A. Proofs of Geometric Properties

For a shape S, we consider visible oriented points (p, v) (i.e., $\xi(p, v) = 1$), such that $q(p, v) = p + d(p, v)v \in S$, unless otherwise specified.

A.1. Property I: Directed Eikonal Equation

First, note that for any visible (p, v) (with $p \notin S$), there exists an $\epsilon > 0$ such that any $\delta \in (\delta_0 - \epsilon, \delta_0 + \epsilon)$ satisfies $q(p, v) = q(p + \delta v, v) \in S$. In such a case, by definition of the directed distance field (DDF), $d(p+\delta v, v) = d(p, v) - \delta$. Restrict δ to this open interval. The directional derivative along v with respect to position p is then

$$\begin{aligned}\nabla_p d(p, v)^T v &= \lim_{\delta \to 0} \frac{d(p + \delta v, v) - d(p, v)}{\delta} \\ &= \lim_{\delta \to 0} \frac{d(p, v) - \delta - d(p, v)}{\delta} \\ &= -1,\end{aligned}$$

as required.

A.1.1 Gradient Norm Lower Bound

For any visible (p, v), since $|\nabla_p dv| = ||\nabla_p d||_2 ||v||_2 \, | \cos(\theta(\nabla_p d, v))| = 1$, where $\theta(u_1, u_2)$ denotes the angle between vectors u_1 and u_2, we also see that $||\nabla_p d||_2 = 1/|\cos(\theta(\nabla_p d, v))|$, and thus $||\nabla_p d||_2 \geq 1$.

A.1.2 Visibility Gradient

Consider the same setup as in A.1. The visibility field satisfies a similar property: $\xi(p + \delta v, v) = \xi(p, v)$, as the visibility cannot change when moving along the same view line (away from S). Thus, we get $\nabla_p \xi(p, v)^T v = 0$.

A.2. Property II: Surface Normals

Consider a coordinate system with origin $q_0 \in S$, with a frame given by $(\widehat{i}, \widehat{j}, \widehat{k})$ where $\widehat{k} = n(q_0)$ and $\widehat{i}, \widehat{j} \in \mathcal{T}_{q_0}(S)$ spans the tangent space at q_0. Locally, near q_0, reparameterize S in this coordinate system via

$$S(x, y) = (x, y, f_S(x, y)), \tag{6}$$

where f_S controls the extension of the surface in the normal direction. Notice that

$$\partial_\alpha S|_{q_0} = (\delta_{x\alpha}, \delta_{y\alpha}, \partial_\alpha f_S|_{q_0}), \tag{7}$$

where $\alpha \in \{x, y\}$ and $\delta_{x\alpha}$ is the Kronecker delta, but since $\partial_x S|_{q_0} = \widehat{i}$ and $\partial_y S|_{q_0} = \widehat{j}$ are in the tangent plane,

$$\partial_\alpha f_S|_{q_0} = 0. \tag{8}$$

Consider any oriented position (p, v) that points to $q(p, v) \in S$ on the surface. Locally, the surface can be reparameterized in terms of (p, v):

$$q(p, v) = p + d(p, v)v \in S. \tag{9}$$

Yet, using $q = (q_x, q_y, q_z)$, we can write this via Eq. 6 as

$$S(q_x, q_y) = (q_x, q_y, f_S(q_x, q_y)), \tag{10}$$

where q_x is the component along the x direction in local coordinates, which depends on p and v. In other words, the z component of q depends on p via:

$$q_z(p, v) = f_S(q_x(p, v), q_y(p, v)). \tag{11}$$

Let (p_0, v_0) point to q_0 (i.e., $q_0 = q(p_0, v_0)$). Then:

$$\begin{aligned}\partial_{p_i} f_S|_{p_0} &= \underbrace{\partial_{q_x} f_S|_{q_0}}_{0} \partial_{p_i} q_x|_{p_0} + \underbrace{\partial_{q_y} f_S|_{q_0}}_{0} \partial_{p_i} q_y|_{p_0} \\ &= 0 \;\; \forall \; i \in \{x, y, z\},\end{aligned}$$

since $\partial_{q_\alpha} f_S|_{q_0} = 0$ using Eq. 8.

Derivatives with respect to position are then given by

$$\partial_{p_z} q_z|_{p_0} = 1 + \partial_{p_z} d(p, v)|_{p_0} v_z = \partial_{p_z} f_S|_{p_0} = 0 \tag{12}$$

$$\partial_{p_\alpha} q_z|_{p_0} = \partial_{p_\alpha} d(p, v)|_{p_0} v_z = \partial_{p_\alpha} f_S|_{p_0} = 0, \tag{13}$$

using Eq. 9 and Eq. 11, with $\alpha \in \{x, y\}$. Thus, using Eq. 12 and Eq. 13,

$$\frac{\partial}{\partial(p_x, p_y, p_z)} d(p, v)|_{p_0} = (0, 0, -1/v_z), \tag{14}$$

in local coordinates. Since the z component here is along the direction of the surface normal, $n(q_0) = n(p_0, v_0)$, this can be rewritten as

$$\nabla_p d(p, v)|_{p_0, v_0} = \frac{-1}{v_z} n(q_0)^T = \frac{-n(q_0)^T}{n(q_0)^T v}. \tag{15}$$

For any C^1 surface S, and point $q_0 \in S$ (that is intersected by visible oriented point (p, v)), we can always construct such a coordinate system, so we can more generally write:

$$\nabla_p d(p, v) = \frac{-n^T}{n^T v}. \tag{16}$$

A.3. Property III: Gradient Consistency

Consider the same setup (coordinate system) and notation as in the Proof of Property II above (Appendix A.2). Since $v \in \mathbb{S}^2$, it suffices to consider an infinitesimal rotational perturbation of some initial view direction v_0:

$$dR(t) = [\omega]_\times dt, \tag{17}$$

where $[\omega]_\times$ is a skew-symmetric cross-product matrix of some angular velocity vector ω, so that $\widetilde{v} = (I + dR(t))v_0$ and $dv = \widetilde{v} - v_0 = [\omega]_\times \, dt \, v_0$ is the change in the view (given dt) and a velocity of $u := \partial_t v = [\omega]_\times v_0$. The change in surface position, $q(p, v(t)) = p + d(p, v(t))v(t)$, with respect to t is then

$$\begin{aligned}\partial_t q &= v \partial_t d + d \partial_t v \\ &= (\partial_v d \partial_t v) v + d \partial_t v \\ &= (\partial_v du) v + du.\end{aligned}$$

The z component, in local coordinates, is then

$$\begin{aligned}\partial_t q_z|_{q_0} &= \partial_t f_S(q_x, q_y)|_{q_0} \\ &= \partial_{q_x} f_S|_{q_0} \partial_t q_x + \partial_{q_y} f_S|_{q_0} \partial_t q_y \\ &= 0,\end{aligned}$$

via Eq. 8. Thus, $\partial_t q_z|_{q_0} = (\partial_v du) v_z + du_z = 0$, meaning

$$du^T n = -(\partial_v du) v^T n \tag{18}$$

using $u_z = u^T n$ and $v_z = v^T n$ at q_0. Recalling that $\partial_p d = -n^T / (n^T v)$ and rearranging, we get

$$\partial_v du = -d \frac{n^T}{v^T n} u = d \partial_p du. \tag{19}$$

Notice that $u = \partial_t v = [\omega]_\times v_0$ is orthogonal to v_0 and the arbitrary vector ω. Hence, for any visible oriented point (p, v) away from viewing the boundary of S (i.e., where $n^T v = 0$) and for all $\omega \in \mathbb{R}^3$, we have

$$\nabla_v d [\omega]_\times v = d \, \nabla_v d [\omega]_\times v. \tag{20}$$

This constrains the derivatives with respect to the view vector to be closely related to those with respect to position, along any directions not parallel to v.

A.3.1 Alternative Expression

Note that the inner product with δ_v primarily serves to restrict the directional derivative in valid directions of v. A cleaner expression can be obtained with a projection operator, which removes directional components parallel to v.

First, let us define d to normalize v:

$$d(p, v) := d\left(p, \frac{v}{||v||_2}\right), \tag{21}$$

for all $v \in \mathbb{R}^3 \setminus \{0\}$. Then, consider a perturbation along the view direction of size $|\delta| < 1$:

$$d(p_0, v_0 + \delta v_0) = d(p_0, v_0(1 + \delta)) = d(p_0, v_0), \tag{22}$$

with $||v_0|| = 1$ and using Eq. 21 for the last step. This means the directional derivative along v must satisfy

$$\partial_v d(p, v) v = 0. \tag{23}$$

In the previous section, we showed that $\partial_v du = d \partial_p du$ for all $u \perp v$. Let $\mathcal{P}_v = I - vv^T$ be the orthogonal projection removing components parallel to v. Then we can rewrite the result of the previous section as $\partial_v d \mathcal{P}_v = d \partial_p d \mathcal{P}_v$. But $\partial_v d \mathcal{P}_v = \partial_v d - \partial_v d v v^T = \partial_v d$ by Eq. 23. Thus, we may write

$$\partial_v d = d \, \partial_p d \, \mathcal{P}_v, \tag{24}$$

which agrees with Eq. 20.

