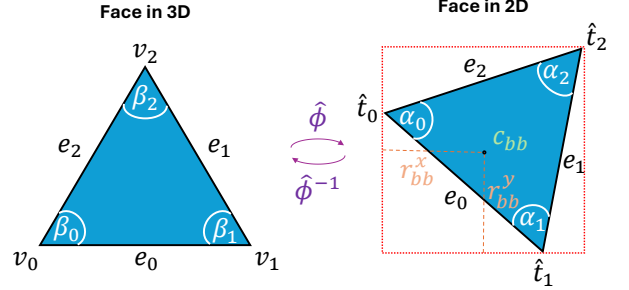


Figure 2. Illustration of a triangle face used during data generation. This face utilizes internal angles function α and β for computing angular distortion, and bounding box for computing repulsion.

- **MLP 1** [514, 1024, 1024, 1024, 128] : processes the input \mathbf{v} and produces a 128-dimensional intermediate feature.
- **MLP 2** [130, 1024, 1024, 1024, 2] : takes input the concatenation of texture coordinate \mathbf{t} and 128-dimensional feature, outputs the final 2D prediction $\hat{\mathbf{t}}$.

We optimize each sample independently using AdamW optimizer [9], with a learning rate schedule decaying from 1×10^{-5} to 1×10^{-7} over 10,000 iterations. To support masked optimization, the occlusion mask M^{vis} is rendered in image space at a resolution of 256×256 using PyTorch3D [12]. The loss weights are set as:

$$\begin{aligned} \text{Human: } \lambda^{\text{align}} &= 0.1, \lambda^{\text{angle}} = 0.01, \lambda^{\text{local}} = 10.0, \lambda^{\text{global}} = 0.01, \\ \text{Cars: } \lambda^{\text{align}} &= 1.0, \lambda^{\text{angle}} = 0.1, \lambda^{\text{local}} = 0.1, \lambda^{\text{global}} = 0.0 \end{aligned}$$

The loss further incorporates three not yet defined key components: angle calculation, bounding box, and repulsion mask. Figure 2 illustrates the mesh-face-derived structures that inform these terms, each contributing a distinct role within the overall formulation. We detail each component below.

Angle Calculation The angle denoted by γ refers to the internal angle of a triangle, calculated by $\alpha \in \mathbb{R}^2$ in the 2D domain or $\beta \in \mathbb{R}^3$ in the 3D domain. In Section 3.1 in the main paper, we define a face as a triplet of vertices, with its edges expressed as $\mathbf{e} = (\overrightarrow{\mathbf{p}_0\mathbf{p}_1}, \overrightarrow{\mathbf{p}_1\mathbf{p}_2}, \overrightarrow{\mathbf{p}_2\mathbf{p}_0})$. Accordingly, the vectorized formulation of the edge set can be written as:

$$[\mathbf{e}_0(\mathbf{p}, \mathbf{f}), \mathbf{e}_1(\mathbf{p}, \mathbf{f}), \mathbf{e}_2(\mathbf{p}, \mathbf{f})] = [\mathbf{p}_1 - \mathbf{p}_0, \mathbf{p}_2 - \mathbf{p}_1, \mathbf{p}_0 - \mathbf{p}_2].$$

The internal angle γ at each vertex of a triangle can be computed as the angle between the two incident edge vectors. Specifically:

$$\gamma(\mathbf{p}, \mathbf{f}) = \cos^{-1} \left(\frac{\mathbf{e}_a(\mathbf{p}, \mathbf{f}) \cdot -\mathbf{e}_b(\mathbf{p}, \mathbf{f})}{\|\mathbf{e}_a(\mathbf{p}, \mathbf{f})\| \cdot \|\mathbf{e}_b(\mathbf{p}, \mathbf{f})\| + \varepsilon} \right), \quad (1)$$

where \mathbf{e}_a and \mathbf{e}_b denote the two incident edge vectors of face f meeting at vertex \mathbf{p} , and ε is a small positive constant added to ensure numerical stability.

Bounding Box Calculation Each 2D triangle face is enclosed by a bounding box whose center is denoted as c_{bb} , and the radius is defined as $r_{bb} = \max(r_{bb}^x, r_{bb}^y)$, where r_{bb}^x and r_{bb}^y are half the width and height of the bounding box, respectively.

Algorithm 1: Repulsion Mask Generation

Input: Occlusion mask for faces or vertices M^{vis}
Output: Repulsion constraint mask M (e.g., M^{soft} or M^{bbox})

```

1 Initialize  $M$  as a zero matrix of size
  Length( $M^{\text{vis}}$ )  $\times$  Length( $M^{\text{vis}}$ );
2 for  $i = 1$  to Length( $M^{\text{vis}}$ ) do
3   Set  $M_{[ii]} \leftarrow \infty$  or 1; // Self-repulsion
4   if  $M_{v_i}^{\text{vis}}$  is not occluded then
5     for  $j = 1$  to Length( $M^{\text{vis}}$ ) do
6        $M_{[ij]} \leftarrow \infty$  or 1;
```

Repulsion Mask Generation Algorithm Algorithm 1 describes the generation of repulsion masks M^{soft} or M^{bbox} , which are used to control the pairwise repulsion loss during training. For both losses, local and global, we follow the same principle: (1) self-repulsion is disabled, and (2) interactions between two non-occluded points are masked out to prevent them from repelling each other.

This ensures that only occluded-to-visible or occluded-to-occluded interactions contribute to the repulsion loss, encouraging surface unfolding in challenging regions. The implementation differs slightly between the two loss function types:

- local employs a masking strategy with large constants (e.g., ∞) to ignore masked distances in the KNN-based computation.
- global uses binary masks and applies masked fill operation applied directly to the loss matrix.

This approach is inspired by adjacency matrix manipulation in graph theory, where selected edges are suppressed either by exclusion or by assigning infinite distances—effectively controlling which interactions contribute to the loss.

A.3. Post-processing

Post-processing filters the training outputs to ensure suitability for the Gaussian parameter prediction pipeline. Every 1,000 epochs, we compute the edge intersection ratio

(ER) to assess the UV layout quality. The final ground-truth parameterization is chosen based on the lowest edge intersection ratio among epochs ≥ 5000 , ensuring that only stable and well-converged layouts are selected.

B. Splatter Layout Framework and Its Applications to Downstream Tasks

Fig. 3 illustrates the overall pipeline of Splatter Layout, which adopts the same U-Net architecture as Splatter Image, with the only modification being the number of output channels. Specifically, we add 9 additional channels: 2 for UV coordinates \mathbf{u} , 3 for surface normals \mathbf{n} , 3 for non-rigid deformation offsets $\Delta \mathbf{p}_n$, and 1 for the normal direction offset $\Delta \mathbf{d}_n$. Splatter Layout also utilizes two rasterizers, \mathcal{R} —one renders Gaussian in $\mathbb{R}_{\text{template}}^3$ space and one in $\mathbb{R}_{\text{original}}^3$ space. This representation supports various reconstruction formats, including meshes, point clouds, and Gaussian splats, and further enables downstream tasks such as animation and appearance editing. In the following sections, we describe how Splatter Layout facilitates mesh generation, motion animation, and direct editing of appearance.

