Millimetric Human Surface Capture in Minutes Supplementary Material

CCS Concepts: • **Computing methodologies** → **Computer vision**; **Motion capture**; **3D imaging**; **Image-based rendering**; **Image processing**.

ACM Reference Format:
. 2024. Millimetric Human Surface Capture in Minutes Supplementary Material. In *SIGGRAPH Asia 2024 Conference Papers (SA Conference Papers '24), December 3–6, 2024, Tokyo, Japan.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3680528.3687690

Contents

1 OPTIMIZATION AND VISUALIZATION LOOPS

A slightly simplified version of our actual optimization procedure is described in alg. 1, matching the diagram of the main paper. Function calls in red denote the invocation of a GPU kernel from the CPU. Some operations such as initializing or subdividing the grid, deleting or allocating voxels and computing the regularizations gradients require multiple kernel invocations that are not detailed here. We extract a mesh with marching cubes after each iteration for visualization purposes if needed. All kernel launches are fully asynchronous. Most of them run in strict launch order thanks to appropriate execution barriers (not shown here) while some of them are allowed to run concurrently when possible. For instance, the ADAM kernels may run simultaneously. A single CPU / GPU synchronization occurs at the end of each training iteration to monitor the loss, the number of extracted vertices and the number of active voxels and tiles. The photometric loss kernel is tasked to apply the color correction as well as to compute the loss and its per-pixel gradient. Note that we use no auto-differentiation, hard-coding partial derivatives instead for efficiency. Each forward pass kernel has a twin backward pass kernel, annotated with the "_backward" suffix.

ALGORITHM 1: Optimization loop

```
foreach level of detail do
    init_or_subdivide_grid() ;                  // [Several kernels]
    foreach training iteration do
        clear_grads() ;                         // [Several kernels]
        smooth_sdf() ;                           // [1 thread/voxel]
        compute_normals() ;                      // [1 thread/voxel]
        batch ← select_views()
        foreach camera ∈ batch do
            color_prediction() ;                 // [1 thread/voxel]
            ray_marching() ;                     // [1 thread/pixel]
            photometric_loss() ;                 // [1 thread/pixel]
            ray_marching_backward() ;            // [1 thread/pixel]
            color_prediction_backward() ;        // [1 thread/voxel]
        end
        apply_regularizations() ;               // [Several kernels]
        compute_normals_backward() ;             // [1 thread/voxel]
        smooth_sdf_backward() ;                  // [1 thread/voxel]
        adam_SDF() ;                             // [1 thread/voxel]
        adam_Features() ;                        // [1 thread/texel]
        adam_MLP() ;                            // [1 thread/weight]
        delete_or_alloc_voxels() ;              // [Several kernels]
        marching_cubes() ;                       // [1 thread/voxel]
    end
end
```

After training is complete, we can visualize the results interactively according to alg. 2. We start by uploading the data to the GPU memory, then apply the smoothing kernel on the sdf and pre-compute the normal vectors. Afterwards, only two kernel invocations are necessary in the rendering loop, first to predict the per-voxel colors from the current viewpoint, and then to compute the volume rendering. The color correction is applied if the user wants to view the result from a particular camera's lens.

2 SDF TO TRANSPARENCY CONVERSION

We can slightly reformulate the rendering equation proposed by [Wang et al. 2021] since we use the opposite convention where α_i denotes the transparency of a sample instead of its opacity.

$$\alpha_{\text{Ours}} = 1 - \alpha_{\text{Neus}} \tag{1}$$

$\Phi_s(x) = \frac{1}{1+exp(-sx)}$ is the sigmoid function with scale parameter s and f_{i-1} and f_i are two consecutive signed-distance values sampled

SA Conference Papers '24, December 3–6, 2024, Tokyo, Japan

ACM ISBN 979-8-4007-1131-2/24/12.
https://doi.org/10.1145/3680528.3687690

ALGORITHM 2: Visualization loop

```
load_data()
smooth_sdf() ;                          // [1 thread/voxel]
compute_normals() ;                     // [1 thread/voxel]
while true do
    camera ← current_view()
    color_prediction() ;                // [1 thread/voxel]
    ray_marching() ;                    // [1 thread/pixel]
    color_correction() ;                // [1 thread/pixel]
    display();
end
```

along a ray.

$$\alpha_i = 1 - \max\left(\frac{\Phi_s(f_{i-1}) - \Phi_s(f_i)}{\Phi_s(f_{i-1})}, 0\right) \tag{2}$$

$$= 1 + \min\left(\frac{\Phi_s(f_i) - \Phi_s(f_{i-1})}{\Phi_s(f_{i-1})}, 0\right) \tag{3}$$

$$= \min\left(\frac{\Phi_s(f_i)}{\Phi_s(f_{i-1})}, 1\right) \tag{4}$$

$$= \min\left(\frac{1+\exp(-sf_{i-1})}{1+\exp(-sf_i)}, 1\right) \tag{5}$$

Neus uses the colors generated in-between the sdf samples $c_{i-1/2}$. Instead, we chose to use the color of the current sampled point c_i for greater efficiency. The reasoning is that we can fetch the color and the sdf values as a 4-tuple from an rgba texture in a single operation, which divides by two the number of fetches compared to an implementation following Neus.

3 ALLOCATION BOUNDS

The allocation bounds $\tau_{\min}$ and $\tau_{\max}$ used to cull or instantiate voxels are derived from the s-parameter of the rendering equation eq. (2) such that the transmittance T of a ray is guaranteed to fall below a given threshold η before exiting the band of allocated voxels. The derivation in eq. (6) assumes the sdf decreases monotonously along the ray. If not, it means the surface part was too thin and the ray will exit with a transmittance $T > \eta$, and will possibly intersect another part of the surface.

$$T_N = \prod_{i<N} \alpha_i = \prod_{i<N} \frac{1+\exp(-sf_{i-1})}{1+\exp(-sf_i)} = \frac{1+\exp(-s\tau_{\max})}{1+\exp(+s\tau_{\max})} < \eta \tag{6}$$

$$\iff \eta \exp(s\tau_{\max}) > 1 \tag{7}$$

$$\iff \tau_{\max} > -\log(\eta)/s \tag{8}$$

In practice, we use $\tau_{\min} = 0.8\tau_{\max}$ with $\tau_{\max} = -1.5\log(\eta)/s$, slightly larger than the theoretical bound because more voxels are needed for interpolation.