A.4. Property V: Local Differential Geometry

Given a visible oriented point (p, v), we have shown that the surface normal $n(p, v)$ on S (at $q = p + d(p, v)v$) is computable from $\nabla_p d$ (see Appendix A.2). Curvatures, however, require second-order information.

We first construct a local coordinate system via n, by choosing two tangent vectors at q: $t_x, t_y \in \mathcal{T}_q(S)$, where $||t_\alpha||_2 = 1$ and $t_x^T t_y = 0$. In practice, this can be done by sampling Gaussian vectors, and extracting an orthogonal tangent basis from them. We can then reparameterize the surface near $q_0 = q(p_0, v_0)$ (with surface normal $n_0 = n(q_0)$) via $S(x, y) = q_0 + x t_x + y t_y + f_S(x, y) n_0$, where x and y effectively control the position on the tangent plane. Alternatively, we can write $S(x, y) = q(p(u), v_0)$, which parameterizes S about the oriented point (or viewpoint) $p(u)$, where $u = (x, y)$, and $p(u) = p_0 + x t_x + y t_y$. Notice that $p(u)$ is essentially a local movement of p parallel to the tangent plane at $q_0 \in S$ (which is "pointed to" by (p_0, v_0)). Note that p_0, v_0, and q_0 are fixed; only u and $p(u)$ are varied. Further, notice that the plane defined by $p(u)$ (in the normalized tangent directions t_x and t_y) is parallel to the surface tangent plane at q_0 (and thus orthogonal to n_0), but *not* necessarily perpendicular to v.

Using this local frame on $\mathcal{T}_{q_0}(S)$, the first-order derivatives of the surface are

$$\begin{aligned}\partial_i S|_{u=0} &= \partial_i \left(p(u) + v_0 d(p(u), v_0) \right)|_{u=0} && (25) \\ &= t_i + \partial_i d(p(u), v_0)|_{u=0} v_0 && (26) \\ &= t_i + (\nabla_p d(p, v_0) t_i) v_0 && (27)\end{aligned}$$

where $i \in \{x, y\}$ and p_j is the jth component of p. Then the metric tensor (first fundamental form) is given by

$$\begin{aligned}g_{ij} &= \partial_i S|_{u=0}^T \, \partial_j S|_{u=0} && (28) \\ &= \delta_{ij} + c_i c_j + c_i v_0^T t_j + c_j v_0^T t_i, && (29)\end{aligned}$$

where $i, j \in \{x, y\}$, $u = (x, y)$, δ_{ij} is the Kronecker delta function, and $c_k = \nabla_p d t_k$. Notice that $\nabla_p d$ is parallel to n (see Property II), and thus orthogonal to t_1 and t_2. Thus, $c_x = c_y = 0$, and so, in theory, $g_{ij} = \delta_{ij}$, but this may not hold for a learned DDF, if one computes directly with Eq. 29. However, in practice, we use the theoretical value (where $g = I_2$ in local coordinates), as deviations from it are due to error only (assuming correct normals).

Next, the shape tensor (second fundamental form) II_{ij} can be computed from the second-order derivatives of the surface [32] via:

$$\begin{aligned}
\mathrm{II}_{ij} &= n_0^T \partial_{ij} S|_{u=0} \qquad (30)\\
&= n_0^T \partial_{ij}(p(u) + v_0 d(p(u), v_0))|_{u=0}\\
&= n_0^T v_0 \partial_{ij} d(p(u), v_0)|_{u=0}\\
&= n_0^T v_0 \partial_i \sum_k \partial_{p_k} d(p(u), v_0) \underbrace{[\partial_j p(u)]_k}_{[t_j]_k} |_{u=0}\\
&= n_0^T v_0 \sum_k \partial_i \partial_{p_k} d(p(u), v_0)[t_j]_k|_{u=0}\\
&= n_0^T v_0 \sum_k [t_j]_k \sum_\ell \partial_{p_\ell} \partial_{p_k} d(p(u), v_0)[\partial_i p(u)]_\ell|_{u=0}\\
&= n_0^T v_0 \sum_k [t_j]_k \sum_\ell \underbrace{\partial_{p_\ell, p_k} d(p(u), v_0)}_{\mathcal{H}_p[d]_{k\ell}}[t_i]_\ell|_{u=0}\\
&= \left(t_j^T \mathcal{H}_p[d] t_i\right) n_0^T v_0, \qquad (31)
\end{aligned}$$

where $i, j \in \{x, y\}$, $u = (x, y)$, $n_0 = n(q_0)$, $[t_i]_k$ is the kth component of t_i, $\mathcal{H}_p[d]$ is the Hessian of d with respect to p (at $u = 0$), and $p(u) = p_0 + x t_x + y t_y$. Note that our parameterization of the surface using the DDF (i.e., $q(p(u), v_0)$) defines surface deviations in terms of v_0; the $n_0^T v_0$ term undoes this effect, rewriting the deviation in terms of n instead.

Curvatures can then be computed via the shape tensor (see, e.g., [32]). Gaussian curvature is given by $\mathcal{C}_K = \det(\mathrm{II}) / \det(g)$, while mean curvature is written $\mathcal{C}_H = \mathrm{tr}(\mathrm{II} g^{-1})$. Notice that these quantities can be computed for any visible (p, v), using only $d(p, v)$ and derivatives of d with respect to p (e.g., with auto-differentiation). Thus, the curvatures of any visible surface point can be computed using only local field information at that oriented point.

A.5. View Consistency

DDFs ideally satisfy a form of view consistency, which demands that an opaque position viewed from one direction must be opaque from all directions, with depth lower-bounded by that known surface position. Consider two oriented points (p_1, v_1) and (p_2, v_2). Assume (i) that (p_1, v_1) is visible, such that $q_1 = p_1 + d(p_1, v_1) v_1 \in S$, and (ii) that there exists $t > 0$ such that $\ell_{p_2,v_2}(t) = p_2 + t v_2 = q_1$. That is, if viewing the scene via (p_2, v_2), we assume our line of sight intersects that of (p_1, v_1), at the surface point q_1.

Then, in this case, we have that: (1) (p_2, v_2) is visible, meaning $\xi(p_2, v_2) = 1$, and (2) $d(p_2, v_2) \le ||p_2 - q_1||_2 = t$. These can be seen by the definition of ξ and d. For (1), since there exists a surface intersection (at q_1) along ℓ_{p_2,v_2}, the oriented point (p_2, v_2) must be visible (i.e., $\xi(p_2, v_2) = 1$). For (2), $d(p_2, v_2)$ can be no further than t, since DDFs return the minimum distance to a point on S along the line $\ell_{p_2,v_2}(t)$, and hence its output can be no greater than the known (assumed) distance t.

A.6. Neural Depth Renderers as DDFs

A natural question is how to parallelize a DDF. Consider a function $f : \Lambda_\Pi \times \Lambda_S \to \mathcal{I}^m_{d,\xi}$, where $\mathcal{I}_{d,\xi} = \mathbb{R}_+ \times [0, 1]$, from continuous camera parameters (i.e., elements of Λ_Π) and some space of shapes Λ_S to an m-pixel depth and visibility image. Then f is implicitly a conditional DDF, as each pixel value can be computed as $(d(p, v), \xi(p, v))$, where p and v are a fixed function of $\Pi \in \Lambda_\Pi$. In other words, the camera intrinsics and extrinsics determine the p and v value for each pixel, with p determined by the centre of projection (camera position) and v determined by the direction corresponding to that pixel, from the centre of projection out into the scene. We may thus regard the depth pixel value as a function of p and v, via the camera parameters. This holds regardless of the architecture of f. Thus, all the properties of DDFs hold in these cases as well: for example, for a depth image I_d, camera position C_p and viewpoint C_v, Property II relates $\partial_{C_p} I_d$ to the surface normals image, while Property III constrains $\partial_{C_v} I_d$. We believe this point of view can improve the framework for differentiable neural rendering of geometric quantities (e.g., [50, 77, 84, 85]).

