Supplementary Materials: Sparse-View 3D Reconstruction of Clothed Humans via Normal Maps

A. Ray-Tracing the Implicit Surface Directly

As an alternative to Marching Tetrahedra, consider casting a ray to find an intersection point with the implicit surface and subsequently using the normal vector defined (directly) by the implicit surface at that intersection point. A number of existing works consider such approaches in various ways, see e.g. [2, 3, 5, 7, 8]. Perturbations of the intersection point depend on perturbations of the ϕ values on the vertices of the tetrahedron that the intersection point lies within. If a change in ϕ values causes the intersection point to no longer be contained inside the tetrahedron, one would need to discontinuously jump to some other tetrahedron (which could be quite far away, if it even exists). A potential remedy for this would be to define a virtual implicit surface that extends out of the tetrahedron in a way that provides some sort of continuity (especially along silhouette boundaries).

Comparatively, our Marching Tetrahedra approach allows us to presume (for example) that the point of intersection remains fixed on the face of the triangle even as the triangle moves. Since the implicit surface has no explicit parameterization, one is unable to similarly hold the intersection point fixed. The implicit surface utilizes an Eulerian point of view where the rays (which represent the discretization) are held fixed while the implicit surface moves (as ϕ values change), in contrast to our Lagrangian discretization where the rays are allowed to move/bend in order to follow fixed intersection points during differentiation. A similar approach for an implicit surface would hold the intersection point inside the tetrahedron fixed even as ϕ changes. Although such an approach holds potential due to the fact that implicit surfaces are amenable to computing derivatives off of the surface itself, the merging/pinching of isocontours created by convexity/concavity would likely lead to various difficulties. Furthermore, other issues would need to be addressed as well, e.g. the gradients (and thus normals) are only piecewise constant (and thus discontinuous) in the piecewise linear tetrahedral mesh basis.

B. Skinning

There are two options for the algorithm ordering between skinning and Marching Tetrahedra (the latter of which reverses the order in Figure 2 of the main paper). For skinning the triangle mesh, the skinned position of each triangle mesh vertex is $v_i(\theta, \phi) = \sum_j w_{ij}(\phi) T_j(\theta) v_i^j(\phi)$ where v_i^j is the location of v_i in the untransformed reference space of joint j. Unlike in Section 4.1 where w_{kj} and u_k^j were fixed, w_{ij} and v_i^j both vary yielding three terms in the product rule. $\partial v_i^j / \partial \phi$ is computed according to Equation 3 in the main paper, noting that u_{k_1} and u_{k_2} are fixed. $w_{ij}(\phi)$ is defined similarly to Equation 2 in the main paper,

$$w_{ij} = \frac{-\phi_{k_2}}{\phi_{k_1} - \phi_{k_2}} w_{k_1 j} + \frac{\phi_{k_1}}{\phi_{k_1} - \phi_{k_2}} w_{k_2 j} \tag{1}$$

where $w_{k_1 j}$ and $w_{k_2 j}$ are fixed; similar to Equation 3, $\partial w_{ij} / \partial \phi$ will contain $\mathcal{O}(1/\epsilon)$ coefficients. For skinning the tetrahedral mesh, Equations 2 and 3 directly define v_i and $\partial v_i / \partial \phi$ since the skinning is moved to the tetrahedral mesh vertices u_k. Then, $\partial v_i / \partial u_k$ is computed according to Equation 2 in order to chain rule to skinning (i.e. to $\partial u_k / \partial \theta$, which is computed according to the equations in Section 4.1).

C. Image Rasterization Implementation

C.1. Normals

Recall (from Section 5) that triangle vertices are reordered (if necessary) in order to obtain outward-pointing face normals. The area-weighted outward face normal is

$$n_f(v_1, v_2, v_3) = \frac{1}{2}(v_2 - v_1) \times (v_3 - v_1) \tag{2}$$

where

$$Area(v_1, v_2, v_3) = \frac{1}{2} ||(v_2 - v_1) \times (v_3 - v_1)||_2 \tag{3}$$

is the area weighting. Area-averaged vertex unit normals $\hat{n}_v$ are computed via

$$n_v = \sum_f n_f \qquad \hat{n}_v = \frac{n_v}{||n_v||_2} \tag{4}$$

where f ranges over all the triangle faces that include vertex v. Note that one can drop the 1/2 in Equation 2, since it cancels out when computing $\hat{n}_v$ in Equation 4.