B.1. Mesh Generation

We adopt two complementary strategies for mesh reconstruction: one that directly preserves the template mesh topology, and another that employs Poisson surface reconstruction to handle geometric artifacts and holes, particularly near UV seams.

The first strategy leverages our unified representation to directly assign geometry and appearance to the canonical set of a fixed number of template vertices \mathbf{v} . We begin by filtering valid pixels \mathbf{h} from the predicted feature map using an opacity threshold of $\sigma > 0.7$. For each valid pixel, we identify the predicted UV coordinate \mathbf{u} that is closest to a predefined template UV location \mathbf{t} , thereby establishing a correspondence between dense pixel-wise predictions (at resolution $N \times N$) and the template mesh vertex set. Each mesh vertex is assigned a new 3D position $\boldsymbol{\mu}$ and color \mathbf{c} from its nearest valid pixel, while the original face topology \mathbf{f} is retained to preserve watertightness and connectivity.

As a second strategy, we explore mesh reconstruction based on spatial neighborhood connectivity over the predicted surface. Specifically, adjacent pixels whose opacity exceeds a threshold ($\sigma > 0.7$) are connected to construct a coarse surface mesh. This approach captures fine-grained geometric details more effectively than the template mesh vertex-based method. However, due to discontinuities near seam boundaries, the resulting mesh often contains inconsistencies and holes. To address this, we apply Poisson surface reconstruction, as implemented in Open3D [19] (with depth = 15) to the coarse surface, producing a watertight mesh. While this method produces smooth and continuous

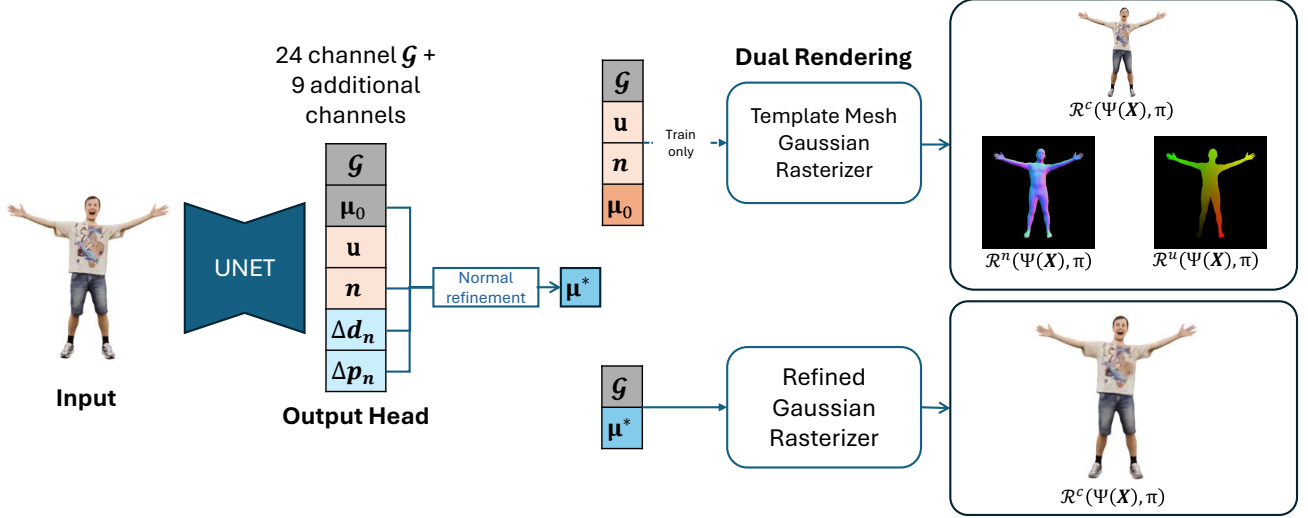


Figure 3. The overall pipeline of Splatter Layout, which employs an increased number of output channels and two Gaussian rasterizers to enable multi-attribute rendering and supervision.

geometry, it also readjusts the topology, over-smooths local details, and breaks vertex-level correspondences—making it less suitable for downstream tasks such as animation or appearance editing.

B.2. Appearance Editing

Editing the appearance is performed either by overlaying image features onto the representation or by directly modifying the color values to update the color attribute c . This can be done using image manipulation software. Given a pixel-wise one-to-one mapping, the changes are directly applied to the representation.

B.3. Human Animation

Leveraging the topological structure of the SMPL mesh, the prediction in the human domain animated using the sequences provided by the AMASS dataset [5, 10]. We first obtain the vertex positions μ from the mesh, generated according to the setup described in Section B.1. Joint positions are then computed using the SMPL joint regressor.

To reconstruct the canonical T-pose, inverse transformations are applied to the relative joint poses. The animation is subsequently driven via linear blend skinning (LBS). While this inversion process may introduce twist ambiguities and minor artifacts, it can be refined through manual rigging, enabling more predictable and accurate motion control by bypassing reliance on inverse kinematics. Finally, the resulting animation is transferred to a Gaussian Splatting representation, where blend weights are interpolated using a rasterization-based interpolation scheme.

C. Experimental Setup

This section describes the training configuration and evaluation protocol for our experiments. We provide details on optimization settings, evaluation metrics, and datasets used throughout analysis.

C.1. Training Setup

For all training experiments, Splatter Layout adopt a uniform setup with consistent camera configuration, hyperparameters, and supervision strategy. We use a learning rate of 5×10^{-5} , a batch size of 8, and optimize with the Adam optimizer [8], combined with exponential moving average (EMA) updates following the Splatter Image configuration. Training runs for 200,000 iterations and takes approximately two days on a single NVIDIA A100 GPU. Each input sample is rendered from a fixed input view at a resolution of 256×256 . For supervision, four novel target views are sampled per iteration from a predefined set of 80 uniformly distributed viewpoints on a viewing sphere, and used to compute the rendering loss, promoting generalization and multi-view consistency.

C.2. Evaluation Metrics

A diverse set of metrics is employed to evaluate both the surface layout quality and the final reconstruction performance of the Splatter Layout framework. For layout evaluation, we measure geometric consistency, bijectivity, and input view alignment accuracy. For reconstruction, we assess geometric fidelity, UV validity, and view synthesis quality using both perceptual and pixel-level criteria.