B. Mesh Data

We display visualizations of each data type in Fig. 10. We briefly describe how each is computed (note that we only need to obtain (p, v), after which (ξ, d) can be obtained via ray-triangle intersection):

- Uniform (U): simply sample $p \sim \mathcal{U}[\mathcal{B}]$ and $v \sim \mathcal{U}[\mathbb{S}^2]$.
- At-surface (A): start with $q_0 \sim \mathcal{U}[S]$ (obtained via area-weighted sampling) and $v_0 \sim \mathcal{U}[\mathbb{S}^2]$. Sample p on the line between q_0 and its intersection with $\mathcal{B}$ along v_0. Set $v = -v_0$. Note that the final output data may not actually intersect q_0 (since one may pass through a surface when sampling p).
- Bounding (B): sample $p \sim \mathcal{U}[\partial\mathcal{B}]$ and $v \sim \mathcal{U}[\mathbb{S}^2]$, but restrict v to point to the interior of $\mathcal{B}$.
- Surface (S): simply use $p \sim \mathcal{U}[S]$ and then take $v \sim \mathcal{U}[\mathbb{S}^2]$.
- Tangent (T): the procedure is the same as for A-type data, except we enforce v_0 to lie in the tangent plane $\mathcal{T}_{q_0}(S)$.
- Offset (O): take a T-type (p, v) and simply do $p \leftarrow p + \varsigma_O \epsilon_O n_0$, where n_0 is the normal at q_0, and we set $\epsilon_O = 0.05$ and sample $\varsigma_O \sim \mathcal{U}[\{-1, 1\}]$.

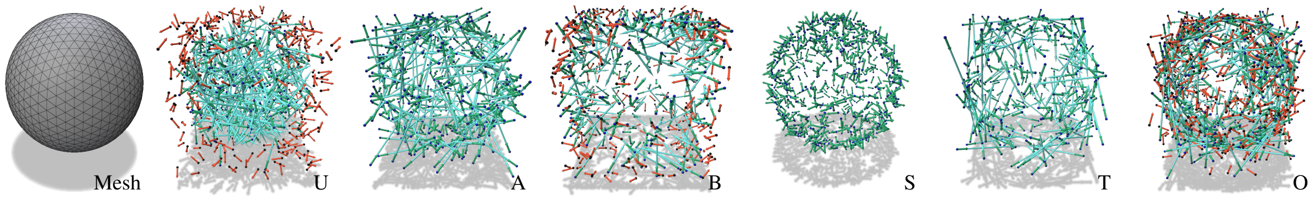

Figure 10. Illustration [68] of data types. Left to right: input sphere mesh, U, A, B, S, T, and O data. Visible points depict p in blue, v in green, and a line from p to q in turquoise; non-visible points depict p in black and v in red. See §3.4 for details and §C.2 for an ablation study of the data types.

Since we assume $\mathcal{B}$ is an axis aligned box (with maximum length of 2; i.e., at least one dimension is $[-1, 1]$), sampling positions on it, or directions with respect to it, is straightforward. See also §C.2, which examines the effect of ablating each data type.

C. Single Entity Fitting Details

Given a mesh, we first extract data of each type (see §3.4), obtaining (p, v) tuples in the following amounts: 250K (A and U) and 125K (B, T, O, and S). Since rendering outside $\mathcal{B}$ uses query points on $\partial\mathcal{B}$, we bias the sampling procedure for T, A, and O data, such that 10% of the p-values are sampled from $\partial\mathcal{B}$. For each minibatch, we sample 6K (A and U) and 3K (B, T, O, and S) points, across data types. In addition to these, we sample an additional 1K uniformly random oriented points per minibatch, on which we compute only regularization losses ($\mathcal{L}_V$ and $\mathcal{L}_{\text{DE}}$), for which ground truth values are not needed.

We note that not all loss terms are applied to all data types. As discussed in §3.4, the transition loss $\mathcal{L}_T$ is only applied to S and T type data (since we do not want spurious field discontinuities, due to the field switching between components, except when necessary). As briefly noted in Property II, the DDF gradient $\nabla_p d$ (and hence field-derived surface normals) are not well-defined when n and v are orthogonal. The gradients are also not well-defined on S-type data (since d is explicitly discontinuous for $p \in S$, and thus $\nabla_p d$ does not exist there). Hence, we do not apply the directed Eikonal regularization $\mathcal{L}_{\text{DE}}$ or the normals loss $\mathcal{L}_n$ to S, T, or O data. Similarly, the weight variance loss $\mathcal{L}_V$ (designed to reduce weight field entropy) should not be applied to S, T, or O samples, as the weight field should be transitioning near those samples (and thus have a higher pointwise Bernoulli variance). The remaining losses are applied on all data types.

We then optimize Eq. 2 via Adam (learning rate: 10^{-4}; $\beta_1 = 0.9$, $\beta_2 = 0.999$), using a reduction-on-plateau learning rate schedule (reduction by 0.9, minimum of 5K iterations between reductions). We run for 100K iterations, using $\gamma_d = 5$, $\gamma_\xi = 1$, $\gamma_n = 10$, $\gamma_V = 1$, $\gamma_{\text{E,d}} = 0.05$, $\gamma_{\text{E},\xi} = 0.01$, and $\gamma_T = 0.25$. Note that we double γ_d on A and U data. The field itself is implemented as a SIREN with seven hidden layers, initialized with $\omega_0 = 1$, each of size 512, mapping $(p, v) \in \mathbb{R}^6 \to (\{d_i\}_{i=1}^K, \{w_i\}_{i=1}^K, \xi) \in \mathbb{R}^{2K+1}$ (note that the set of w_i's has $K-1$ degrees of freedom). We use $K = 2$ delta components. Since $w_2 = 1 - w_1$, we output only w_1 and pass it through a sigmoid non-linearity. We apply ReLU to all d_j outputs, to enforce positive values, and sigmoid to ξ. We use the implementation from [9] for SIREN. In terms of data, we test on shapes from the Stanford Scanning Repository [73] (data specifically from [8, 33, 78]).

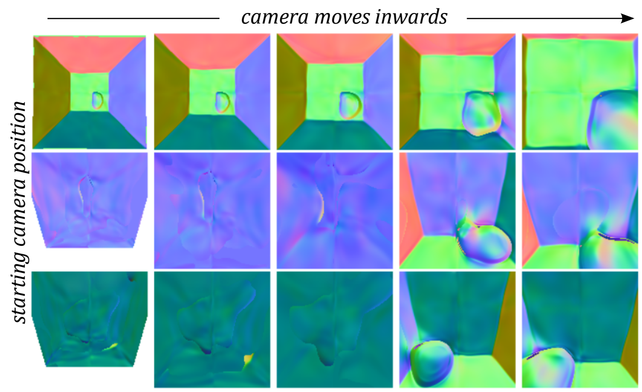

Figure 11. Example showing underfitting without compositional model (same scene as Fig. 6). Colours correspond to surface normals estimated via the DDF. (See Appendix C.1).

C.1. Non-Compositional Fitting Example

To show the improved scaling of composing DDFs, we also attempted to fit the same scene using the single-object fitting procedure above. For fairness, we doubled the number of data samples extracted, as well as the size of each hidden layer. In comparison to Fig. 6, this naive approach struggles to capture some high frequency details, though we suspect this could be mitigated by better sampling procedures.

Ablation	Minimum Distance Error (L_1; $\times 10$) $\downarrow$						Visibility Error (BCE) $\downarrow$					
	$\mathcal{L}_{d,1}$-U	$\mathcal{L}_{d,1}$-A	$\mathcal{L}_{d,1}$-B	$\mathcal{L}_{d,1}$-O	$\mathcal{L}_{d,1}$-T	$\mathcal{L}_{d,1}$-S	$\mathcal{L}_{\xi}$-U	$\mathcal{L}_{\xi}$-A	$\mathcal{L}_{\xi}$-B	$\mathcal{L}_{\xi}$-O	$\mathcal{L}_{\xi}$-T	$\mathcal{L}_{\xi}$-S
U	**0.58**	0.79	0.18	0.49	0.75	0.47	**2.11**	0.03	0.07	**0.56**	0.14	0.05
A	0.48	**0.82**	**0.20**	0.49	0.81	0.54	0.20	0.05	0.07	0.49	**0.18**	**0.07**
B	0.37	0.67	**0.20**	0.46	0.73	0.58	0.20	0.03	**0.10**	0.50	0.15	0.06
O	0.39	0.67	0.18	**0.56**	0.70	0.59	**0.28**	0.01	**0.11**	**1.71**	0.03	0.03
T	0.39	0.70	0.19	0.45	**0.84**	0.65	0.19	**0.09**	0.06	0.32	**0.48**	**0.07**
S	**0.55**	**0.95**	**0.23**	**0.51**	**0.89**	**1.43**	0.14	**0.06**	0.06	0.42	0.17	**0.48**
–	0.45	0.75	0.19	0.50	0.77	**0.67**	0.23	0.04	0.08	**0.56**	0.15	**0.07**

Table 2. Data sample type ablation on the Stanford Bunny (see §C.2). Rows: type of data (i.e., (p, v) sample type) ablated. Columns: errors on held-out data (left: minimum distance loss, but computed with L_1 instead of L_2, for greater interpretability; right: binary cross-entropy-based visibility loss). Each column computes the error on a different sample type (e.g., $\mathcal{L}_{d,1}$-A is the minimum distance error on A-type data). Error computation is on 25K held-out samples, for each data type. Per loss type, the **red** numbers are the *worst* (highest) error cases; the **pink** numbers are the *second-worst* error cases. Each ablation scenario uses 100K of each data type, except for the ablated one (of which it uses zero); the "–" case uses 83,333 samples of all six data types (to control for the total number of points). Several observations are notable: (1) in most cases, performance on a data type is worst or second-worst when that type is ablated in training; (2) ablating S appears to be damaging across other data types as well; and (3) there is some interplay between data types (e.g., ablating T-type data is worse for A-type data error, than ablating A-type data itself).