C.2. Camera Model

The camera rotation and translation are used to transform each vertex v_g of the geometry to the camera view coordinate system (where the origin is located at the camera aperture), i.e. $v_c = Rv_g + T$. The normalized device coordinate system normalizes geometry in the viewing frustum (with $z \in [n, f]$) so that all $x, y \in [-1, 1]$ and all $z \in [0, 1]$. See Figure 1, left. Vertices are transformed into this coordinate system via

$$\begin{bmatrix} [v_{NDC}] \; z_c \\ z_c \end{bmatrix} = \begin{bmatrix} \frac{2n}{W} & 0 & 0 & 0 \\ 0 & \frac{2n}{H} & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & \frac{-fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} [v_c] \\ 1 \end{bmatrix} \tag{5}$$

where $H = 2n \tan(\theta_{\text{fov}}/2)$ is the height of the image, θ_{fov} is the field of view, $W = Ha$ is the width of the image, and a is the aspect ratio. The screen coordinate system is obtained by transforming the origin to the top left corner of the image, with $+x$ pointing right and $+y$ pointing down. See Figure 1, right. Vertices are transformed into this coordinate system via

$$\begin{bmatrix} [v'] \\ 1 \end{bmatrix} = \begin{bmatrix} -W/2 & 0 & 0 & W/2 \\ 0 & -H/2 & 0 & H/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} [v_{NDC}] \\ 1 \end{bmatrix} \tag{6}$$

or via

$$\begin{bmatrix} [v'] \; z_c \\ z_c \end{bmatrix} = \begin{bmatrix} -n & 0 & W/2 & 0 \\ 0 & -n & H/2 & 0 \\ 0 & 0 & \frac{f}{f-n} & \frac{-fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} [v_c] \\ 1 \end{bmatrix} \tag{7}$$

which is obtained by multiplying both sides of Equation 6 by z_c and substituting in Equation 5.

C.3. Normal Map

For each pixel, a ray is cast from the camera aperture through the pixel center to find its first intersection with the triangulated surface at a point p in world space. Denoting v_1, v_2, v_3 as the vertices of the intersected triangle, barycentric weights for the intersection point

$$\begin{aligned} \hat{\alpha}_1 &= \frac{Area(p, v_2, v_3)}{Area(v_1, v_2, v_3)} \\ \hat{\alpha}_2 &= \frac{Area(v_1, p, v_3)}{Area(v_1, v_2, v_3)} \\ \hat{\alpha}_3 &= \frac{Area(v_1, v_2, p)}{Area(v_1, v_2, v_3)} \end{aligned} \tag{8}$$

[1]https://pytorch3d.org/docs/cameras

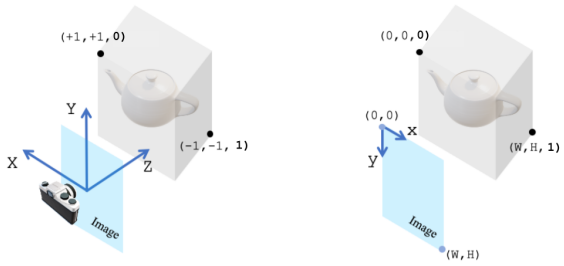


Figure 1. The normalized device (left) and screen (right) coordinate systems used during rasterization (based on Pytorch3D conventions[1]).

are used to compute a rotated (into screen space) unit normal from the unrotated vertex unit normals (see Equation 4) via

$$\hat{n} = R \frac{\hat{\alpha}_1 \hat{n}_{v_1} + \hat{\alpha}_2 \hat{n}_{v_2} + \hat{\alpha}_3 \hat{n}_{v_3}}{||\hat{\alpha}_1 \hat{n}_{v_1} + \hat{\alpha}_2 \hat{n}_{v_2} + \hat{\alpha}_3 \hat{n}_{v_3}||} \tag{9}$$

for the normal map. Note that dropping the denominators in Equation 8 does not change $\hat{n}$.