4 ADAPTIVE STEPPING

It has been shown that sampling the transmittance uniformly is beneficial for performance [Li et al. 2023]. We can do so very easily since the sdf is encoded explicitly in the voxel grid. The basic idea is to compute a step size Δt such that the transmittance at the next sample will decrease by a fixed amount T_{step}. Starting from the known sdf value of the last sample f_{i-1}, we compute an expected value $\hat{f}_i$ in eq. (11) that would achieve the desired decrease in transmittance.

$$T_{\text{step}} = T_i - T_{i+1} = T_i - \alpha_i T_i \tag{9}$$

$$\implies \alpha_i = 1 - T_{\text{step}}/T_i = \frac{1+\exp(-sf_{i-1})}{1+\exp(-s\hat{f}_i)} \tag{10}$$

$$\implies \hat{f}_i = \frac{-1}{s}\log\left(\frac{1+\exp(-sf_{i-1})}{1-T_{\text{step}}/T_i} - 1\right) \tag{11}$$

We can then use a first order approximation of the sdf to compute the step Δt based on $\hat{f}_i$ in eq. (12).

$$f(\mathbf{X} + \Delta t\mathbf{V}) \approx f(\mathbf{X}) + \frac{\partial}{\partial t} f(\mathbf{X} + t\mathbf{V})|_{t=0}\Delta t \tag{12}$$

$$= f(\mathbf{X}) + \nabla f(\mathbf{X}) \cdot \mathbf{V}\Delta t \tag{13}$$

$$\implies \hat{f}_i = f_{i-1} + \nabla f(\mathbf{X}) \cdot \mathbf{V}\Delta t \tag{14}$$

$$= f_{i-1} + (\mathbf{N} \cdot \mathbf{V})\Delta t \tag{15}$$

$$\implies \Delta t = \frac{\hat{f}_i - f_{i-1}}{\mathbf{N} \cdot \mathbf{V}} \tag{16}$$

We further approximate $\mathbf{N} \cdot \mathbf{V}$ as $(f_{i-1} - f_{i-2})/\Delta t_{i-1}$ in the previous equation. Using this method, each ray will advance independently by a step tailored to the local behavior of the surface at the current sample point. In practice, we restrict $0.1 < \Delta t < 0.5$ in units of a voxel size.

5 ADDITIONAL EXPERIMENTAL DETAILS

We run all experiments on a linux workstation equipped with an Nvidia RTX A6000, unless stated otherwise. We use all images available both at training and test time. In the following, we provide additional details specific to each method used for the comparison in the main paper. The resulting mesh of the baselines reconstructing a complete scene (Colmap, RealityCapture, 2DGS and GOF) is cropped with a bounding box before evaluation.

Ours We fit a bounding box on the complete acquisition volume and compute a coarse visual hull to find the region of interest and initialize the volume. Visual hull computation and initialization takes less than 1 second.

Neus2 Neus2 provides two means of accounting for varying camera exposure: an extra latent code and an exposure parameter that are trained on a per-camera basis. We enable both in an attempt to match our optimization of a per-camera color curve. We train Neus2 for 7500 iterations using the DTU configuration. We translate and scale the cameras to place the scene to be reconstructed inside the unit ball.

Voxurf We use the DTU configuration files with masks setting. Voxurf starts training with a coarse initialization based on DVGO. They extract a bounding box from this reconstruction before running their coarse and fine optimization schemes. We use the default calibration since Voxurf can automatically find the region of interest.

Colmap We run colmap using the cameras' calibration and disable the optimization of the cameras' parameters. We run patchmatch stereo on full size images using the geometric consistency setting. The depthmaps are fused in a dense point cloud then meshed using the Poisson mesher.

RealityCapture We run RealityCapture on a windows laptop since linux is not supported. We let it estimate the calibration data since we could not find a simple way to provide calibrated inputs.

Gaussian splatting methods The sparse point cloud resulting from the triangulation of the features detected by colmap is given to 3DGS, GOF and 2DGS. We train them on full size 2048^2 images instead of a re-scaled version that is used by default when the images exceed a width of 1600 pixels. We train for the default 30k iterations. We supply the "decoupled appearance" option to GOF since our images contain inter-camera color variations that are not specifically handled for the other two. We train 2DGS with the median depth setting instead of the average depth since it otherwise fails to extract a unique surface with its TSDF fusion scheme. The data loader of 3DGS (and inherited by 2DGS and GOF) silently discards the principal point from the calibration. We changed the code of the three baselines to take into account our non-centered cameras. This only amounts to changing the projection matrix for 3DGS and 2DGS as noted by other contributors but these changes are not part of the main branches yet. After discussion with the authors, the principal point must additionally be forwarded to several CUDA kernels in GOF's case. We opened a pull request so that others may benefit from this improvement. We observe that 3DGS converges reliably whereas 2DGS and GOF have more trouble in some instances (see fig. 1) in which a few gaussians occlude the scene from one or several viewpoints, lowering the average PSNR. In terms of geometric quality, 2DGS creates noisy polygons attached to the inside of the outer surface in the highly textured regions, as visible in fig. 2 (Top). These polygons are not removed when we select the main connected component since they are attached. GOF extracts a mesh on an irregular tetrahedral grid which limits the impact on the accuracy metric due to the larger triangles in the noisy areas, even though it can fail more catastrophically. We also demonstrate a smoother and more detailed surface than both 2DGS and GOF on an example with low frequency texture (See fig. 2 (Bottom)). GOF also fails to extract the feet in this specific instance.

Fig. 1. Convergence failures for gaussian opacity fields (left) due to gaussians near the cameras occluding the scene, and for 2D gaussian splatting (right) due to gaussians above the ground occluding the feet.

6 DATA STRUCTURES COMPARISON

A wide range of scene representations have been used in the literature, this short description will attempt to compare their most prominent features.

(1) **Octrees** [Liu et al. 2020] and [Yu et al. 2021] fit an octree on an already optimized NeRF. Octrees present ideal sparsity but are complex to update. Furthermore, deep hierarchical structures imply many levels of indirection that can hurt rendering performance since the pointer stack already requires many registers [Laine and Karras 2011].
(2) **Dense grids** [Sun et al. 2022; Wu et al. 2023] simply use dense grids that cannot scale beyond resolutions of a few hundred voxels, but can be very efficient at small resolutions.
(3) **Factorized grids** [Cao and Johnson 2023; Chen et al. 2022; Fridovich-Keil et al. 2023] factorize a dense grid into outer products of global planes or vectors which removes the memory bottleneck of a dense grid but prevents local updates. [Gao et al. 2023] improves significantly over global factorization methods in terms of quality and model size by adopting local factorizations. However, it lacks a proper acceleration structure to allow for fast inference.
(4) **Hash grids** [Müller et al. 2022; Wang et al. 2023] rely on hierarchical hash grids. An evolving occupancy grid restricts the sampling to non-empty regions so that the hash tables only need to be fitted close to the surface. Though practical, this strategy presents several downsides. First, each hash table has a fixed and predetermined size which means that the number of parameters cannot be increased dynamically to accommodate larger reconstructions. Second, the hash tables inherently hinder cache coherence which creates a sharp drop in performance when their size exceeds that of the GPU cache [Müller et al. 2022].

7 OUTPUT MODEL SIZE

Our trained model size is roughly 100MB in total but about half of the allocated parameters are kept unused. This is the consequence of our sparse structure storing voxels in tiles of 4^3. We compress the model as zip archive and obtain a final file size of about 50MB. Linearizing the datastructure to remove unused voxels and features would be another way of achieving the same effect.