C.2. Data Sample Type Ablation Experiment

We perform a small-scale ablation experiment to discern the importance of each data sample type. We perform a single-shape fit to the Stanford Bunny, with slight modifications to the algorithm discussed in §C just above. In particular, we consider six scenarios, in each of which we train with 100K samples of each type except one, which is removed/ablated. We consider one other scenario that has the same number of total points, but no single type is ablated (i.e., 83,333 samples for each data type). We then measure the depth and visibility prediction error on 25K held-out samples of each data type, including the ablated one. All other aspects of the fitting process remain the same.

See Table 2 for results. We first notice that the worst errors are incurred for models trained where that data type was ablated in training. For the minimum distance error, the worse or second-worst result is always obtained for the model that ablates that data type; this is also true for the visibility error, except for ablating and testing on A-type data. This case may be due to the relation between A-type and T-type data, since T-type data is effectively the hardest subset of A-type data (excepting perhaps S-type data). There is also significant overlap between U-type and B-type data with A-type data. Hence, we speculate that ablating T-type data removes access to the most difficult (and most likely to have high error) of A-type samples, leading to high visibility error, whereas ablating A-type data itself can in part be accounted for by other data types (e.g., U-type).

One surprising scenario is the S-type data ablation, as it worsens performance across all other data types as well. This is potentially due to the greater difficulty in the network knowing when to transition between Dirac delta components in the weight field.

Finally, we also found that ablating B-type data has a relatively minor effect on the resulting errors. We remark that, for *rendering* applications, where the camera is more likely to be outside the box (and thus most field query positions p will be on $\partial\mathcal{B}$; see §3.2), B-type data will be much more important, and hence useful to concentrate on. Further, we note that the data bias for T-, A-, and O-type data (ensuring there are positions $p \in \partial\mathcal{B}$) leads to increased overlap with B-type data, reducing the effect of ablating it. Other effects, such as enforcing the directed Eikonal property, should help as well. We remark that both A and B sample types are biased: specifically, caring more about the shape than the non-visible parts of the scene (for A), and focusing on rendering-oriented applications (for B).

Overall, though this is only on a single shape with a simplistic set of scenarios, it suggests that each data type has information that the other types cannot completely make up for, especially for U, O, T, and S data.

D. UDF and v^*

D.1. v^* Extraction

We obtain v^* via a fitting procedure, with a forward pass similar to composition (see §4.1). Starting from an already trained (P)DDF, we define a small new SIREN network g_v (five hidden layers, each size 128), which maps position to a set of K_c *candidate directions*, such that $g_v : \mathbb{R}^3 \to \mathbb{S}^{2\times K_c}$ Given a position p and candidates $\{v_i^*\}_{i=1}^{K_c} = g_v(p)$, we can compute $\zeta_{v^*} = \{d(p, v_i^*(p)), \xi(p, v_i^*(p))\}_{i=1}^{K_c}$. To obtain a UDF depth estimate, we use

$$\widehat{\mathrm{UDF}}(p) = \sum_i \omega^{(i)}_{\zeta_{v^*}}(p)\, d(p, v_i^*(p)), \qquad (32)$$

where the weights $\omega_{\zeta_{v^*}}$ are based on those from our explicit composition method:

$$\omega_{\zeta_{v^*}}(p) = \text{Softmax}\left(\left\{ \frac{\eta_T^{-1}\xi(p, v_i^*(p))}{\varepsilon_s + d(p, v_i^*(p))} \right\}_i \right). \tag{33}$$

We use the same η_T and ε_s as for composition. Note that this formulation essentially takes the candidate directions, computes their associated distances, and then (softly) chooses the one that has lowest distance value while still being visible. We can also obtain $v^*(p)$ via these weights, using

$$\widetilde{v}^*(p) = \sum_i \omega^{(i)}_{\zeta_{v^*}}(p)\, v_i^*(p) \tag{34}$$

$$v^*(p) = \frac{\widetilde{v}^*(p)}{||\widetilde{v}^*(p)||_2}, \tag{35}$$

Note that, in the ideal case, $v^*(p) = -\nabla_p \text{UDF}(p)^T$, meaning we could compute it with a backward pass, but we found such an approach was noisier in practice.

To train g_v to obtain good candidates, we use the following loss:

$$\mathcal{L}_{v^*} = \frac{1}{K_c}\sum_i d(p, v_i^*(p)) - \xi(p, v_i^*(p)) \tag{36}$$

$$+ \frac{2\tau_n}{K_c^2 - K_c}\sum_i \sum_{j \neq i} v_i^*(p)^T v_j^*(p) \tag{37}$$

$$+ \frac{\tau_d}{K_c}\sum_i \left[v_i^*(p)^T n(p, v_i^*(p)) + 1 \right]^2 \tag{38}$$

where the first sum encourages obtaining the direction with minimum depth that is visible, the second prevents collapse of the candidates to a single direction (e.g., local minima), and the third encodes alignment with the local surface normals (similar to the directed Eikonal regularizer; see Appendix D.2). Note that the first sum (Eq. 36) includes a d term, which linearly penalizes longer distances, and a $-\xi$ term, which pushes the visibility (i.e., probability of surface existence in the direction $v_i^*(p)$, from position p) to be high. We use $K_c = 5$, $\tau_n = 5 \times 10^{-3}$, and $\tau_d = 0.1$. Optimization was run for 10^4 iterations using Adam (LR: 10^{-4}; $\beta_1 = 0.9$, $\beta_2 = 0.999$). Each update used 4096 points, uniformly drawn from the bounding volume ($p \sim \mathcal{U}[\mathcal{B}]$).

D.2. Surface Normals of v^*

Let v^* be defined as in §4.2 and choose p such that $v^*(p)$ is not multivalued (i.e., neither on S nor on the medial surface of S). Recall that, by definition,

$$\text{UDF}(p) = d(p, v^*(p)) \tag{39}$$

and

$$n(p, v) = n(q(p, v)) = \frac{\nabla_p d(p, v)^T}{||\nabla_p d(p, v)||} \tag{40}$$

where $n(p, v)^T v < 0$ (see Property II). Then, by Eq. 39,

$$\begin{aligned}\nabla_p \text{UDF}(p) &= \nabla_p d(p, v^*(p)) \\ &= ||\nabla_p d(p, v^*(p))||\, \frac{\nabla_p d(p, v^*(p))}{||\nabla_p d(p, v^*(p))||} \\ &= ||\nabla_p \text{UDF}(p)||\; n(p, v^*(p))^T \\ &= n(p, v^*(p))^T,\end{aligned}$$

via the S/UDF Eikonal equation in the last step.

Finally, by the directed Eikonal property (I), we have:

$$\nabla_p \text{UDF}(p) v^*(p) = \nabla_p d(p, v^*(p)) v^*(p) = -1, \tag{41}$$

meaning $\nabla_p \text{UDF}(p) = -v^*(p)^T$. Indeed, recall that the gradient of a UDF has (1) unit norm (i.e., satisfies the Eikonal equation) and (2) points away from the closest point on S, which is precisely the definition of $-v^*(p)$.

E. Single-Image 3D Reconstruction

Architecture and Optimization. Our CPDDF is a modulated SIREN [44] with layer sizes `(512, 512, 512, 256, 256, 256, 256)`. Note that we use a softplus activation instead of ReLU, when multiplying the modulator to the intermediate features. The encoders (one ResNet-18 for the camera, and another for inferring the latent conditioning shape vector) take 128×128 RGBA images as input. However, when rendering the visibility mask for $\mathcal{L}_M$, we output 64×64 images. We set $\gamma_{R,S} = 1$, $\gamma_{R,\Pi} = 5$, and $\gamma_{R,M} = 10$. We also used small weight decays on the camera and shape predictors (10^{-6} and 10^{-3}, respectively). Training ran for 100K iterations with AdamW [42] and a batch size of 32. We used $\dim(z_s) = 512$ and a SIREN initializer of $\omega_0 = 1$.

Shape-fitting Loss. We use the same settings as the single-shape fitting experiments, with slight modifications: $\gamma_d = 5$, $\gamma_\xi = 10$, $\gamma_n = 10$, $\gamma_V = 1$, $\gamma_{\text{E,d}} = 0.1$, $\gamma_{\text{E},\xi} = 0.25$, and $\gamma_T = 0.1$. An additional loss on the variance of ξ is also applied (to reduce the visibility entropy at each point, which leads to fuzzy renders):

$$\mathcal{L}_{V,\xi} = \gamma_{V,\xi}\xi(p, v)[1 - \xi(p, v)], \tag{42}$$

where we set $\gamma_{V,\xi} = 0.25$. This encourages less blurriness in the shape output due to uncertainty in the visibility field. For minibatches, we sample 1.2K (A and U) and 300 (S, B, T, and O) oriented points per shape, with each minibatch containing 32 shapes.