C.4. Scanline Rendering

After projecting a visible triangle into the screen coordinate system (via Equation 7), its projected area can be computed as

$$Area2D(v'_1, v'_2, v'_3) = -\frac{1}{2} det \begin{pmatrix} x'_2 - x'_1 & y'_2 - y'_1 \\ x'_3 - x'_1 & y'_3 - y'_1 \end{pmatrix} \tag{10}$$

similar to Equation 3 (where the negative sign accounts for the fact that visible triangles have normals pointing towards the camera). When a projected triangle overlaps a pixel center p', barycentric weights for p' are computed by using $Area2D$ instead of $Area$ in Equation 8. Notably, unnormalized world space barycentric weights can be computed from un-normalized screen space barycentric weights via $\alpha_1 = z'_2 z'_3 \alpha'_1$, $\alpha_2 = z'_1 z'_3 \alpha'_2$, $\alpha_3 = z'_1 z'_2 \alpha'_3$ or

$$\begin{aligned} \alpha_1 &= z'_2 z'_3 Area2D(p', v'_2, v'_3) \\ \alpha_2 &= z'_1 z'_3 Area2D(v'_1, p', v'_3) \\ \alpha_3 &= z'_1 z'_2 Area2D(v'_1, v'_2, p') \end{aligned} \tag{11}$$

giving

$$\hat{n} = R \frac{\alpha_1 \hat{n}_{v_1} + \alpha_2 \hat{n}_{v_2} + \alpha_3 \hat{n}_{v_3}}{||\alpha_1 \hat{n}_{v_1} + \alpha_2 \hat{n}_{v_2} + \alpha_3 \hat{n}_{v_3}||} \tag{12}$$

as an (efficient) alternative to Equation 9. If more than one triangle overlaps p', the closest one (i.e. the one with the smallest value of $z' = \hat{\alpha}'_1 z'_1 + \hat{\alpha}'_2 z'_2 + \hat{\alpha}'_3 z'_3$ at p') is chosen.

C.5. Computing Gradients

For each pixel overlapped by the triangle mesh, the derivative of the normal (in Equation 12) with respect to the vertices of the triangle mesh is required, i.e. $\partial\alpha_i/\partial v_g$ and $\partial\hat{n}_{v_i}/\partial v_g$ are required. $\partial\alpha_i/\partial v'$ can be computed from Equations 11 and 10, $\partial v'/\partial v_c$ can be computed from Equation 7, and $\partial v_c/\partial v_g$ can be computed from $v_c = Rv_g + T$. $\partial\hat{n}_{v_i}/\partial v_g$ can be computed from Equations 4 and 2.

D. Additional Comparisons

Additional results using the PeopleSnapshot dataset are shown in Figure 2. Our method is able to recover significantly more face and clothing details compared to prior work.

E. Network Efficacy

Given ground truth 3D data from [6], we show that our network has the capacity and flexibility to reconstruct clothed humans from either a single image or multiple images. Regardless of the number of input images, the network is trained by minimizing the normal map loss, SDF regularization losses, and silhouette losses. In the multiview case, each image is considered individually (i.e. we treat multiview as a collection of single view examples). Figure 3 shows an example of the results obtained by training our network on 8 camera views surrounding the person (as compared to the ground truth).

References

[1] Thiemo Alldieck, Marcus Magnor, Weipeng Xu, Christian Theobalt, and Gerard Pons-Moll. Video based reconstruction of 3d people models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8387–8397, 2018. 4

[2] Sai Praveen Bangaru, Michaël Gharbi, Fujun Luan, Tzu-Mao Li, Kalyan Sunkavalli, Milos Hasan, Sai Bi, Zexiang Xu, Gilbert Bernstein, and Fredo Durand. Differentiable rendering of neural sdfs through reparameterization. In *SIGGRAPH Asia 2022 Conference Papers*, pages 1–9, 2022. 1

[3] Xu Chen, Tianjian Jiang, Jie Song, Jinlong Yang, Michael J Black, Andreas Geiger, and Otmar Hilliges. gdna: Towards generative detailed neural avatars. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20427–20437, 2022. 1