Data Generation Metrics

- **Mean Square Error (MSE):** The mean squared ℓ_2 distance between the projected 3D vertex $\Pi(\mathbf{v})$ and its corresponding UV coordinate \mathbf{t} is used to evaluate input view alignment accuracy. We report two variants of this metric: $\text{MSE}^{\mathcal{V}^{\text{vis}}}$, computed over the visible vertex set \mathcal{V}^{vis} , and $\text{MSE}^{\mathcal{V}}$, computed over the full vertex set. The metric is defined as:

$$\text{MSE} = \frac{1}{|\mathcal{V}|} \sum_{\mathbf{v} \in \mathcal{V}} \|\Pi(\mathbf{v}) - \mathbf{t}\|^2. \quad (2)$$

- **Edge Intersection Ratio (ER):** To evaluate the bijectivity of the Splatter Layout framework, the ratio of intersecting edges within the mapping is computed. Here, intersections of each edge are efficiently detected using the [1] sweep line algorithm. This ratio is formally expressed by the following formula:

$$\text{ER} = \frac{\text{intersected edges}}{\text{total number of edges}} \times 100\%. \quad (3)$$

- **Angular Distortion (AD)** Similar to prior work [13], which optimizes angular metrics in radians, we define an Angular Distortion (AD) metric that explicitly operates in degrees. This metric evaluates how well the internal angles of the 3D surface $\mathcal{M}_{3D}(\mathbf{v}, \mathbf{f})$ are preserved in the 2D layout $\mathcal{M}_{2D}(\mathbf{t}, \mathbf{f})$. For each triangle $\mathbf{f} \in \mathcal{F}$, we denote the set of internal angles in 3D as β and in 2D as α , both measured in degrees. Analogous to the angle preservation term in $\mathcal{J}^{\text{unfold}}$, the angular distortion is defined as:

$$\text{AD} = \frac{1}{3|\mathcal{F}|} \sum_{i=1}^{|\mathcal{F}|} |\alpha(\mathbf{t}, \mathbf{f}_i) - \beta(\mathbf{v}, \mathbf{f}_i)|. \quad (4)$$

Splatter Layout used Metrics

- **Chamfer Distance (CD):** To evaluate geometric accuracy, we use the Chamfer Distance (CD), which measures the similarity between two point clouds. CD is defined as the sum of squared Euclidean distances from each point in one set to its nearest neighbor in the other. In our setup, CD is computed between the predicted Gaussian centers $\mu \in \mathcal{G}$ and a reference point set \mathcal{P} . We evaluate two variants: $\text{CD}^{\mathcal{T}}$, where \mathbf{p} is obtained via barycentric interpolation over the template mesh provided by DecoMR or template car uv map $B^{\mu}(\mathcal{M}_{2D}(\mathbf{t}, \mathbf{f}))$ and $\text{CD}^{\mathcal{M}}$, where \mathbf{p} is generated by applying Poisson disk sampling at resolution 128^2 on the original mesh using Open3D [19]. The metric is defined as:

$$\text{CD} = \sum_{\mu \in \mathcal{G}} \min_{\mathbf{p} \in \mathcal{P}} \|\mu - \mathbf{p}\|^2 + \sum_{\mathbf{p} \in \mathcal{P}} \min_{\mu \in \mathcal{G}} \|\mathbf{p} - \mu\|^2. \quad (5)$$

- **Reprojection Error (RE):** Reprojection error is computed as the average Euclidean distance between the predicted 3D points obtained via mapping from the predicted

UV $\phi^{-1}(\mathbf{u})$ and the corresponding ground-truth vertices \mathbf{v} . This metric evaluates the quality of mapping for 2D-to-3D, providing a measure of reconstruction consistency and spatial accuracy, defined as:

$$\text{RE} = \frac{1}{N} \sum_{i=1}^N \|\phi^{-1}(\mathbf{u}_i) - \mathbf{v}_i\|^2. \quad (6)$$

- **PSNR/LPIPS/SSIM:** Following the previous works [14], we evaluate the quality of view synthesis using Peak Signal-to-Noise Ratio (PSNR), Learned Perceptual Image Patch Similarity (LPIPS), and Structural Similarity Index (SSIM). During training, we utilize source views to generate images from novel camera viewpoints and evaluate their fidelity and perceptual consistency based on the aforementioned metrics.

C.3. Dataset Overview

The constructed dataset comprises 742 human meshes sourced from five public datasets: CustomHuman [7], THuman2.0 [16], TightCap [4], CLOTH4D [20], and Rocketbox [6], as well as 100 car meshes collected from ShapeNet [3] and MeshFleet [2]. For the human dataset, we employ the SMPL model together with DecoMR UV parameterization [17] to construct a consistent template mesh. For the car dataset, we design a watertight sedan template mesh and generate its UV parameterization using Blender to ensure a topology-consistent representation.

In human domain, to obtain each SMPL mesh, CustomHuman provides SMPL-X fittings, which we convert directly to SMPL format. THuman2.0 includes pre-fitted SMPL models. For the remaining datasets, we estimate SMPL parameters using a multi-view image-based fitting approach based on PaMIR [18]. We perform manual cleaning only on the TightCap dataset's 3D original mesh due to the presence of unprocessed and noisy scans. All processed data are passed through our data generation pipeline to produce the unfolded surface ground truth used for training.

For the car dataset, we align the template mesh to each target mesh using ICP, and then refine the alignment by projecting every template vertex onto the nearest target triangles, yielding a deformed template that conforms closely to the target surface. Although this procedure is not yet as mature as the human template, which is already more uniform, it nevertheless ensures bijectivity and symmetry, both of which are crucial for constructing a consistent template. We acknowledge that more advanced methods for template construction may exist, but exploring them lies beyond the scope of this paper. Importantly, our results demonstrate that the proposed method remains effective as long as a bijective template is available.