8 NEURAL ARCHITECTURE

Our network architecture is a small MLP with 32 neurons and 2 hidden layers. We use features with a dimensionality of 8 in all experiments and a spherical harmonic embedding of the reflected vector with 16 coefficients. We use ReLU activations with a sigmoid in the last layer. For comparison, NeuS2 uses a 2 hidden layers MLP with 64 neurons and Voxurf uses a 4 hidden layers MLP with 192 neurons, requiring a much greater number of floating-point operations.

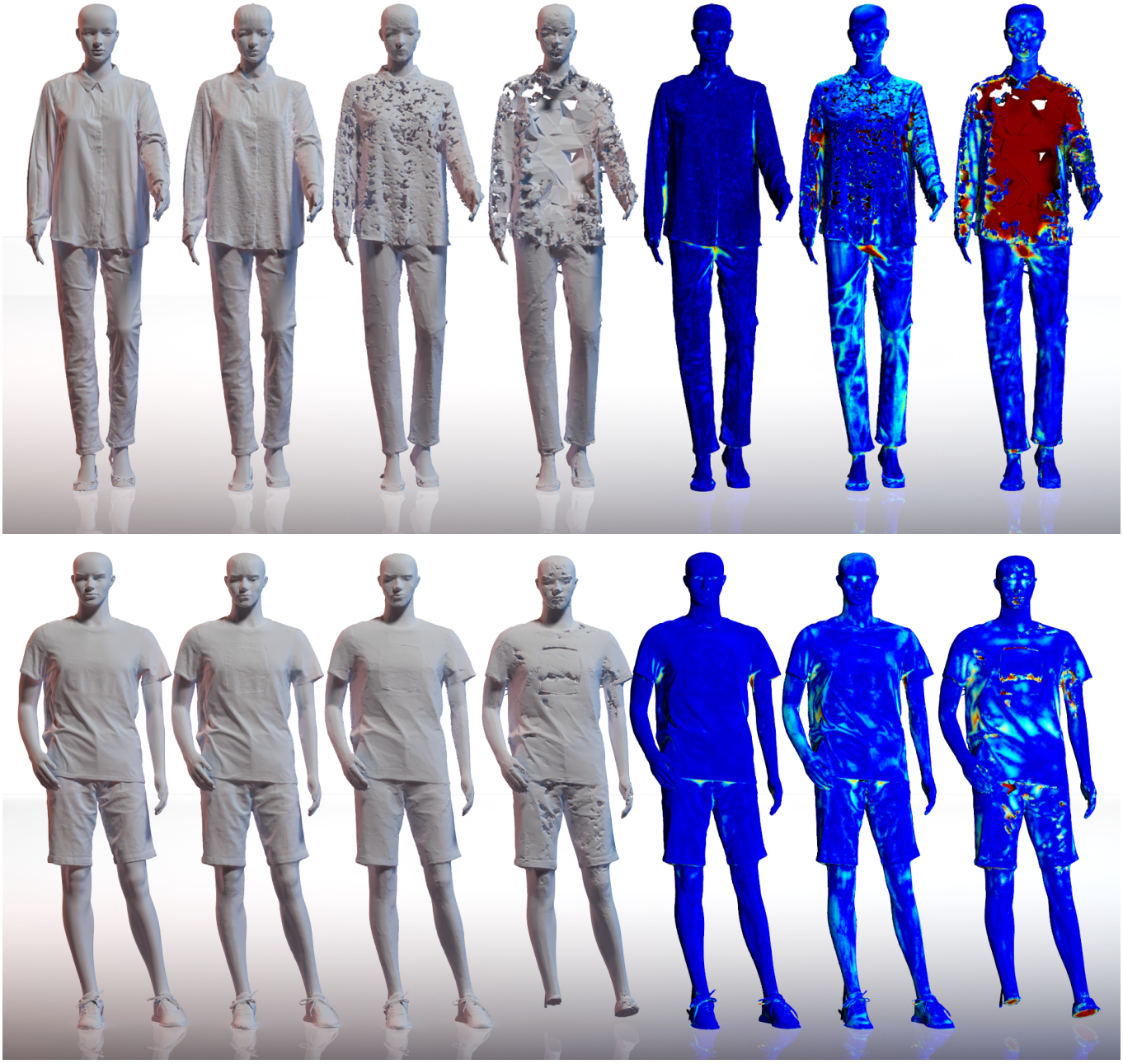

Fig. 2. Top: Female "jea" scene, bottom: Male "sho" scene. From left to right: Reference scan, Our reconstruction, 2DGS, GOF and their associated accuracy heatmaps. Top: Highly textured areas are not well reconstructed both for 2DGS and GOF, leading to noise and holes. Bottom: We achieve better or equivalent geometric quality than 2DGS and GOF even in regions with less texture.

In terms of grid resolution, we use voxels with 2mm side length at the highest level of detail, which translates to about 10 million active voxels. In contrast, Voxurf uses 96^3 and 256^3 dense grids at the coarse and fine reconstruction stages respectively, corresponding to voxels of about 8mm at the fine stage assuming a bounding box of 2m. NeuS2 use 14 hash grid levels spanning a space of size 16^3 up to 2048^3 which gives voxels of 1mm with the same bounding box of

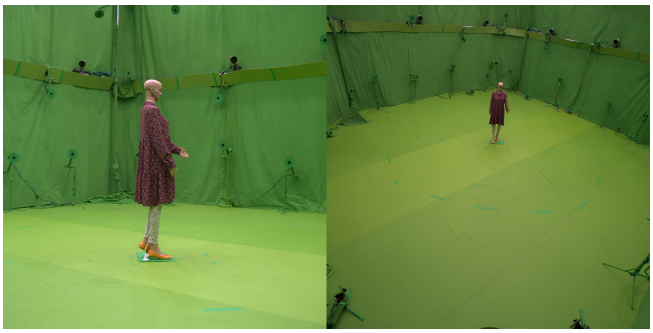

Fig. 3. Field of view differences imply exposure variations between the cameras.

Fig. 4. The camera sensors display some level of noise (left) that can be alleviated by temporal averaging (right) but color fringing and staircase artifacts remain due to the Bayer filter.

2m. However, each hash grid level only contains $2^{19} \approx 500K$ entries, necessarily leading to a large number of collisions in the last level.

9 EXPOSURE VARIATIONS AND SENSOR NOISE

The acquisition platform used mixes large and medium camera fields of view leading to changes in exposure, as visible in fig. 3. Figure 4 shows closeups on the sensor noise and Bayer filter artifacts in the input images and fig. 5 provides a qualitative comparison of the color smoothness regularizer.

10 DETAILED QUANTITATIVE RESULTS

Tables 1, 2 and 3 provide complete metrics for the chosen baselines on all the scenes of our dataset. Figure 6 shows the error curves corresponding to the ablations of the main paper.

11 VOLUME RENDERINGS

Figures 7 through 21 show the volume rendering results of all the baselines on all the scenes of our dataset, in the same order as the tables 1, 2 and 3. See figures 7, 12 and 19 for closeups and additional commentary.

Fig. 5. Our reconstruction without the color smoothness regularizer (left) and with (right). The region around the eyes exhibits a noticeable pixelizing.