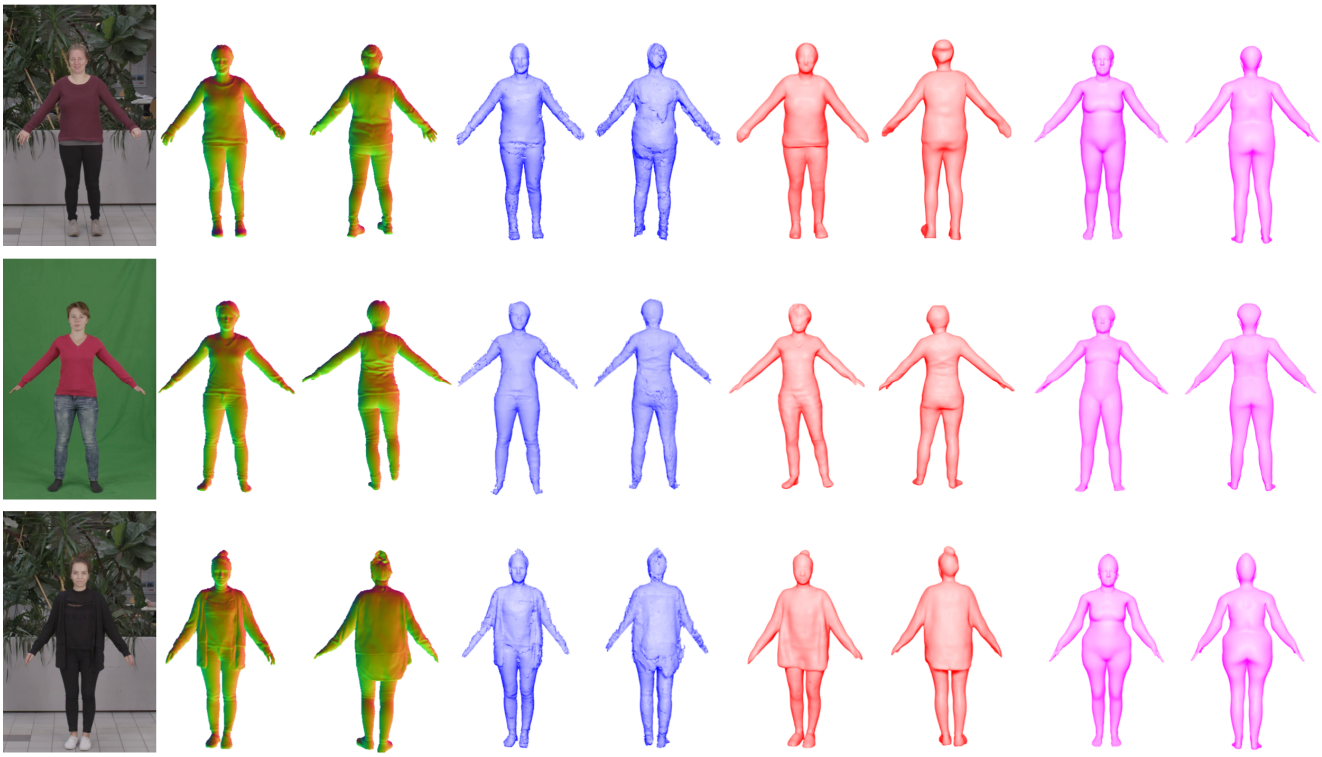
[4] Boyi Jiang, Yang Hong, Hujun Bao, and Juyong Zhang. Selfrecon: Self reconstruction your digital avatar from monocular video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5605–5615, 2022. 4

[5] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3504–3515, 2020. 1

[6] Renderpeople, 2018. 3

[7] Delio Vicini, Sébastien Speierer, and Wenzel Jakob. Differentiable signed distance function rendering. *ACM Transactions on Graphics (TOG)*, 41(4):1–18, 2022. 1

[8] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance. *Advances in Neural Information Processing Systems*, 33:2492–2502, 2020. 1



(a) Input Video (b) Inferred Normals (c) Our Method (d) SelfRecon (e) VideoAvatar

Figure 2. Additional PeopleSnapshot dataset results comparing our method, SelfRecon [4], and VideoAvatar [1].

Figure 3. After training from 8 camera views, the input image in the first column results in the geometry shown in the second column. Note that the geometry is shown from novel views. For the sake of comparison, the ground truth geometry is shown from the same novel views. See also Figure 4.

Figure 4. PIFuHD results, inferred using the input image in Figure 3 and shown from the same novel views (as in Figure 3). We stress that these results were obtained using inference from a single image, and so one would not expect the same efficacy (especially from novel views); however, these images do help to calibrate what one might expect from state-of-the-art inference. The conclusion is that our network has the ability to output high-quality reconstructed geometry.