All meshes are rendered in Blender using a perspective camera setup, with 80 viewpoints for humans and 50 for

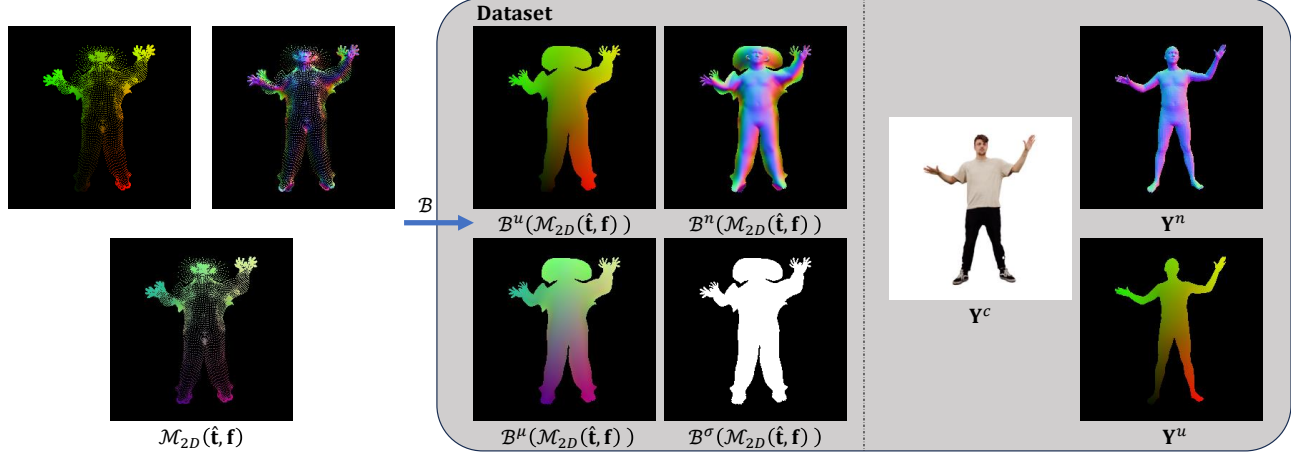


Figure 4. A sample of the data used for supervision is illustrated here. We apply barycentric interpolation on $B(\mathcal{M}(\hat{t}, f)(\mathcal{M}(\hat{t}, f)))$ to construct per-pixel surface attributes for $\mathcal{L}^{\text{surf}}$.

cars, uniformly distributed around the subject at distances of 3.5 meters and 4 meters, respectively. The entire dataset is then processed through our data generation pipeline to produce the ground-truth used for supervision. Specifically, from the template meshes we render per-vertex UV coordinates and normal maps, while the original meshes are rendered with textures to provide appearance supervision.

For the human tasks, we split the dataset into 90% for training and 10% for evaluation, while for the car tasks, we use an 80%-20% split. All data samples are generated through our data generation pipeline, producing tuples of interpolated 2D unfolded layouts $\mathcal{M}_{2D}(\hat{t}, f)$ and corresponding rendered images Y , as illustrated in Fig. 4.

C.4. Barycentric Interpolation Pre-computation

Since the unfolded layout $\mathcal{M}_{2D}(\hat{t}, f)$ is inherently sparse, we precompute a rasterization-based barycentric interpolation B^a to bridge the gap between the vertex-based attribute space $\mathbb{R}^{|v| \times D}$ and the dense image grid $\mathbb{R}^{N \times N \times D}$. Each vertex v on the 2D surface \mathcal{M} is associated with attributes a , including opacity σ , position μ , UV coordinate u , and normal n . By rasterizing these attributes into dense per-pixel maps, we enable geometry-aware, pixel-level supervision directly aligned with the unfolded layout.

$$\hat{a}_h = \sum_{i=1}^3 w_i a_{v_i}, \quad \text{where } \sum w_i = 1, w_i \geq 0.$$

Here, w_i are the barycentric coordinates of pixel h with respect to face f , ensuring \hat{a}_h is computed as a convex combination of the three vertex attributes a_{v_i} . This interpolation produces smooth per-pixel transitions and enables dense 2D maps of 3D attributes aligned with the unfolded layout.

D. Comprehensive Experimental Analysis

We conduct a thorough experimental analysis on the Human domain to evaluate the contribution of each component in our framework and to validate the effectiveness of our design choices. This section presents a series of experiments, including ablations on the Splatter Layout pipeline and evaluations of generalization to in-the-wild scenarios. Together, these experiments provide a comprehensive understanding of how each module and supervision strategy contributes to the final performance.

Table 1. Ablation Study on attribute of $B(\mathcal{M}_{2D}(\hat{t}, f))$ for $\mathcal{L}^{\text{surf}}$ supervision

	LPIPS ↓	RE ↓	CD(M) ↓
(+) B^n (+) B^u supervision	0.070	2.883	2.44
(-) B^n (+) B^u supervision	0.066	3.179	2.58
(+) B^n (-) B^u supervision	0.071	10.902	2.97
(-) B^n (-) B^u supervision	0.068	11.914	2.67

D.1. Ablation Studies of Splatter Layout

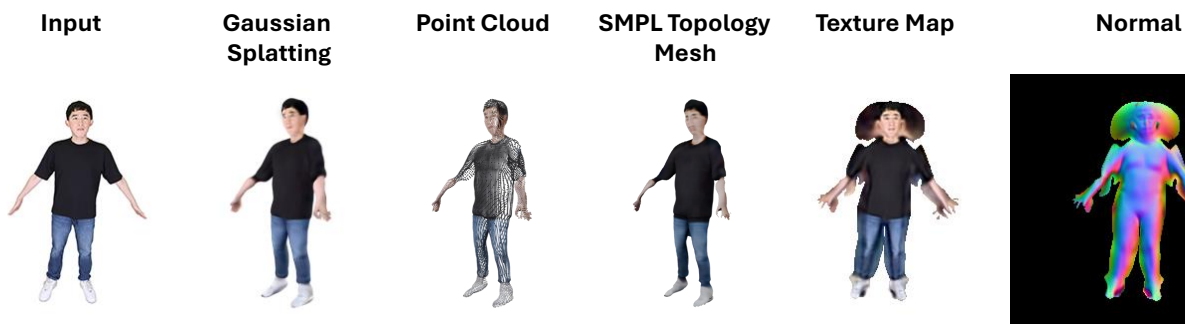
Table 1 shows that the choice of supervised attribute a in $B^a(\mathcal{M}_{2D}(\hat{t}, f))$ for $\mathcal{L}^{\text{surf}}$ affects the tradeoff between rendering quality and geometric accuracy. Using UV supervision on B^u results in a higher Chamfer Distance (CD), indicating improved geometric fidelity. In contrast, supervision on B^n leads to better positional alignment and correspondence with the SMPL surface, as evidenced by lower Reprojection Error (RE). However, the improvement in geometric quality from normal supervision comes at the cost of reduced rendering quality, indicating a trade-off between appearance fidelity and structural precision.

D.2. In the wild result

To evaluate the Splatter Layout generalization ability, we test its performance on in-the-wild images beyond the training dataset. We use inputs sourced from real-world photographs and SDXL [11]-generated content, demonstrating the robustness of our method under diverse visual conditions. Backgrounds are removed and replaced with uniform white color, and the images are processed using the same camera parameters as during training. The resulting reconstructions are shown in Fig. 5.