Camera Loss. The camera fitting loss $\mathcal{L}_\Pi$ utilizes a camera representation only involving extrinsic position (the camera is assumed to be looking at the origin). In particular, we use the azimuth, elevation, and radius representation of the camera position. Before computing the L_2 loss, we z-score normalize each element, based on the training

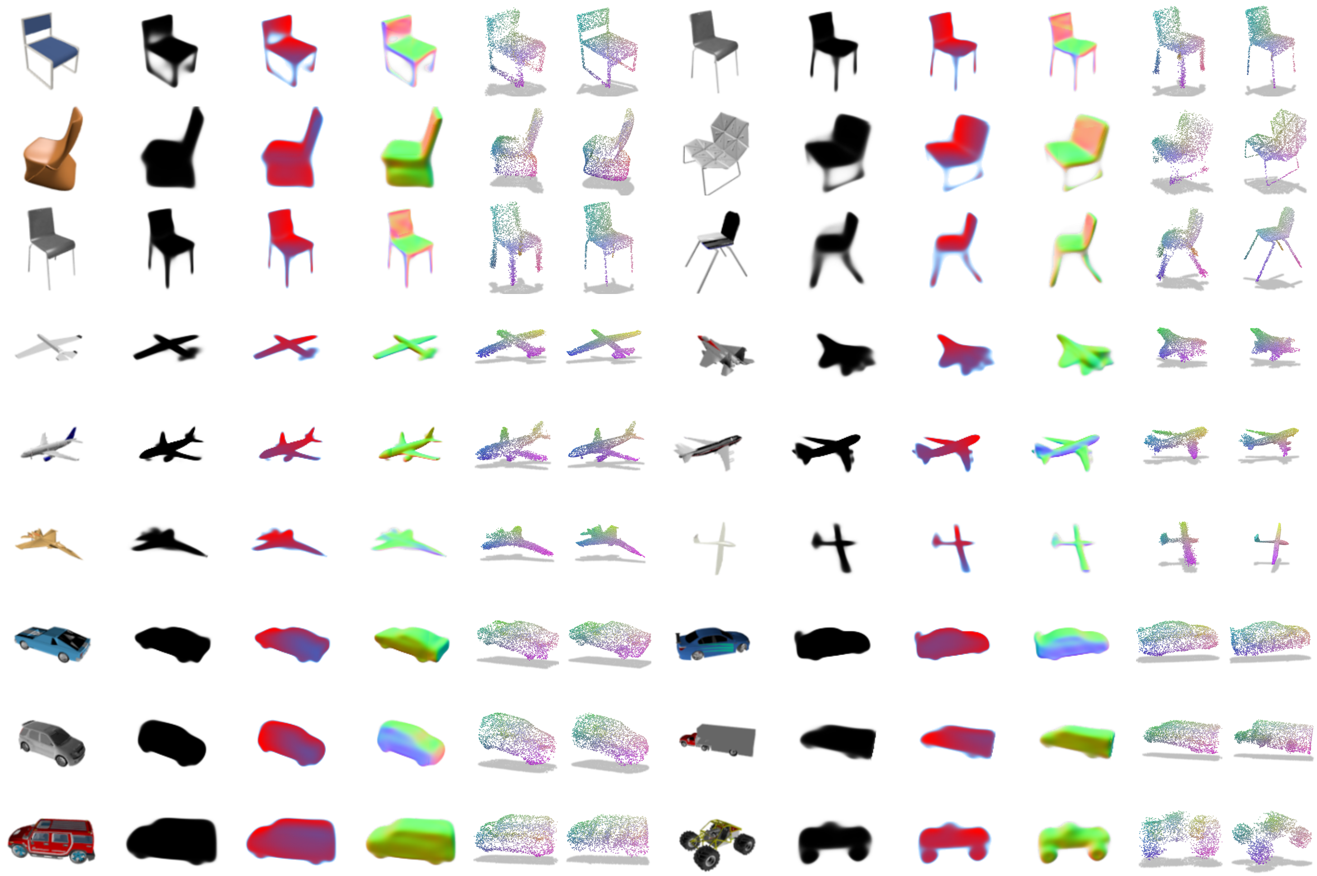

Figure 12. Single-image 3D reconstruction visualizations on held-out data. Per inset, columns represent (i) the input RGB image, (ii) the visibility $\widehat{\xi}$, (iii) the depth $\widehat{d}$, (iv) the normals $\widehat{n}$, (v) the sampled point cloud (PC) from the DDF, and (vi) a sample from the ground-truth PC. Quantities (ii-v) are all differentiably computed directly from the CPDDF and $\widehat{\Pi}$, per point or pixel (i.e., no post-processing needed). PC colours denote 3D coordinates. A high-error example is in the lower-right of each category. See also Fig. 8 and §E.2.

set statistics. We also restrict the predicted camera $\widehat{\Pi}$ to be within the range of the parameters observed in the dataset.

Rendering Scale Factor. We note that an additional scale factor is needed for rendering DDFs for ShapeNet. Since ShapeNet shapes are normalized with an instance-dependent measure (the bounding box diagonal), one needs to know the scale to reproduce the output image. This is an issue as our CPDDF always outputs a shape in $[-1, 1]^3$, with training data normalized to have longest axis-aligned bounding box length equal to two. Further, it creates an ambiguity (with respect to the output image) with the camera position (radius from the shape). At train time, before rendering to compute $\mathcal{L}_M$, we use the ground-truth scale factor. At test time, we estimate it by sampling a point cloud and measuring the diagonal.

Data Extraction. We use the ShapeNet-v1 [5] data and splits from Pixel2Mesh [81, 82], with the renders by Choy et al. [7]. For DDF training, per data type, we sample 20K (A and U) and 10K (S, B, T, and O) training samples per shape, using the watertight form of the meshes (via [24]), decimated to 10K triangles. These samples are only used for training, not evaluation, and are in a canonical aligned pose. We set the maximum offset size for O-type data as $\epsilon_O = 0.02$ (see §B); remaining parameters are the same as those used for single shapes (see §C).

Explicit Sampling Details. We also remark that our explicit sampling algorithm slightly oversamples points initially (when requesting a point set of size n_p, we actually sample $(1 + \varepsilon_p)n_p$ via $p \sim \mathcal{U}[\mathcal{B}]$). The final point cloud, however, is sorted by visibility (i.e., by $\xi(p, \widehat{v}^*(p))$) and only the top n_p points are returned. We used $\varepsilon_p = 0.1$ in all experiments. This is to prevent outputting non-visible points.

PC-SIREN Baseline Details. Recall that the PC-SIREN is our architecture-matched baseline, with identical encoders and a nearly identical decoder architecture to the

		DDF			
		Π_g-L	Π_g-S	$\widehat{\Pi}$-L	$\widehat{\Pi}$-S
Chairs	$D_C \downarrow$	0.477	0.532	0.861	0.928
	$F_\tau \uparrow$	54.37	47.26	46.81	40.35
	$F_{2\tau} \uparrow$	71.62	66.33	63.09	58.09
Planes	$D_C \downarrow$	0.201	0.231	0.748	0.799
	$F_\tau \uparrow$	80.69	76.71	63.86	60.48
	$F_{2\tau} \uparrow$	90.23	88.49	75.30	73.55
Cars	$D_C \downarrow$	0.235	0.309	0.545	0.628
	$F_\tau \uparrow$	68.21	57.36	59.84	49.98
	$F_{2\tau} \uparrow$	83.99	76.41	76.87	69.32

Table 3. Single-image 3D reconstruction results with $N_H = 1$.

DDF one. Here, the decoder is a mapping $f_b : \mathbb{R}^3 \to \mathbb{R}^3$, which is trained to compute $f_b(p) \in S$ from $p \sim [-1, 1]^3$, but uses an identical set of SIREN hidden layers as the DDF. A set of sampled random points can thus be mapped into a point cloud of arbitrary size. The baseline has 25,645,574 parameters, while the DDF-based model has 25,647,367 parameters. The camera loss $\mathcal{L}_\Pi$ is unchanged and the mask-matching loss $\mathcal{L}_M$ is not used. The shape-fitting loss $\mathfrak{L}_S$ is replaced with a standard Chamfer loss D_C [16], computed with 1024 points per shape with a batch-size of 32. The remaining aspects of training remain the same.

E.1. Ablation with $N_H = 1$

Recall that N_H is the number of times to "cycle" the points (projecting them towards the surface via the DDF) when sampling an explicit point cloud shape from a DDF (see §4.3). We show results with $N_H = 1$ in Table 3. In most cases, it is slightly worse than using $N_H = 3$, by 1-3 F-score units; occasionally, however, it is marginally better: on Planes-$\widehat{\Pi}$, it has slightly lower D_C, though this does not translate to better F-score.