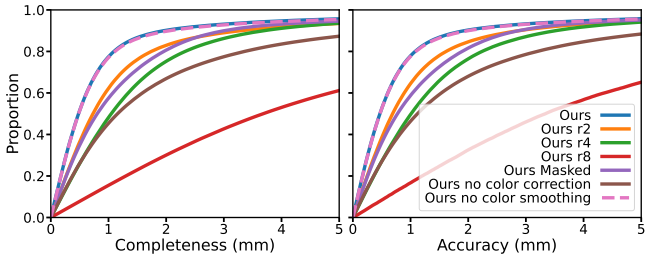

Fig. 6. Error curves for the ablations presented in the main paper.

REFERENCES

Ang Cao and Justin Johnson. 2023. HexPlane: A Fast Representation for Dynamic Scenes. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 130–141. https://doi.org/10.1109/cvpr52729.2023.00021

Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. 2022. TensoRF: Tensorial Radiance Fields. In *European Conference on Computer Vision*. Springer, Springer Nature Switzerland, 333–350. https://doi.org/10.1007/978-3-031-19824-3_20

Sara Fridovich-Keil, Giacomo Meanti, Frederik Rahbæk Warburg, Benjamin Recht, and Angjoo Kanazawa. 2023. K-Planes: Explicit Radiance Fields in Space, Time, and Appearance. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 12479–12488. https://doi.org/10.1109/cvpr52729.2023.01201

Quankai Gao, Qiangeng Xu, Hao Su, Ulrich Neumann, and Zexiang Xu. 2023. Strivec: Sparse Tri-Vector Radiance Fields. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, 17569–17579. https://doi.org/10.1109/iccv51070.2023.01611

Samuli Laine and Tero Karras. 2011. Efficient Sparse Voxel Octrees. *IEEE Transactions on Visualization and Computer Graphics* 17, 8 (Aug. 2011), 1048–1059. https://doi.org/10.1109/TVCG.2010.240

Ruilong Li, Hang Gao, Matthew Tancik, and Angjoo Kanazawa. 2023. NerfAcc: Efficient Sampling Accelerates NeRFs. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE. https://doi.org/10.1109/iccv51070.2023.01699

Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. 2020. Neural Sparse Voxel Fields. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/b4b758962f17808746e9bb832a6fa4b8-Abstract.html

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics* 41, 4 (July 2022), 1–15. https://doi.org/10.1145/3528223.3530127

Cheng Sun, Min Sun, and Hwann-Tzong Chen. 2022. Direct Voxel Grid Optimization: Super-fast Convergence for Radiance Fields Reconstruction. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 5459–5469. https://doi.org/10.1109/cvpr52688.2022.00538

Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. 2021. NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, Marc'Aurelio Ranzato,

	female								male					
Completeness (mm)	cos	naked	jea	opt1	opt2	opt3	sho	tig	cos	naked	jea	opt	sho	tig
Ours	1.69	**0.53**	**1.22**	**1.14**	2.16	1.45	**0.98**	**0.61**	**1.99**	**0.50**	**0.82**	**1.76**	**0.73**	**0.70**
Ours Masked	2.05	1.50	1.62	1.33	2.23	1.46	1.41	0.88	3.54	1.52	1.06	2.55	1.14	1.17
Voxurf	**1.44**	1.27	1.63	1.39	**1.89**	**1.35**	1.27	0.89	2.57	1.21	1.25	2.79	1.05	1.00
Neus2	5.18	1.47	2.59	2.47	3.44	2.41	2.22	1.28	4.48	1.48	1.89	3.06	1.66	1.50
RealityCapture	3.74	6.16	6.07	6.47	5.77	3.47	4.68	3.94	7.04	4.89	2.49	5.57	3.83	4.47
Colmap	3.25	3.95	3.34	5.17	4.54	2.97	3.63	2.24	4.68	3.69	2.36	5.54	2.99	3.03
2DGS	3.82	2.12	2.63	2.13	2.55	2.35	2.02	1.56	3.69	2.09	1.38	2.39	1.72	1.64
GOF	5.86	1.45	7.01	2.29	2.88	4.34	2.47	4.14	2.74	1.42	2.24	2.53	4.77	1.61

Table 1. Per-scene average completeness (mm)

	female								male					
Accuracy (mm)	cos	naked	jea	opt1	opt2	opt3	sho	tig	cos	naked	jea	opt	sho	tig
Ours	**1.41**	**0.55**	**0.99**	**1.17**	2.08	**1.45**	**1.17**	**0.75**	**1.55**	**0.68**	**0.79**	**1.67**	**0.78**	**0.80**
Ours Masked	1.53	1.47	1.19	1.31	2.11	1.57	1.50	0.97	2.86	1.66	0.97	2.13	1.14	1.21
Voxurf	1.96	1.29	1.61	1.84	2.24	1.87	1.68	1.08	2.62	1.39	1.29	2.29	1.20	1.12
Neus2	2.98	1.44	1.74	2.11	2.68	2.14	1.97	1.23	2.85	1.63	1.49	2.29	1.53	1.44
RealityCapture	1.93	5.03	4.83	4.86	4.07	2.33	3.77	3.31	4.51	4.36	1.98	3.05	3.15	3.72
Colmap	2.12	4.59	2.65	4.01	3.83	2.65	3.78	2.77	3.54	4.72	2.60	3.14	3.18	3.43
2DGS	6.29	2.42	13.98	4.37	4.25	4.90	3.31	5.47	4.67	2.40	2.12	3.21	2.23	2.10
GOF	5.32	1.30	3.10	1.88	**2.05**	4.40	1.86	5.44	1.99	1.47	4.68	2.39	1.67	1.77

Table 2. Per-scene average accuracy (mm)

	female								male					
PSNR (db)	cos	naked	jea	opt1	opt2	opt3	sho	tig	cos	naked	jea	opt	sho	tig
Ours	29.24	**40.03**	**30.55**	**39.94**	38.01	**33.82**	37.18	**31.38**	**39.22**	**40.88**	35.25	38.59	37.11	37.49
Ours Masked	**29.31**	39.6	30.54	39.63	37.8	33.69	37.0	31.29	39.03	40.5	35.1	38.3	36.9	37.3
Voxurf	28.4	37.5	30.2	39.3	37.3	32.8	36.7	30.8	38.2	40.1	34.5	37.6	36.7	37.2
Neus2	27.6	34.0	28.6	31.9	33.2	30.3	31.6	28.9	31.3	33.2	29.7	25.8	32.4	32.7
3DGS	28.1	39.91	29.7	39.4	**38.28**	33.2	**37.42**	30.6	38.5	40.62	**36.10**	**38.60**	**37.47**	**37.79**
2DGS	27.3	37.9	28.9	37.6	36.8	32.2	36.2	29.9	38.0	38.8	35.0	37.3	36.1	36.5
GOF	27.1	37.2	28.1	37.1	36.3	31.5	35.3	28.8	37.2	38.5	33.4	35.9	35.3	34.9

Table 3. Per-scene average PSNR

Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 27171–27183. https://proceedings.neurips.cc/paper/2021/hash/e41e164f7485ec4a28741a2d0ea41c74-Abstract.html

