Supplementary Material
Single-Shot Scene Reconstruction

Sergey Zakharov[1], **Rareş Ambruş**[1], **Vitor Guizilini**[1], **Dennis Park**[1], **Wadim Kehl**[2], **Fredo Durand**[3], **Joshua B. Tenenbaum**[3], **Vincent Sitzmann**[3], **Jiajun Wu**[4], **Adrien Gaidon**[1]

[1]Toyota Research Institute, [2]Woven Planet, [3]Massachusetts Institute of Technology, [4]Stanford University

1 Implementation Details

In this section, we provide a detailed description of how we trained the different components of our pipeline, as well as what data was used. All our networks are implemented using PyTorch [1] and trained on a GPU cluster of 8 Tesla V100 GPUs.

1.1 Detection Transformer Block

We used a pretrained panoptic DETR [2] with a ResNet101 backbone initially trained on MS COCO [3] dataset. In particular, we used off-the-shelf pretrained weights for the backbone, transformer encoder and decoder, 2D bounding box head, and class label head.

1.2 3D Reasoning Block

The 3D reasoning block is trained using the AdamW [4] solver with learning rate of 10^{-5} and weight decay set to 10^{-4}.

NOCS Generation To train the foreground NOCS prediction head we first generate a collection of NOCS maps given ground truth object masks, object poses, and depth maps from a large synthetic dataset of road scenes by Parallel Domain [5]. The dataset contains poses and masks for each of the objects present in high-quality rendered RGB scenes (see Figure 1). The depth map is first transformed to a point cloud using given camera parameters and separate object instances are selected using object masks. Each instance is then transformed using the inverse pose and normalized using a common non-uniform scale per class - for vehicles we use a scale factor of $(3, 1.3, 1.1)$. A non-uniform scaling is applied to maximize the surface area and subsequently improve NOCS prediction. The transformed points are then stored in an image or 2D NOCS map by utilizing initial pixel indices.

Foreground NOCS Prediction and Pose Estimation Our NOCS head regresses tensors of size $H/4 \times W/4 \times C$, where C represents the number of unique colors of the correspondence map, which is set to 256 in our implementation. The resulting output channels store probability values for the class corresponding to the channel number. Then, we form NOCS maps consisting of U, V, and W channels, by aggregating output channels using resulting probability values - we first apply softmax along the channel dimension to ensure that all probabilities sum up to 1 and then multiply them by the channel index. Discrete color class classification problem proved to be useful for much faster convergence and for better overall correspondence quality and shape prediction when compared to direct regression, which can be explained by a significant simplification of the output solution space - discrete 256^3 for classification and infinite continuous solution space $[-1; 1]^3$ in the case of direct regression.

Our pose estimation is based on 2D-3D correspondence estimation. The procedure is defined as follows: Our NOCS head outputs normalized object coordinates (NOCS), mapping each RGB pixel to a 3D location on the object's surface. NOCS are then used to (i) recover a partial canonical

5th Conference on Robot Learning (CoRL 2021), London, UK.

Figure 1: Example renderings from the PD dataset used for SDN training.

shape of the detected object and multiplied by the class scale to recover real dimensions, and (ii) to establish correspondences between the recovered partial shape and image points. We then use a standard OpenCV P*n*P-RANSAC solver given estimated correspondences and a camera matrix to estimate the pose.

Amodal Background Prediction. To train the background depth prediction network, we generate a complementary dataset by rendering random objects onto the scene and then completing the occluded regions during the training as shown in Figure 4. For the RGB network we only estimate colors of the missing regions, while for the depth network we refine the geometry of the entire image (Figure. 3). Both networks use a standard ResNet [6] backbone. Upsampling is implemented using bilinear interpolation as opposed to deconvolution to decrease the number of optimization parameters. As for objective functions we use simple L1 reconstruction losses for both RGB and depth, and an additional surface normal loss for the depth domain comparing normals estimated using ground truth and predicted depth maps. The adversarial losses for both domains aim to make the predictions look more realistic. Both networks are trained for 100 epochs on the PD dataset. In the case of VKITTI2, we additionally refine the depth head for another 20 epochs.