E.2. Additional Visualizations

Some additional visualizations are shown in Fig. 12. For highly novel inputs, we also observe that, sometimes, the network does not adapt well to the shape (e.g., see the chairs example in the second row and second column). While much error is due to the incorrectly predicted camera, the DDF outputs can also be a bit blurrier, especially when it is uncertain about the shape. This can occur on thin structures, which are hard to localize (e.g., the chair legs in either row one and column one, or row three column two), or atypical inputs (e.g., row three, column two of the cars examples), where the network does not have enough examples to obtain high quality geometry. One can also see some non-uniform densities from our point cloud sampling algorithm (e.g., concentrations of points on the chair legs in column two, or on the wheels of several examples of cars).

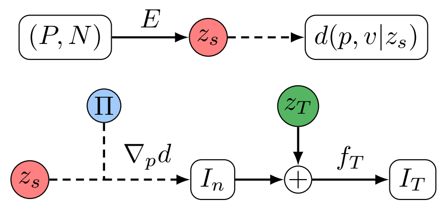

Figure 13. Two-stage unpaired generative modelling architecture. *Upper inset*: VAE formulation mapping a point cloud P and associated normals N to latent shape vector z_s via PointNet encoder E, and decoding into depth values via conditioning a PDDF. *Lower inset*: latent shape z_s and camera Π are randomly sampled and used to render a surface normals image I_n, via the derivatives of the learned conditional PDDF (see Property II). The normals map I_n is then concatenated ($\oplus$) with a sampled latent texture z_T, and used to compute the final RGB image $I_T = f_T(I_n, z_T)$. Coloured circles indicate random variables sampled from a particular distribution; dashed lines indicate computation with multiple forward passes (per point or pixel).

F. Generative Modelling

See Fig. 13 for a diagram of our two-stage 3D-to-2D modality translation architecture. See also Fig. 14 and 15 for additional sample visualizations (as in Fig. 9).

F.1. Shape VAE

The first stage learns a VAE [31, 62] on 3D shape, in order to (i) obtain a latent shape variable z_s that is approximately Gaussian distributed, and (ii) learn a CPDDF using z_s that can encode a shape in a form that is easily and efficiently rendered, yet still encodes higher order shape information. We learn a PointNet encoder [58] E that maps a point cloud with normals (P, N) to a latent shape vector $z_s = E(P, N)$. The conditional PDDF (CPDDF) then acts as the decoder, computing depth values as $d(p, v|z_s)$ for a given input. To implement conditional field computations, we use the modulated SIREN approach [44]. This can be trained by the following β-VAE loss [21]:

$$\mathfrak{L}_{\text{VAE}} = \mathfrak{L}_S + \beta \mathcal{L}_{\text{KL}}, \tag{43}$$

where $\mathcal{L}_{\text{KL}}$ is the standard KL divergence loss between a Gaussian prior and the approximate VAE posterior, and $\mathfrak{L}_S$ acts analogously to the reconstruction likelihood. We use ShapeNet cars [5] to fit the Shape VAE. Our goal is to show that we can rapidly train a model that can sample renders of surface normal geometry (from a unified 3D model), which is either difficult (or computationally costly) for most implicit approaches, or it uses a model directly on 2D surface normal images (missing a unified 3D shape model). These images could be useful for other downstream models, which may expect 2D input, but still propagate gradients to an underlying 3D object model.

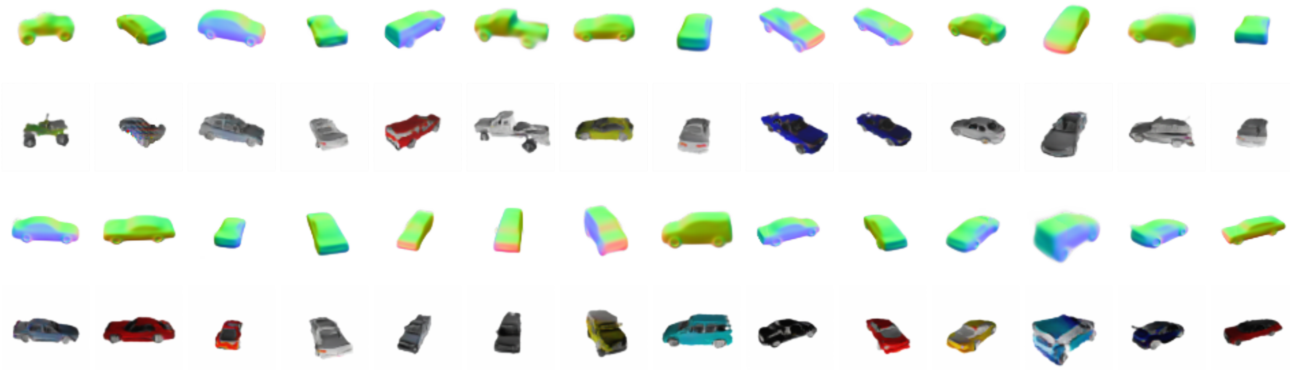

Figure 14. Additional example samples from the ShapeVAE and translational image GAN.

Figure 15. Additional example interpolations from the ShapeVAE and translational image GAN.

Data. Data is extracted from 1200 randomly chosen shapes from ShapeNet-v1 cars [5], sampling 60k (A and U) and 30K (S, B, T, and O) oriented points. We downsample shapes to 10K triangles before extraction. For minibatches, we sample 1.2K (A and U) and 300 (S, B, T, and O) oriented points per shape, with each minibatch containing 32 shapes. We also sample point clouds P (with normals N) of size 1024 to send to the PointNet at each minibatch.

Architectures. The PointNet encoder follows the standard classification architecture [58], with four `Conv1D-BatchNorm-ReLU` blocks (sizes: 64, 64, 128, 1024), followed by max pooling and a multilayer perceptron with `Linear-BatchNorm-ReLU` blocks (two hidden layers of size 512; dropout probability 0.1). No point transformers are used. The final output is of size $2\dim(z_s)$. We then compute the approximate variational posterior $q(z_s|P,N) = \mathcal{N}(z_s|\mu(P,N), \Sigma(P,N))$ with two networks for mean μ and diagonal log-variance matrix Σ, each structured as `Linear-ReLU-Linear` (in which all layers are of dimensionality $\dim(z_s)$; note that each posterior parameter network takes half of the vector output from the PointNet encoder as input).

For the decoder, we use eight layers `(512, 512, 512, 512, 256, 256, 256, 256)` (with the modulated SIREN [44]). We set $\dim(z_s) = 400$ and $\dim(z_T) = 64$.

Training. We run for 100K iterations, using $\beta = 0.05$, $\gamma_d = 5$, $\gamma_\xi = 10$, $\gamma_n = 10$, $\gamma_V = 1$, $\gamma_{\text{E,d}} = 0.1$, $\gamma_{\text{E},\xi} = 0.1$, $\gamma_{V,\xi} = 0.1$, and $\gamma_T = 0.1$. Adam is used for optimization (learning rate 10^{-4}; $\beta_1 = 0.9$, $\beta_2 = 0.999$).

F.2. Image GAN

After training a Shape VAE, the CPDDF decoder can be used to render a surface normals image I_n (see §3.2 and Fig. 5). To perform generation, we first sample latent shape $z_s \sim \mathcal{N}(0, I)$ and camera Π, which includes extrinsics (position and orientation) and focal length, following with normals map rendering to get I_n. We then use a convolutional network f_T to obtain the RGB image I_T, based on the residual image-to-image translation architecture from CycleGAN [92]. This is done by sampling $z_T \sim \mathcal{N}(0, I)$ and computing $I_T = f_T(I_n, z_T)$, where one concatenates z_T to each pixel of I_n before processing. Notice that z_s, z_T, and Π are independent, while the final texture (appearance) depends directly on z_T, and indirectly on z_s and Π, through I_n.

For training, a non-saturating GAN loss [14] with a zero-centered gradient penalty for the discriminator [45, 65] is used (as in [4, 51]). To ensure that f_T preserves information regarding 3D shape and latent texture, we use two consistency losses:

$$\mathcal{L}_{\text{C,S}} = \text{MSE}(I_n, \widehat{I}_n) \quad \& \quad \mathcal{L}_{\text{C,T}} = \text{MSE}(z_T, \widehat{z}_T), \quad (44)$$

where MSE is the mean squared error, $\widehat{I}_n = g_T(I_T)$ (with g_T having identical architecture to f_T), and $\widehat{z}_T = h_T(I_T)$ (with h_T implemented as a ResNet-20 [17, 22]). The first loss, $\mathcal{L}_{\text{C,S}}$, encourages the fake RGB image I_T to retain the 3D shape information from I_n through the translation process (i.e., implicitly forcing the image resemble the input normals) while the second loss, $\mathcal{L}_{\text{C,T}}$, does the same for the latent texture (implicitly strengthening textural consistency across viewpoints). The final loss for the GAN image generator is

$$\mathfrak{L}_{\text{GAN}} = \mathcal{L}_{\text{A}} + \gamma_{C,S}\mathcal{L}_{\text{C,S}} + \gamma_{C,T}\mathcal{L}_{\text{C,T}}, \qquad (45)$$

where $\mathcal{L}_{\text{A}}$ is the adversarial loss for the generator and the last two terms enforce consistency (see also [1, 28, 48]). We fit to the dataset of ShapeNet renders from Choy et al. [7]. Note that information on the correspondence with the 3D models is not used (i.e., the images and shapes are treated independently).