Yiming Wang, Qin Han, Marc Habermann, Kostas Daniilidis, Christian Theobalt, and Lingjie Liu. 2023. NeuS2: Fast Learning of Neural Implicit Surfaces for Multi-view Reconstruction. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, 3295–3306. https://doi.org/10.1109/iccv51070.2023.00305

Tong Wu, Jiaqi Wang, Xingang Pan, Xudong Xu, Christian Theobalt, Ziwei Liu, and Dahua Lin. 2023. Voxurf: Voxel-based Efficient and Accurate Neural Surface Reconstruction. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. https://openreview.net/pdf?id=DSy8tP4WctmZ

Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. 2021. PlenOctrees for Real-time Rendering of Neural Radiance Fields. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, 5752–5761. https://doi.org/10.1109/iccv48922.2021.00570

Fig. 7. Qualitative rendering comparisons on the female "cos" scene. Left to right: input image, Ours, Voxurf, Neus2, 3DGS, 2DGS, GOF. The gaussian splatting methods achieve a good image quality overall but tend to fit highly anisotropic gaussians in textured regions, which impacts the extracted geometry in 2DGS and GOF's cases. Neus2 suffers from noise while Voxurf creates sharp boundaries at depth discontinuities. Our method reconstructs both smooth edges that accurately blend with the background image and detailed textured patterns.

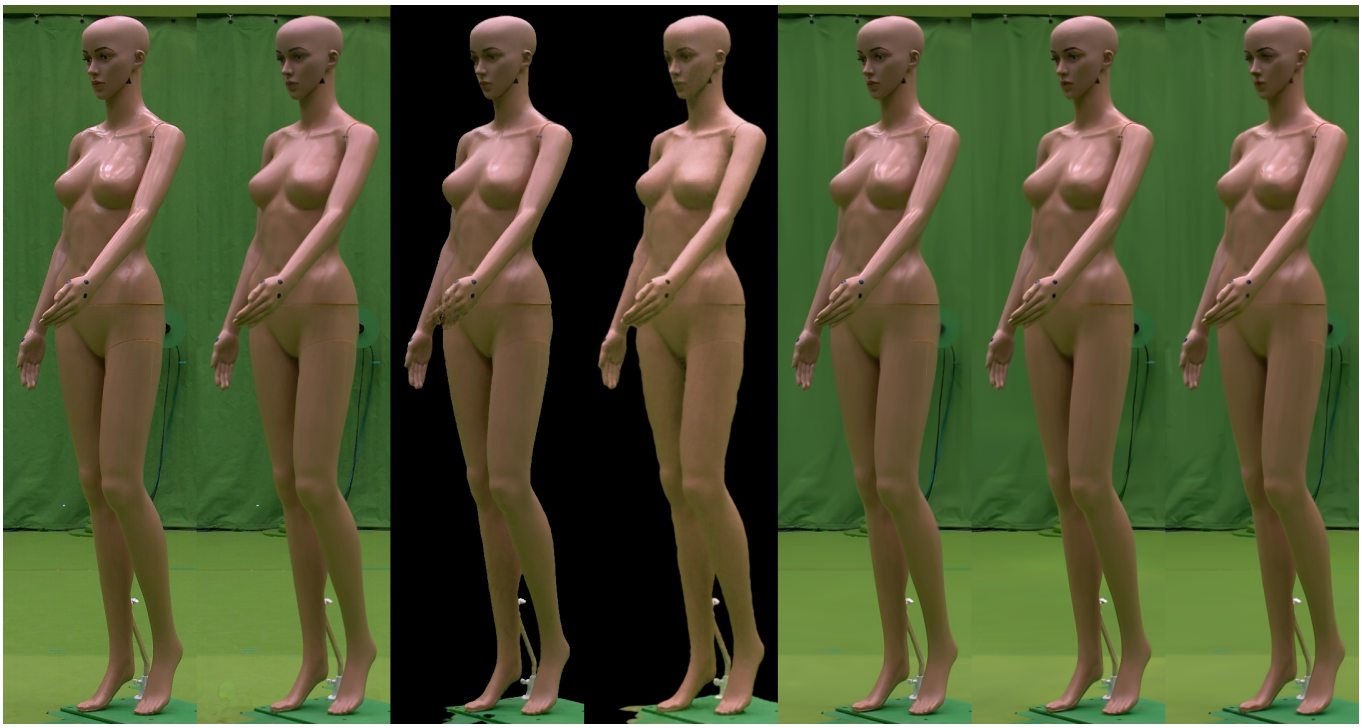

Fig. 8. Qualitative rendering comparisons on the female "naked" scene. Left to right: input image, Ours, Voxurf, Neus2, 3DGS, 2DGS, GOF.

Fig. 9. Qualitative rendering comparisons on the female "jea" scene. Left to right: input image, Ours, Voxurf, Neus2, 3DGS, 2DGS, GOF.

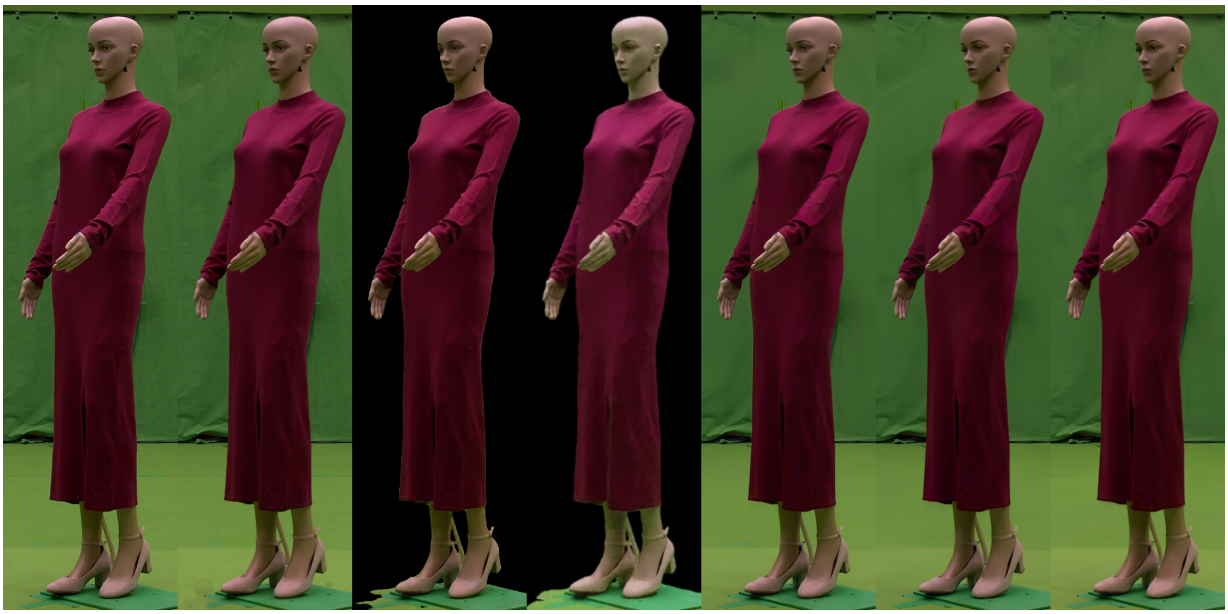

Fig. 10. Qualitative rendering comparisons on the female "opt1" scene. Left to right: input image, Ours, Voxurf, Neus2, 3DGS, 2DGS, GOF.