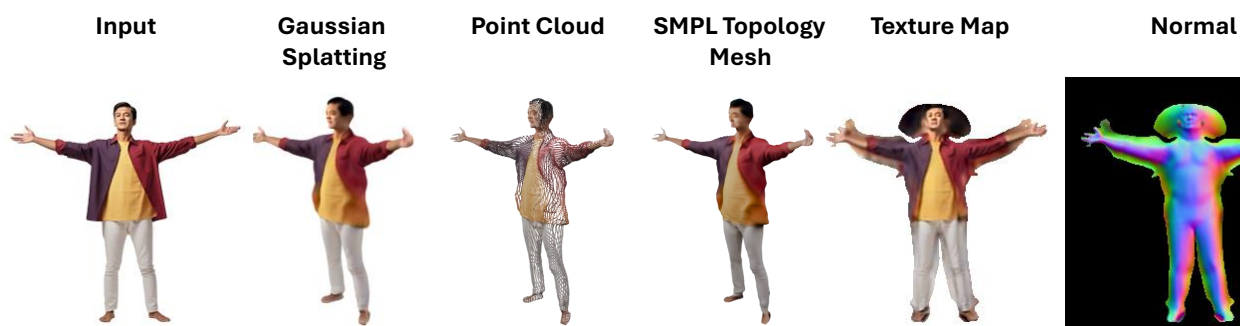
E. Visual Results

To support and complement our main findings, we present extended visual results in Figure 6, 7, 8, 9, 10, 11, 12, and 13. Figure 6, 7, and 8 highlight SMPL-based downstream tasks such as mesh reconstruction, appearance editing, and animation, showing that our representation preserves surface connectivity, supports localized texture edits without undesirable propagation to occluded regions, and maintains stability under animation. Figure 9 (human) and Figure 10 (car) compare the features produced by our method with those from Splatter Image, demonstrating that unfolded surface supervision yields more faithful geometry, smoother connectivity, and sharper normals; since Splatter Image does not directly predict normals, its normals shown here are estimated using the Open3D implementation [19]. Finally, Figure 11 and 12 (human), and Figure 13 (car) provide detailed qualitative comparisons in challenging regions such as hands and wheels, further illustrating that our method consistently reconstructs fine-grained geometry with higher accuracy than Splatter Image, producing results that are both visually convincing and structurally reliable.

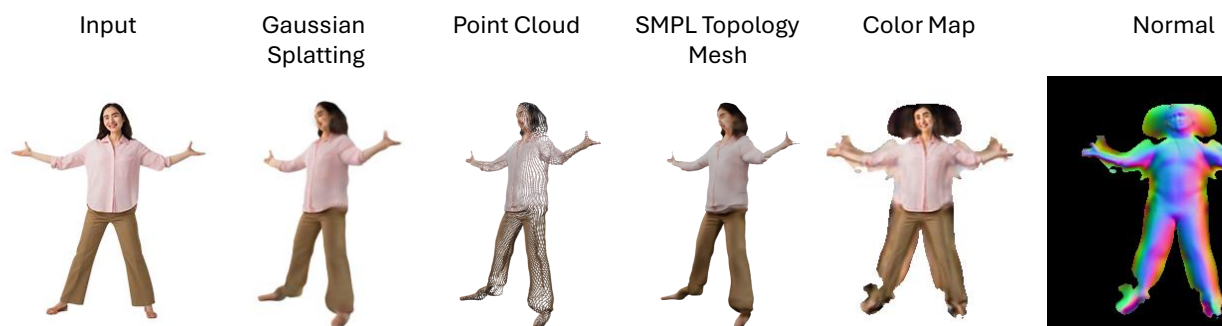


(a) The in the wild result from real images

Prompt: A man, t pose, in white plain background, colored shirt, straight arm, full body



Prompt: A woman, t pose, in white plain background, colored shirt, straight arm, full body

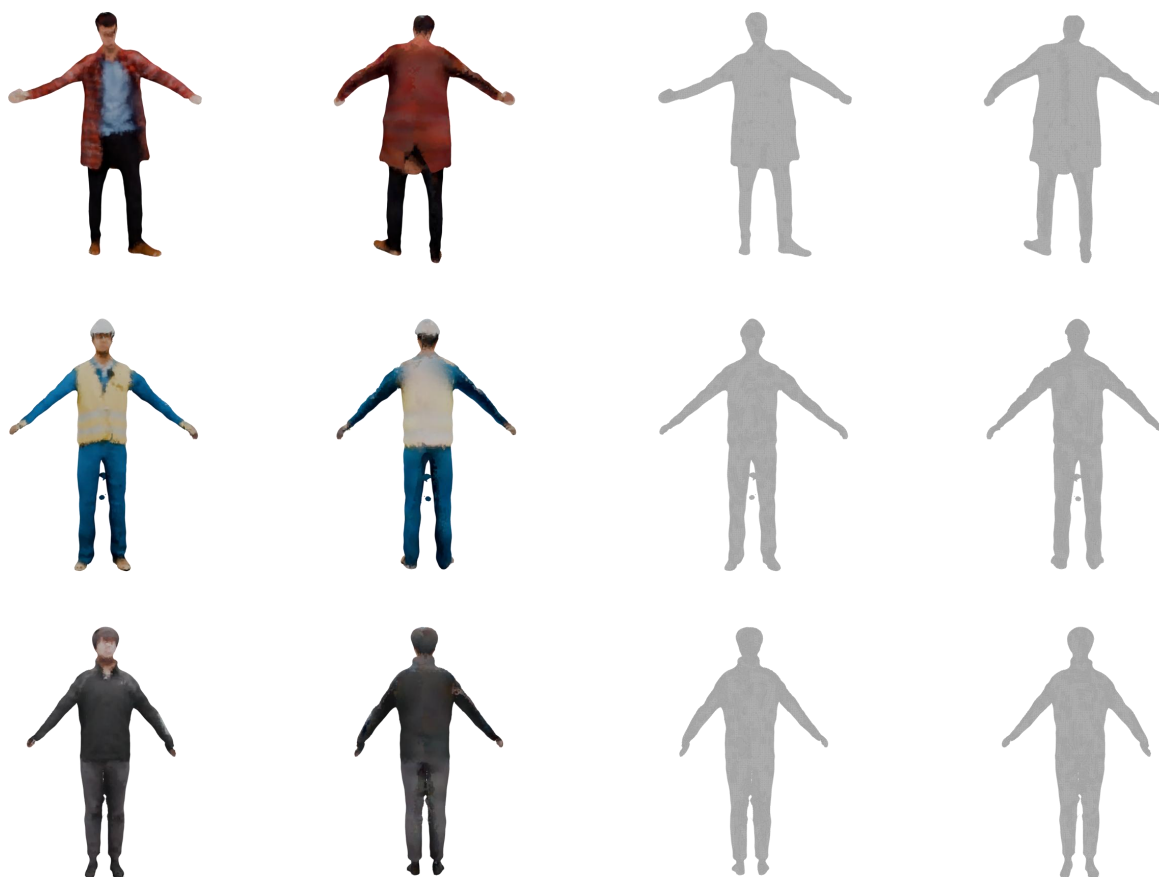


(b) The in the wild result from SDXL

Figure 5. Qualitative in-the-wild results. Our method is applicable to both (a) real-world images and (b) synthetic images generated by SDXL, demonstrating its potential to generalize across diverse input types.