Generation process. Recall that our generation process can be written $I_T = G(z_s, z_T, \Pi) = f_T(I_n(z_s, \Pi) \oplus z_T)$, where $I_n(z_s, \Pi)$ denotes the CPDDF normals render (see §3.2) and $\oplus$ refers to concatenating z_T to every pixel of I_n. For latent sampling, $z_s, z_T \sim \mathcal{N}(0, I)$, while the camera Π is sampled from the upper hemisphere above the object, oriented toward the origin at a fixed distance and with a fixed focal length. The image size was set to 64×64.

Networks. The translation network f_T exactly follows the architecture from CycleGAN [92] consisting of residual blocks (two downsampling layers, then six resolution-retaining layers, followed by two upsampling layers), using the code from [93]. The normals consistency network, g_T, has identical architecture to f_T, while the texture consistency network, h_T, is a ResNet-20 [17, 22]. We utilize the convolutional discriminator implementation from Mimicry [36], based on DCGAN [59].

Training. Our image GAN is trained in the standard alternating manner, using two critic training steps for every generator step. The non-saturating loss [14] was used, along with a zero-centered gradient penalty [45, 65] (with a weight of 10 during critic training). We used the following loss weights: $\gamma_{C,S} = 1$ and $\gamma_{C,T} = 1$ For optimization, we use Adam (learning rate 10^{-4}; $\beta_1 = 0.0$, $\beta_2 = 0.9$) for 100K iterations (with the same reduce-on-plateau scheduler as in Appendix C).

2D GAN Comparison. As mentioned in the paper, we trained a convolutional GAN with the same loss and critic architecture using the Mimicry library [36]. We evaluated both our model and the 2D GAN with Frechet Inception Distance (FID), using torch-fidelity [53] with 50K samples.

References

[1] Tristan Aumentado-Armstrong, Alex Levinshtein, Stavros Tsogkas, Konstantinos G Derpanis, and Allan D Jepson. Cycle-consistent generative rendering for 2D-3D modality translation. In *2020 International Conference on 3D Vision (3DV)*, pages 230–240. IEEE, 2020. 1, 7, 8, 22

[2] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-NeRF: A multiscale representation for anti-aliasing neural radiance fields. *arXiv preprint arXiv:2103.13415*, 2021. 3

[3] Ruojin Cai, Guandao Yang, Hadar Averbuch-Elor, Zekun Hao, Serge Belongie, Noah Snavely, and Bharath Hariharan. Learning gradient fields for shape generation. In *European Conference on Computer Vision*, pages 364–381. Springer, 2020. 2

[4] Eric R Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-GAN: Periodic implicit generative adversarial networks for 3D-aware image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5799–5809, 2021. 1, 2, 5, 21

[5] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. `https://shapenet.org/`. Terms of use: `https://shapenet.org/terms`. 7, 8, 19, 20, 21

[6] Julian Chibane, Aymen Mir, and Gerard Pons-Moll. Neural unsigned distance fields for implicit function learning. *arXiv preprint arXiv:2010.13938*, 2020. 2, 6

[7] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3D-R2N2: A unified approach for single and multi-view 3D object reconstruction. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016. 7, 8, 19, 22

[8] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312, 1996. 16

[9] Aman Dalmia. dalmia/siren. *https://doi.org/10.5281/zenodo.3902941*, June 2020. Zenodo: 10.5281/zenodo.3902941. `https://github.com/dalmia/siren` (MIT License). 16

[10] Jun Gao, Wenzheng Chen, Tommy Xiang, Clement Fuji Tsang, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Learning deformable tetrahedral meshes for 3D reconstruction. *arXiv preprint arXiv:2011.01437*, 2020. 5

[11] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. FastNeRF: High-fidelity neural rendering at 200fps. *arXiv preprint arXiv:2103.10380*, 2021. 3

[12] Georgia Gkioxari, Jitendra Malik, and Justin Johnson. Mesh R-CNN. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9785–9795, 2019. 5

[13] Shubham Goel, Angjoo Kanazawa, and Jitendra Malik. Shape and viewpoint without keypoints. In *European Conference on Computer Vision*, pages 88–104. Springer, 2020. 1

[14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014. 8, 21, 22

[15] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. *arXiv preprint arXiv:2002.10099*, 2020. 5

[16] Thibault Groueix. Pytorch chamfer distance. *Thibault-GROUEIX/ChamferDistancePytorch*, Nov. 2021. `https://github.com/ThibaultGROUEIX/ChamferDistancePytorch` (MIT License). 20

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 6, 21, 22

[18] Peter Hedman, Pratul P Srinivasan, Ben Mildenhall, Jonathan T Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. *arXiv preprint arXiv:2103.14645*, 2021. 3

[19] Paul Henderson, Vagia Tsiminaki, and Christoph H Lampert. Leveraging 2D data to learn textured 3D mesh generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7498–7507, 2020. 1, 8

[20] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017. 8

[21] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. Beta-VAE: Learning basic visual concepts with a constrained variational framework. *International Conference on Learning Representations (ICLR)*, 2017. 7, 20

[22] Yerlan Idelbayev. Proper ResNet implementation for CIFAR10/CIFAR100 in PyTorch. `https://github.com/akamaster/pytorch_resnet_cifar10`. Accessed: Nov 2021. (BSD-2-clause License). 21, 22

[23] Eldar Insafutdinov and Alexey Dosovitskiy. Unsupervised learning of shape and pose with differentiable point clouds. *Advances in Neural Information Processing Systems*, pages 2807–2817, 2018. 7

[24] Krishna Murthy Jatavallabhula, Edward Smith, Jean-Francois Lafleche, Clement Fuji Tsang, Artem Rozantsev, Wenzheng Chen, Tommy Xiang, Rev Lebaredian, and Sanja Fidler. Kaolin: A PyTorch library for accelerating 3D deep learning research. *arXiv:1911.05063*, 2019. `https://github.com/NVIDIAGameWorks/kaolin` (Apache License). 19

[25] Yue Jiang, Dantong Ji, Zhizhong Han, and Matthias Zwicker. Sdfdiff: Differentiable rendering of signed distance fields for 3D shape optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1251–1261, 2020. 1, 2

[26] Kyungmin Jo, Gyumin Shim, Sanghun Jung, Soyoung Yang, and Jaegul Choo. Cg-nerf: Conditional generative neural radiance fields. *arXiv preprint arXiv:2112.03517*, 2021. 5

[27] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3D mesh renderer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3907–3916, 2018. 7

[28] Berk Kaya and Radu Timofte. Self-supervised 2d image to 3D shape translation with disentangled representations. In *2020 International Conference on 3D Vision (3DV)*, pages 1039–1048. IEEE, 2020. 7, 8, 22

[29] Petr Kellnhofer, Lars Jebe, Andrew Jones, Ryan Spicer, Kari Pulli, and Gordon Wetzstein. Neural lumigraph rendering. In *CVPR*, 2021. 2, 3

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

[31] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013. 7, 20

[32] Erwin Kreyszig. *Differential Geometry*. University of Toronto Press, 1959. Mathematical Expositions, No. 11. Dover edition. 15

[33] Venkat Krishnamurthy and Marc Levoy. Fitting smooth surfaces to dense polygon meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 313–324, 1996. 16

[34] Nilesh Kulkarni, Justin Johnson, and David F Fouhey. What's behind the couch? directed ray distance functions (drdf) for 3d scene reconstruction. *arXiv e-prints*, pages arXiv–2112, 2021. 2

[35] Tejas D Kulkarni, Will Whitney, Pushmeet Kohli, and Joshua B Tenenbaum. Deep convolutional inverse graphics network. *arXiv preprint arXiv:1503.03167*, 2015. 1

[36] Kwot Sin Lee and Christopher Town. Mimicry: Towards the reproducibility of GAN research. *CVPR Workshop on AI for Content Creation*, 2020. `https://github.com/kwotsin/mimicry` (MIT License). 22

[37] Chen-Hsuan Lin, Chaoyang Wang, and Simon Lucey. SDF-SRN: Learning signed distance 3D object reconstruction from static images. *arXiv preprint arXiv:2010.10505*, 2020. 5

[38] David B Lindell, Julien NP Martel, and Gordon Wetzstein. Autoint: Automatic integration for fast neural volume rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14556–14565, 2021. 3