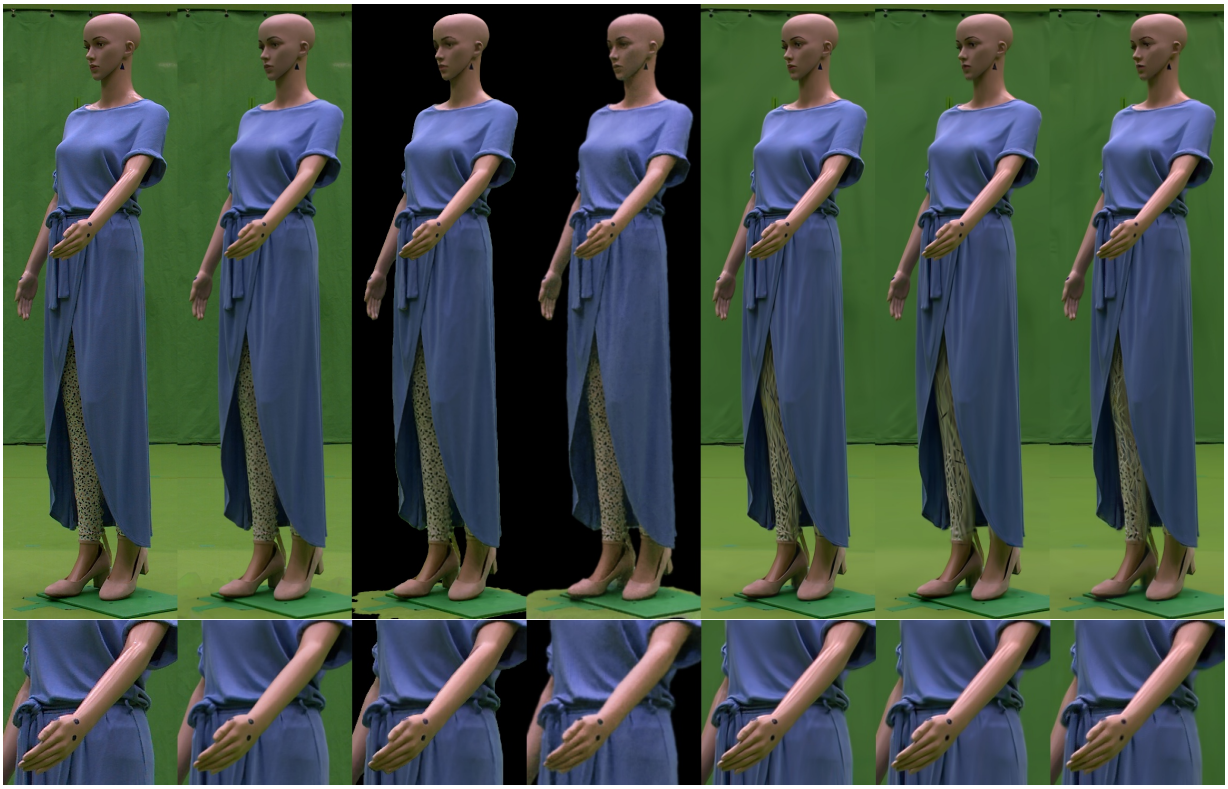

Fig. 11. Qualitative rendering comparisons on the female "opt2" scene. Left to right: input image, Ours, Voxurf, Neus2, 3DGS, 2DGS, GOF. The splatting based methods are particularly good at reconstructing high frequency specularities compared to the neural methods, with Voxurf having a noticeable advantage in that regard thanks to its larger MLP.

Fig. 12

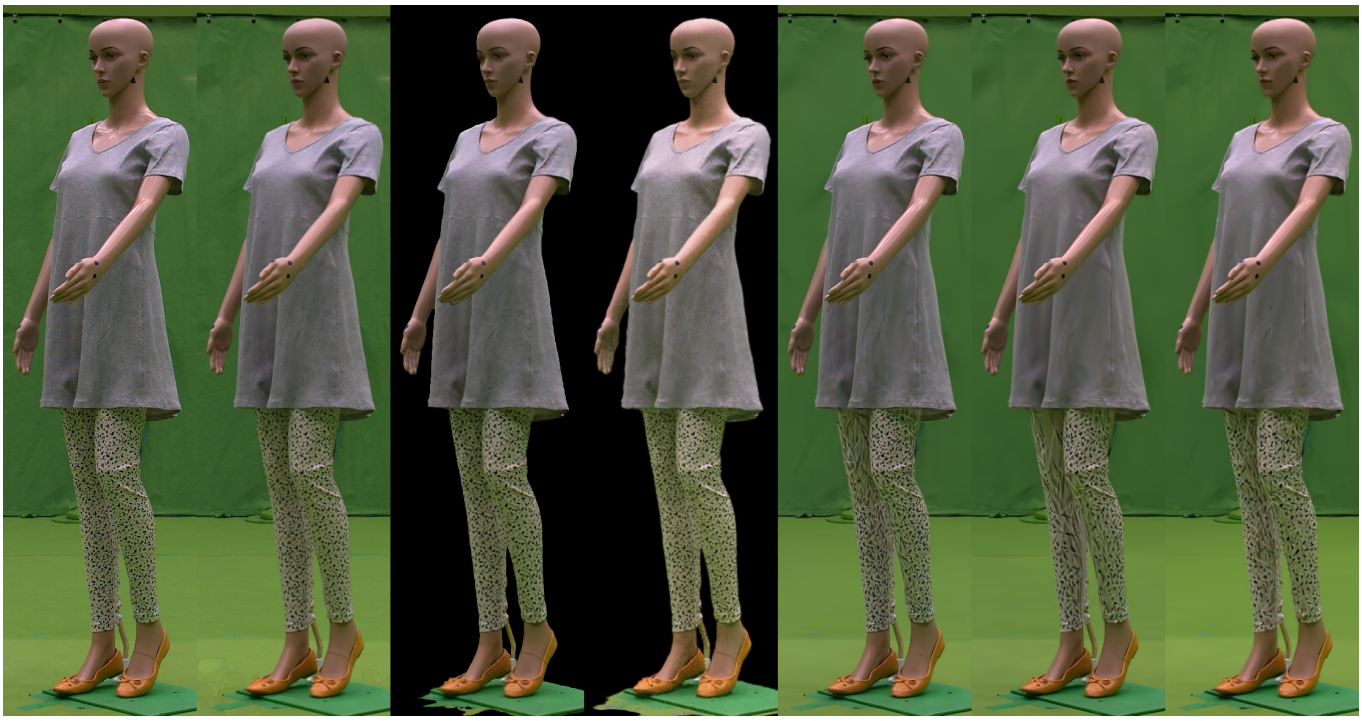

Fig. 13. Qualitative rendering comparisons on the female "opt3" scene. Left to right: input image, Ours, Voxurf, Neus2, 3DGS, 2DGS, GOF.