(a) Frontal and back views for SMPL mesh reconstruction result.



(b) Frontal and back views for Poisson sampled mesh reconstruction result.

Figure 6. Mesh reconstruction results obtained from two different pipelines: one from mapping to SMPL topology, and the other by Poisson reconstruction

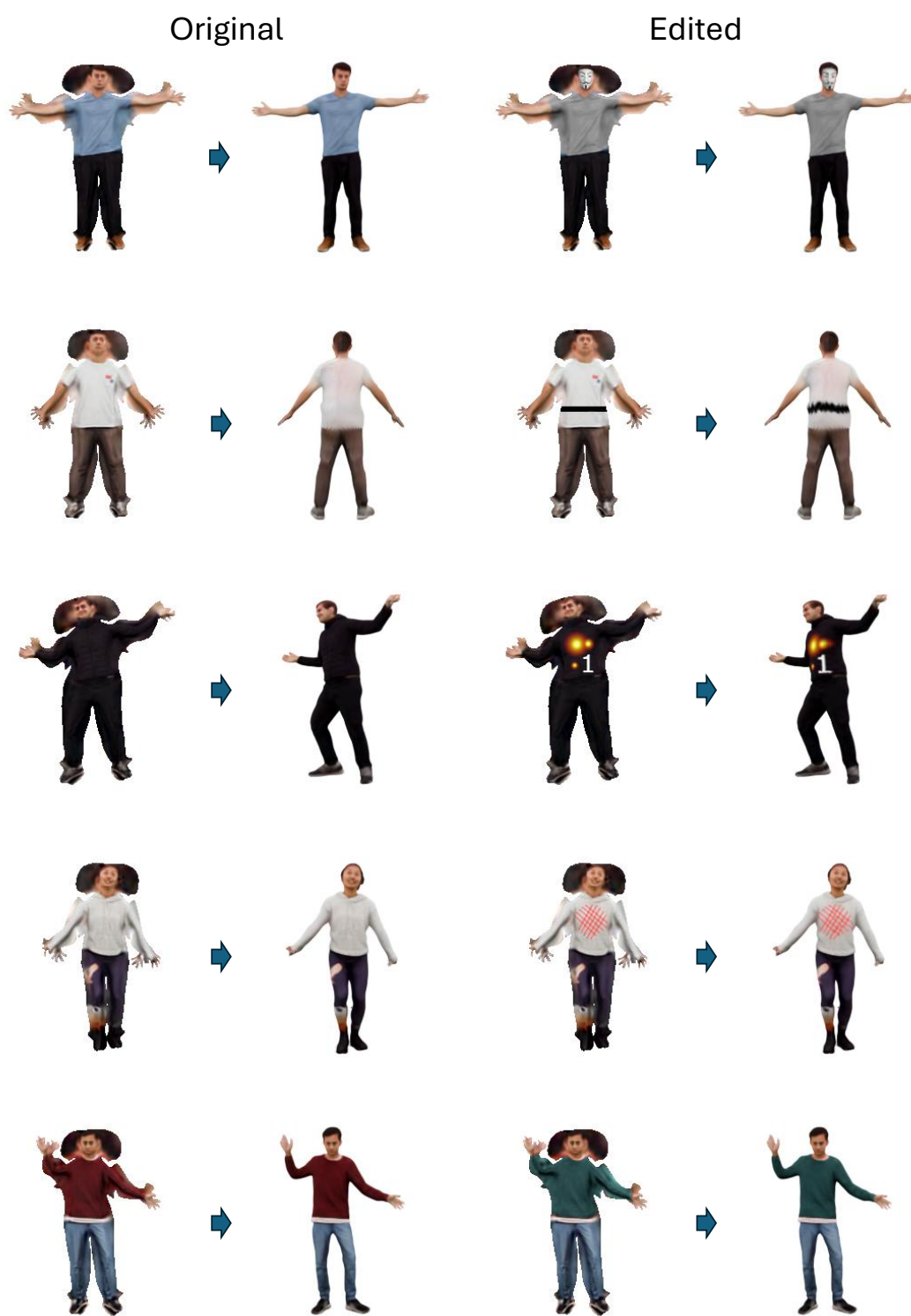


Figure 7. Appearance editing results. Left: original prediction of c . Right: edited version obtained by either overlaying an image or directly modifying the color values in the representation.



Figure 8. Animation pipeline using AMASS dataset. Joint motions are extracted through inverse transformation to reconstruct the full-body animation.