[39] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3D reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7708–7717, 2019. 1, 4, 5

[40] Shichen Liu, Shunsuke Saito, Weikai Chen, and Hao Li. Learning to infer implicit surfaces without 3D supervision. *arXiv preprint arXiv:1911.00767*, 2019. 2

[41] Shaohui Liu, Yinda Zhang, Songyou Peng, Boxin Shi, Marc Pollefeys, and Zhaopeng Cui. Dist: Rendering deep implicit signed distance function with differentiable sphere tracing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2019–2028, 2020. 1, 2, 4

[42] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. 6, 18

[43] Julien NP Martel, David B Lindell, Connor Z Lin, Eric R Chan, Marco Monteiro, and Gordon Wetzstein. Acorn: Adaptive coordinate networks for neural scene representation. *arXiv preprint arXiv:2105.02788*, 2021. 8

[44] Ishit Mehta, Michaël Gharbi, Connelly Barnes, Eli Shechtman, Ravi Ramamoorthi, and Manmohan Chandraker. Modulated periodic activations for generalizable local functional representations. *arXiv preprint arXiv:2104.03960*, 2021. 6, 18, 20, 21

[45] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for GANs do actually converge? In *International conference on machine learning*, pages 3481–3490. PMLR, 2018. 21, 22

[46] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3D reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019. 1, 2

[47] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer, 2020. 1, 2, 4

[48] Yutaro Miyauchi, Yusuke Sugano, and Yasuyuki Matsushita. Shape-conditioned image generation by learning latent appearance representation from unpaired data. In *Asian Conference on Computer Vision*, pages 438–453. Springer, 2018. 7, 22

[49] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *arXiv preprint arXiv:2201.05989*, 2022. 8

[50] Thu Nguyen-Phuoc, Chuan Li, Stephen Balaban, and Yong-Liang Yang. RenderNet: a deep convolutional network for differentiable rendering from 3D shapes. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 7902–7912, 2018. 3, 15

[51] Michael Niemeyer and Andreas Geiger. Giraffe: Representing scenes as compositional generative neural feature fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11453–11464, 2021. 1, 21

[52] 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, 2

[53] Anton Obukhov, Maximilian Seitzer, Po-Wei Wu, Semen Zhydenko, Jonathan Kyl, and Elvis Yu-Jing Lin. High-fidelity performance metrics for generative models in PyTorch, 2020. Version: 0.3.0, DOI: 10.5281/zenodo.4957738 (Apache License). 8, 22

[54] Michael Oechsle, Songyou Peng, and Andreas Geiger. Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. *arXiv preprint arXiv:2104.10078*, 2021. 2, 3

[55] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019. 1, 2

[56] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alch'e Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 5

[57] Dario Pavllo, Graham Spinks, Thomas Hofmann, Marie-Francine Moens, and Aurelien Lucchi. Convolutional generation of textured 3D meshes. *arXiv preprint arXiv:2006.07660*, 2020. 1

[58] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3D classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017. 8, 20, 21

[59] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015. 22

[60] Daniel Rebain, Wei Jiang, Soroosh Yazdani, Ke Li, Kwang Moo Yi, and Andrea Tagliasacchi. DeRF: Decomposed radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14153–14161, 2021. 3

[61] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. KiloNeRF: Speeding up neural radiance fields with thousands of tiny mlps. *arXiv preprint arXiv:2103.13744*, 2021. 3

[62] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR, 2014. 7, 20

[63] Lukasz Romaszko, Christopher KI Williams, Pol Moreno, and Pushmeet Kohli. Vision-as-inverse-graphics: Obtaining a rich 3D explanation of a scene from a single image. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 851–859, 2017. 1

[64] Azriel Rosenfeld and John L Pfaltz. Distance functions on digital pictures. *Pattern recognition*, 1(1):33–61, 1968. 2

[65] Kevin Roth, Aurelien Lucchi, Sebastian Nowozin, and Thomas Hofmann. Stabilizing training of generative adversarial networks through regularization. *arXiv preprint arXiv:1705.09367*, 2017. 21, 22

[66] Vishwanath Saragadam, Jasper Tan, Guha Balakrishnan, Richard G Baraniuk, and Ashok Veeraraghavan. Miner: Multiscale implicit neural representations. *arXiv preprint arXiv:2202.03532*, 2022. 8

[67] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. Graf: Generative radiance fields for 3D-aware image synthesis. *arXiv preprint arXiv:2007.02442*, 2020. 1, 8

[68] Nicholas Sharp et al. Polyscope, 2019. `www.polyscope.run` v1.2.0. (MIT License). 16

[69] Daeyun Shin, Charless C Fowlkes, and Derek Hoiem. Pixels, voxels, and views: A study of shape representations for single view 3D object shape prediction. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3061–3069, 2018. 7

[70] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020. 5

[71] Vincent Sitzmann, Semon Rezchikov, William T Freeman, Joshua B Tenenbaum, and Fredo Durand. Light field networks: Neural scene representations with single-evaluation rendering. *arXiv preprint arXiv:2106.02634*, 2021. 3, 5

[72] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3D-structure-aware neural scene representations. *arXiv preprint arXiv:1906.01618*, 2019. 1, 2, 4

[73] Stanford Computer Graphics Laboratory. The Stanford 3D scanning repository. `http://graphics.stanford.edu/data/3Dscanrep/`. Accessed: 09/08/21. 16

[74] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3D shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11358–11367, 2021. 2

[75] Maxim Tatarchenko, Stephan R Richter, René Ranftl, Zhuwen Li, Vladlen Koltun, and Thomas Brox. What do single-view 3D reconstruction networks learn? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3405–3414, 2019. 7

[76] Shubham Tulsiani, Nilesh Kulkarni, and Abhinav Gupta. Implicit mesh reconstruction from unannotated image collections. *arXiv preprint arXiv:2007.08504*, 2020. 1

[77] Shubham Tulsiani, Tinghui Zhou, Alexei A Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2626–2634, 2017. 3, 7, 15

[78] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 311–318, 1994. 16

[79] Rahul Venkatesh, Tejan Karmali, Sarthak Sharma, Aurobrata Ghosh, R Venkatesh Babu, László A Jeni, and Maneesh Singh. Deep implicit surface point prediction networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12653–12662, 2021. 2

[80] Rahul Venkatesh, Sarthak Sharma, Aurobrata Ghosh, Laszlo Jeni, and Maneesh Singh. Dude: Deep unsigned distance embeddings for hi-fidelity representation of complex 3D surfaces. *arXiv preprint arXiv:2011.02570*, 2020. 2, 6

[81] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3D mesh models from single RGB images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 52–67, 2018. 7, 19

[82] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Hang Yu, Wei Liu, Xiangyang Xue, and Yu-Gang Jiang. Pixel2mesh: 3D mesh model generation via image guided deformation. *IEEE transactions on pattern analysis and machine intelligence*, 2020. 7, 19

[83] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *arXiv preprint arXiv:2106.10689*, 2021. 2

[84] Jiajun Wu, Yifan Wang, Tianfan Xue, Xingyuan Sun, Bill Freeman, and Josh Tenenbaum. Marrnet: 3D shape reconstruction via 2.5D sketches. *Advances in Neural Information Processing Systems*, 30:540–550, 2017. 3, 15

[85] Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. Perspective transformer nets: Learning single-view 3D object reconstruction without 3D supervision. *Advances in Neural Information Processing Systems*, 29:1696–1704, 2016. 3, 15

[86] Mingyue Yang, Yuxin Wen, Weikai Chen, Yongwei Chen, and Kui Jia. Deep optimized priors for 3D shape modeling and reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3269–3278, 2021. 5

[87] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. *arXiv preprint arXiv:2106.12052*, 2021. 2, 3

[88] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. *arXiv preprint arXiv:2103.14024*, 2021. 3

[89] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. PixelNeRF: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4578–4587, 2021. 1, 2

[90] Alan Yuille and Daniel Kersten. Vision as Bayesian inference: analysis by synthesis? *Trends in cognitive sciences*, 10(7):301–308, 2006. 1

[91] Jingyang Zhang, Yao Yao, and Long Quan. Learning signed distance field for multi-view surface reconstruction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6525–6534, 2021. 2

[92] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017. 8, 21, 22

[93] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Cyclegan and pix2pix in PyTorch. *junyanz/pytorch-CycleGAN-and-pix2pix*, Nov. 2021. `https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix` (BSD License). 22

[94] Jun-Yan Zhu, Zhoutong Zhang, Chengkai Zhang, Jiajun Wu, Antonio Torralba, Josh Tenenbaum, and Bill Freeman. Visual object networks: Image generation with disentangled 3D representations. *Advances in Neural Information Processing Systems*, 31:118–129, 2018. 7

[95] Ehsan Zobeidi and Nikolay Atanasov. A deep signed directional distance function for object shape representation. *arXiv preprint arXiv:2107.11024*, 2021. 2, 5