Fig. 14. Qualitative rendering comparisons on the female "sho" scene. Left to right: input image, Ours, Voxurf, Neus2, 3DGS, 2DGS, GOF

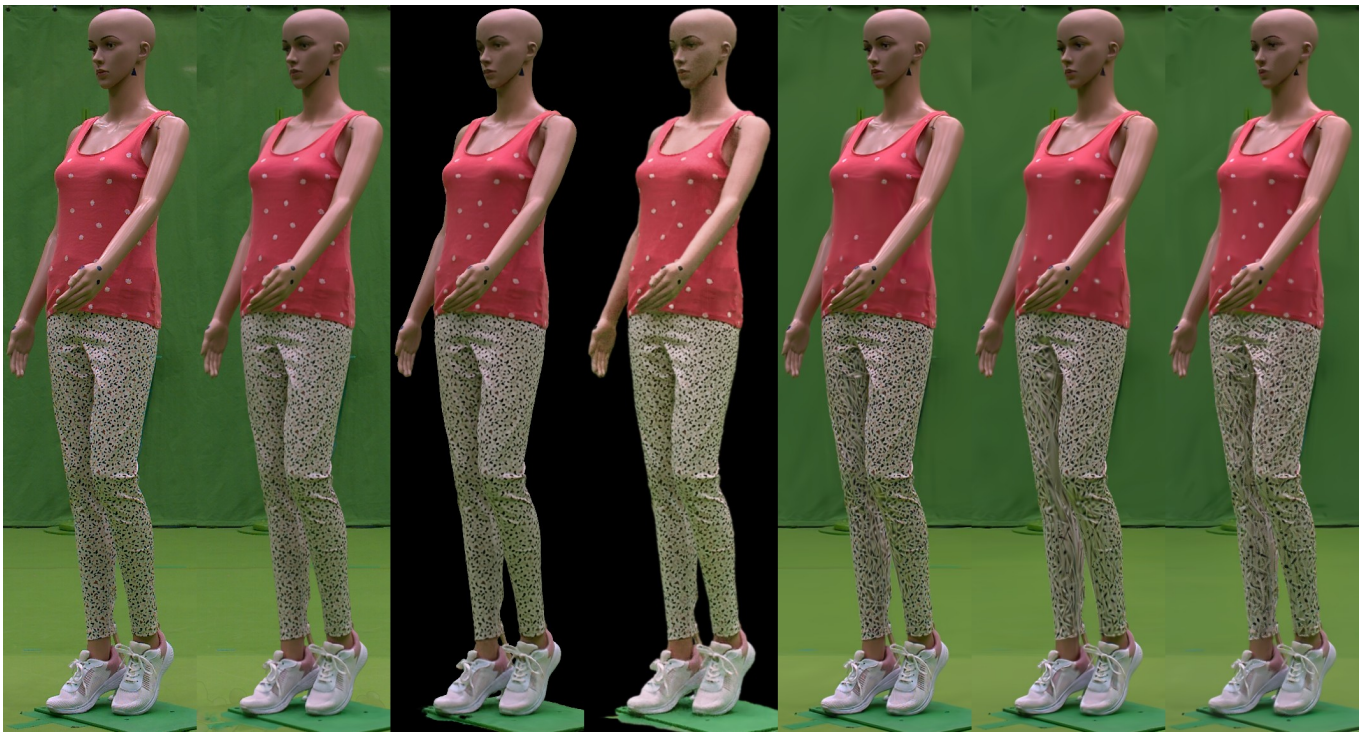

Fig. 15. Qualitative rendering comparisons on the female "tig" scene. Left to right: input image, Ours, Voxurf, Neus2, 3DGS, 2DGS, GOF.

Fig. 16. Qualitative rendering comparisons on the male "cos" scene. Left to right: input image, Ours, Voxurf, Neus2, 3DGS, 2DGS, GOF.

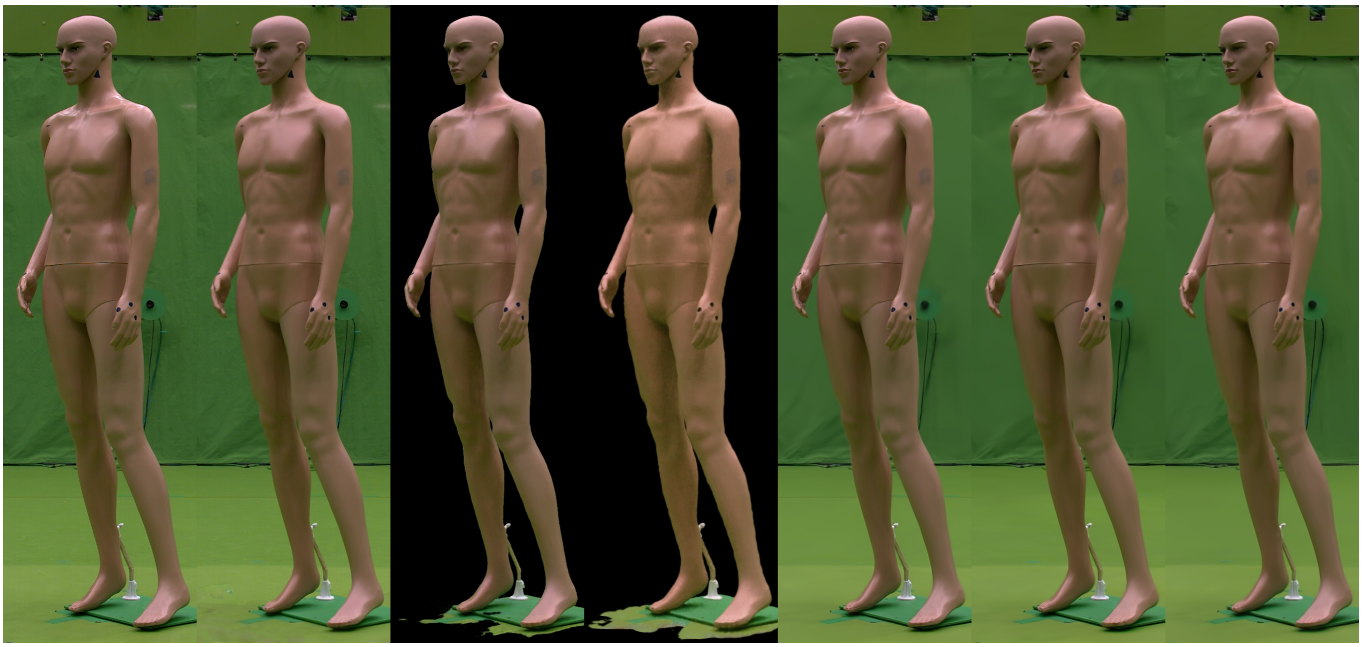

Fig. 17. Qualitative rendering comparisons on the male "naked" scene. Left to right: input image, Ours, Voxurf, Neus2, 3DGS, 2DGS, GOF.