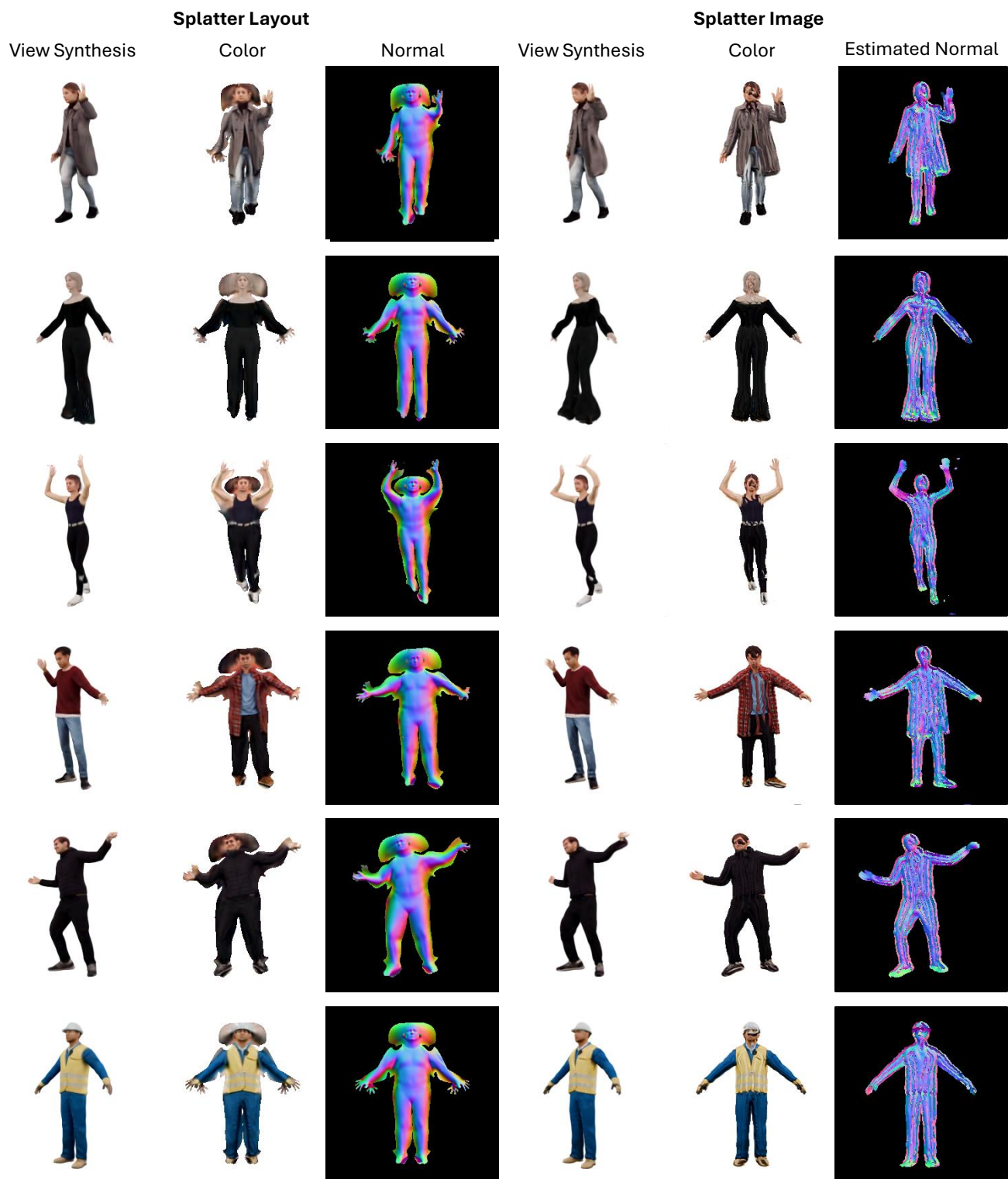


Figure 9. Qualitative comparison showcasing the difference between features produced by our method and Splatter Image.

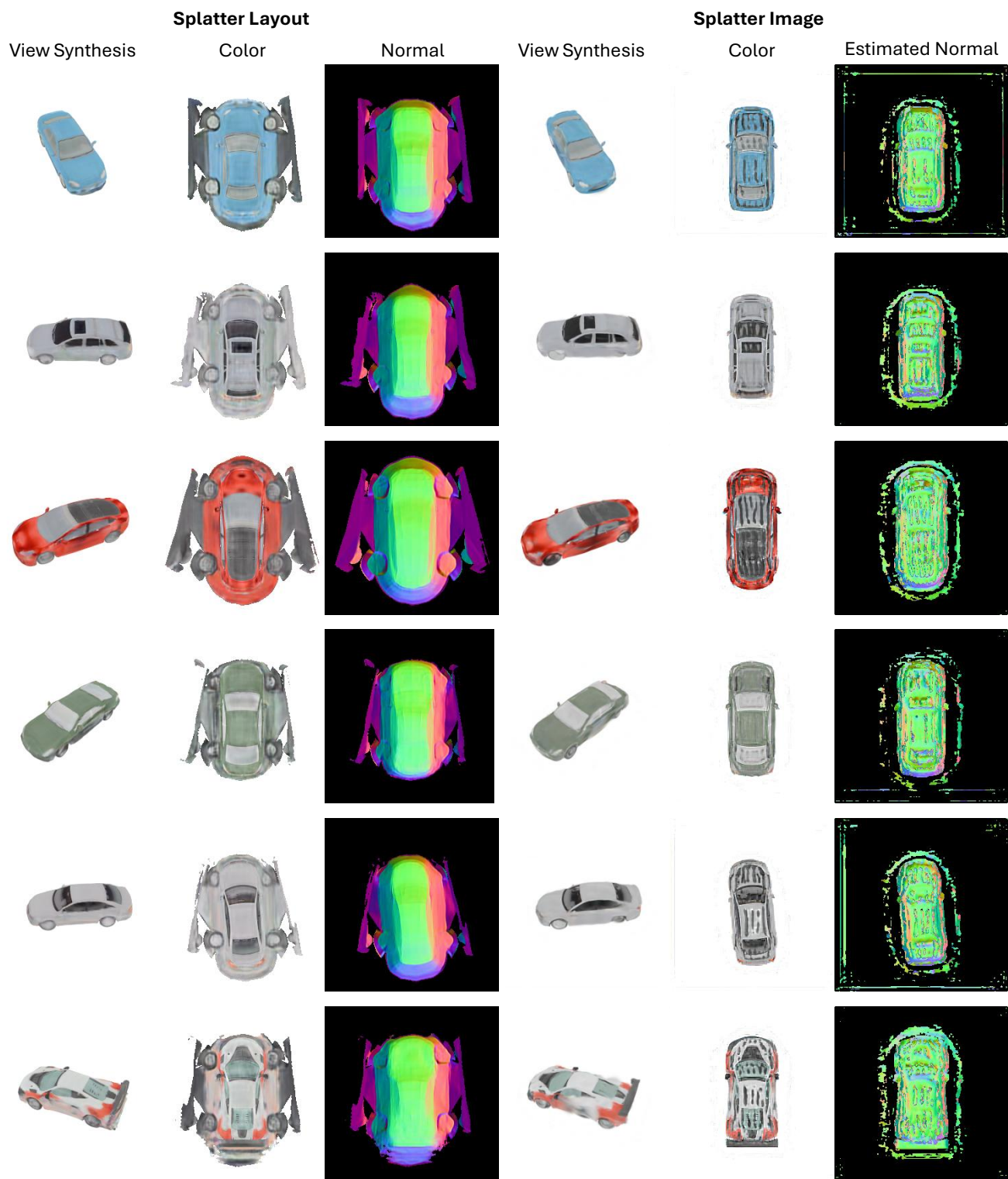


Figure 10. Qualitative comparison showcasing the difference between features produced by our method and Splatter Image.



Figure 11. Qualitative comparison showcasing improved reconstruction of fine-scale details, particularly in facial features and hand geometry. Our method preserves local structure more faithfully, resulting in sharper and more accurate surface detail.



Figure 12. Additional Qualitative comparison showcasing improved reconstruction of fine-scale details, particularly in facial features and hand geometry. Our method preserves local structure more faithfully, resulting in sharper and more accurate surface detail.

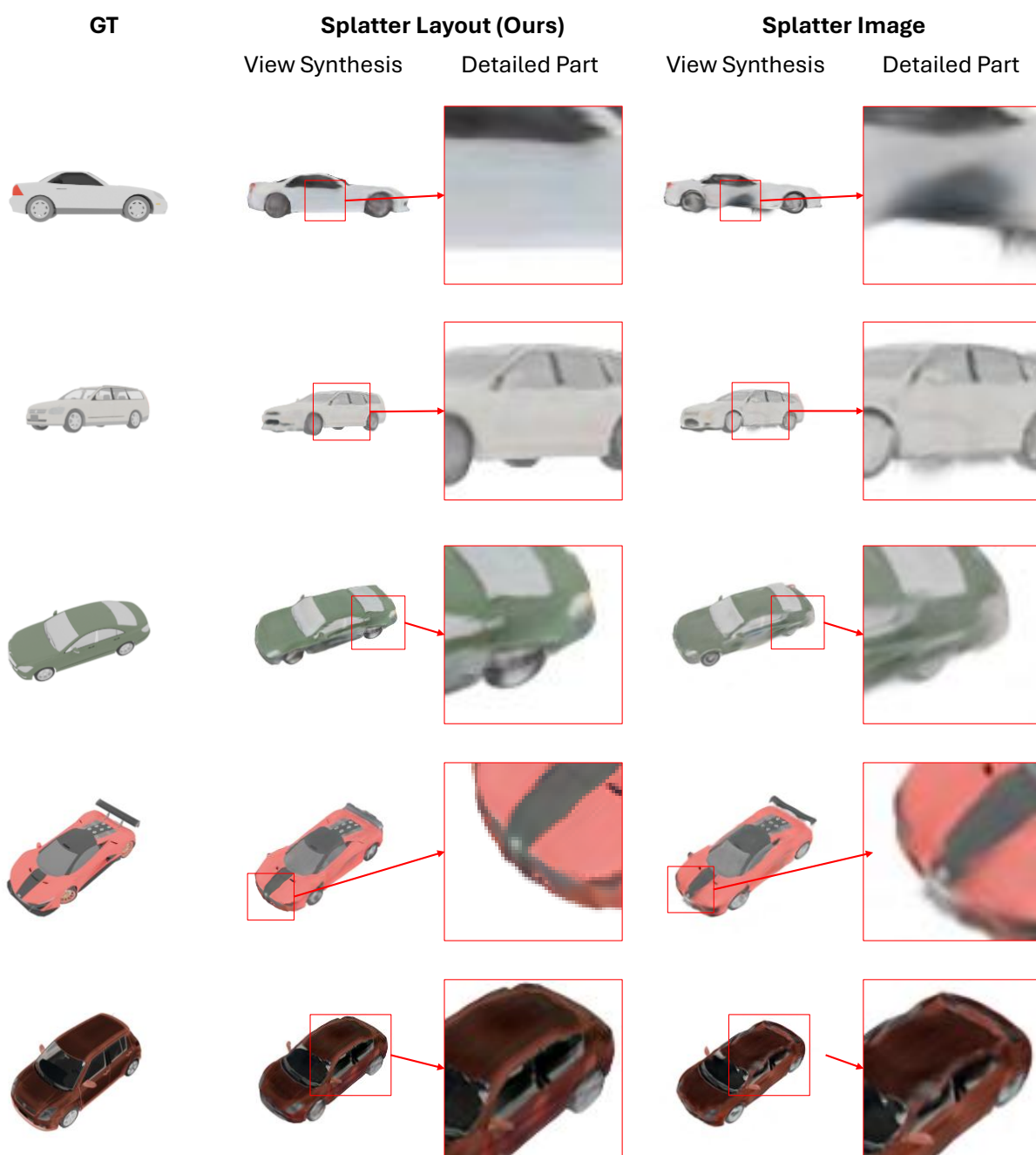


Figure 13. Qualitative comparison showcasing improved reconstruction of fine-scale details, particularly in near doors and wheels.

References

- [1] Bentley and Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, 1979. 4
- [2] Damian Boborzi, Phillip Mueller, Jonas Emrich, Dominik Schmid, Sebastian Mueller, and Lars Mikelsons. Meshfleet: Filtered and annotated 3d vehicle dataset for domain specific generative modeling. *arXiv preprint arXiv:2503.14002*, 2025. 4
- [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. 4
- [4] Xin Chen, Anqi Pang, Wei Yang, Peihao Wang, Lan Xu, and Jingyi Yu. Tightcap: 3d human shape capture with clothing tightness field. *ACM TOG*, 2021. 4
- [5] Nima Ghorbani and Michael J. Black. SOMA: Solving optical marker-based mocap automatically. In *ICCV*, 2021. 3
- [6] Mar Gonzalez-Franco, Eyal Ofek, Ye Pan, Angus Antley, Anthony Steed, Bernhard Spanlang, Antonella Maselli, Domna Banakou, Nuria Pelechano, Sergio Orts-Escolano, et al. The rocketbox library and the utility of freely available rigged avatars. *Frontiers in virtual reality*, 2020. 4
- [7] Hsuan-I Ho, Lixin Xue, Jie Song, and Otmar Hilliges. Learning locally editable virtual humans. In *CVPR*, 2023. 4
- [8] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014. 3
- [9] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv:1711.05101*, 2017. 1
- [10] Naureen Mahmood, Nima Ghorbani, Nikolaus F. Troje, Gerard Pons-Moll, and Michael J. Black. AMASS: Archive of motion capture as surface shapes. In *ICCV*, 2019. 3
- [11] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv:2307.01952*, 2023. 6
- [12] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv:2007.08501*, 2020. 1
- [13] Alla Sheffer, Bruno Lévy, Maxim Mogilnitsky, and Alexander Bogomyakov. Abf++: fast and robust angle based flattening. *ACM Transactions on Graphics (TOG)*, 24(2):311–330, 2005. 4
- [14] Stanislaw Szymanowicz, Christian Rupprecht, and Andrea Vedaldi. Splatter image: Ultra-fast single-view 3d reconstruction. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. 4
- [15] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *CVPR*, 2018. 1
- [16] Tao Yu, Zerong Zheng, Kaiwen Guo, Pengpeng Liu, Qionghai Dai, and Yebin Liu. Function4d: Real-time human volumetric capture from very sparse consumer rgbd sensors. In *CVPR*, 2021. 4
- [17] Wang Zeng, Wanli Ouyang, Ping Luo, Wentao Liu, and Xiaogang Wang. 3d human mesh regression with dense correspondence. In *CVPR*, 2020. 1, 4
- [18] Zerong Zheng, Tao Yu, Yebin Liu, and Qionghai Dai. Pamir: Parametric model-conditioned implicit representation for image-based human reconstruction. *TPAMI*, 2021. 4
- [19] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018. 2, 4, 6
- [20] Xingxing Zou, Xintong Han, and Waikeng Wong. Cloth4d: A dataset for clothed human reconstruction. In *CVPR*, 2023. 4