Fig. 18. Qualitative rendering comparisons on the male "jea" scene. Left to right: input image, Ours, Voxurf, Neus2, 3DGS, 2DGS, GOF.

Fig. 19. Qualitative rendering comparisons on the male "opt" scene. Left to right: input image, Ours, Voxurf, Neus2, 3DGS, 2DGS, GOF. We observe that NeuS2 is sometimes unable to explain the inter-camera color variations from some viewpoints whereas the other baselines do not seem impacted.

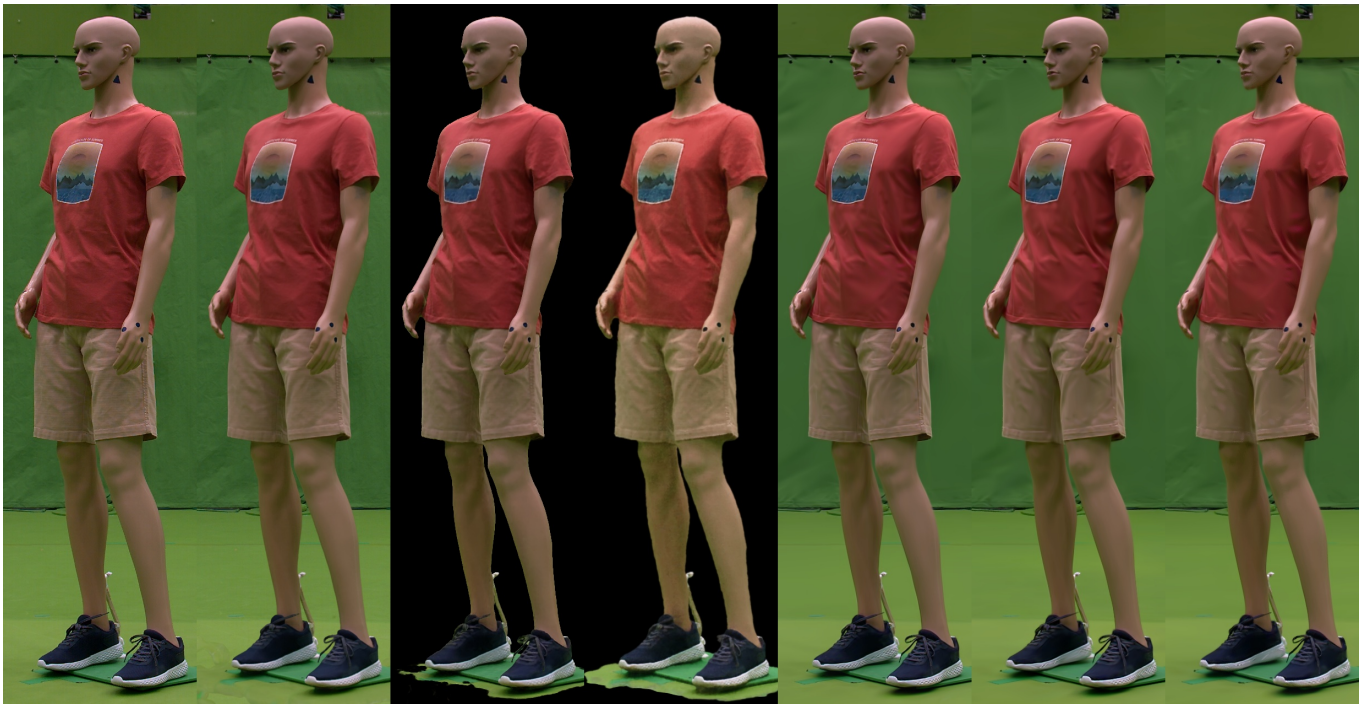

Fig. 20. Qualitative rendering comparisons on the male "sho" scene. Left to right: input image, Ours, Voxurf, Neus2, 3DGS, 2DGS, GOF.

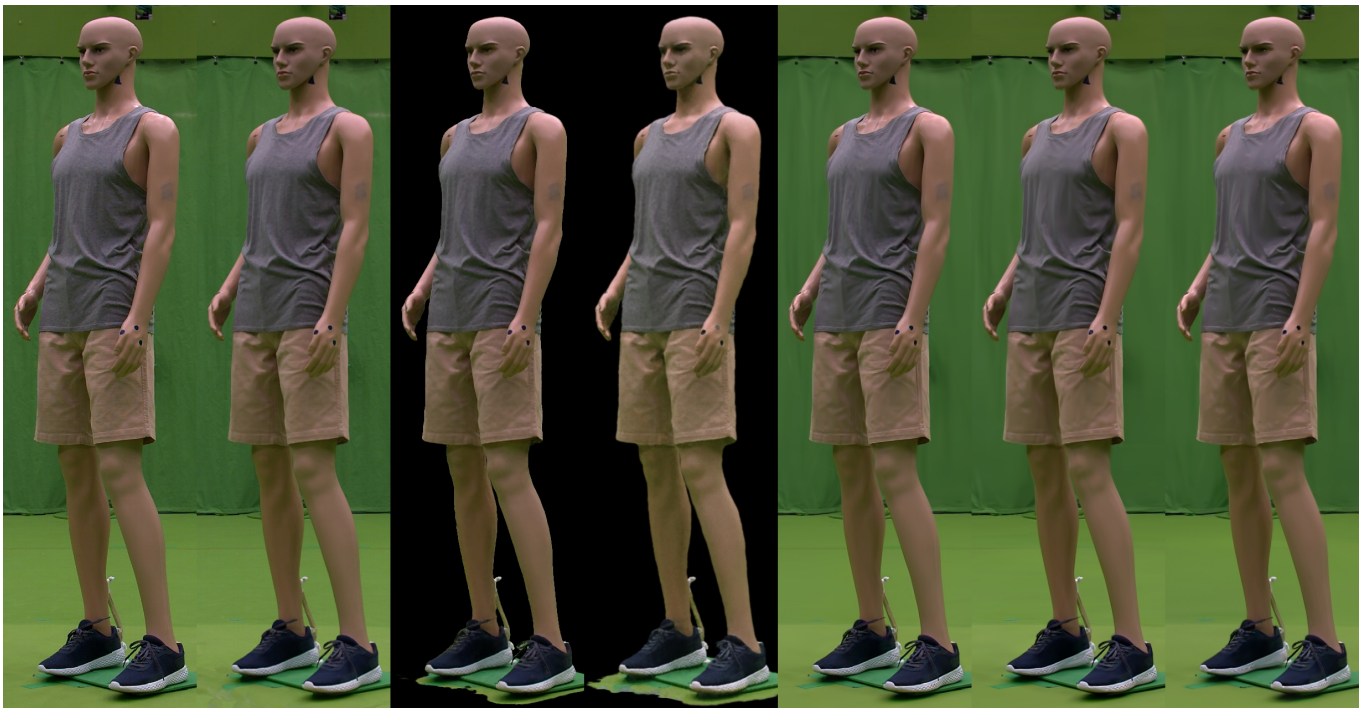

Fig. 21. Qualitative rendering comparisons on the male "tig" scene. Left to right: input image, Ours, Voxurf, Neus2, 3DGS, 2DGS, GOF.