1.3 PriorDB

Our differentiable database of object priors is first pretrained on a collection of 27 Parallel Domain geometrical primitives (Figure 2) to recover their SDF fields with high precision. All vehicles are normalized using a non-uniform scale factor of $(3, 1.3, 1.1)$ to maximize the surface area. The LF fields are then pretrained using RGB renderings (Figure 1) and generated NOCS maps. In particular, for each object instance we recover its partial shape using the NOCS map and use corresponding RGB colors as the learning target.

As opposed to the networks in the object reasoning block, both SDF and LF networks are implemented using SIREN-based [7] 6-layer MLPs with 512-dimentional hidden layers, which make use of periodic activation functions and are shown to be more capable than ReLU implicit representations at representing fine details. We train *PriorDB* using the Adam solver with a learning rate of 10^{-6}.

1.4 Surfel-based Rendering

The essential component of our surfel-based renderer is the formation of disc primitives. Similarly to [8], we construct the surface discs with the following steps:

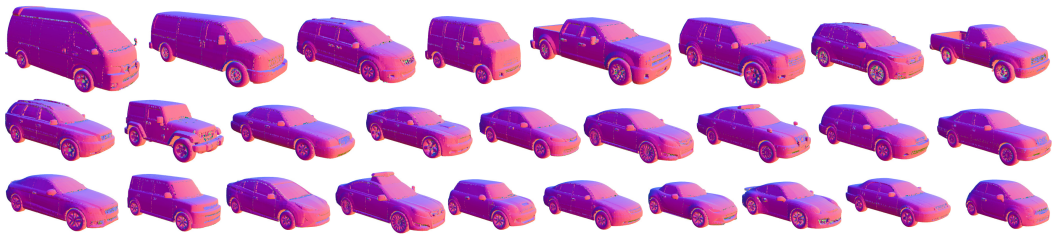


Figure 2: Object primitives used as prior shapes for *PriorDB*.

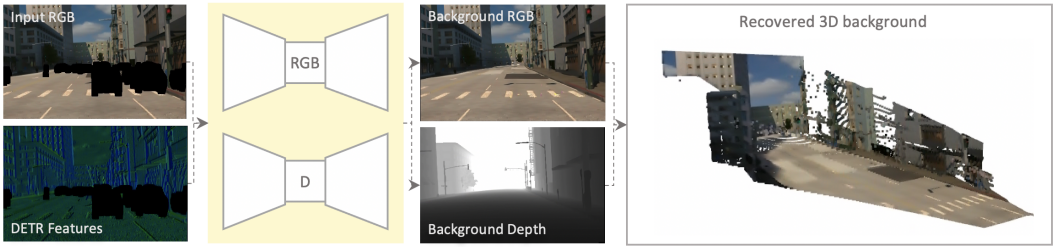


Figure 3: **Amodal Background Prediction:** Given an object mask predicted by SDN, we mask out the foreground of the input RGB images as well as predicted features. A concatenated tensor is then fed to the amodal background prediction nets, which recover the full background depth (including the occluded regions) and RGB colors for the occluded regions.

1. Given the normal of a projected point $n_i = \frac{\partial f(p_i;\mathbf{z})}{\partial p_i}$, we estimate the 3D coordinates of the resulting tangent plane visible in the screen. The distance d of the plane to each 2D pixel (u, v) can be computed by solving a system of linear equations for the plane and camera projection:

$$\begin{cases} u' = (u - o_u)\frac{d}{f_u} \\ v' = (v - o_v)\frac{d}{f_v} \\ Au' + Bv' + Cd - Au'_i - Bv'_i - Cd_i = 0 \end{cases} \tag{1}$$

The first two equations are the perspective projection equations and the third one is a plane equation. If we solve the above system by a simple substitution, we get the following:

$$\begin{aligned} d(\frac{A(u - o_u)}{f_u} + \frac{B(v - o_v)}{f_v} + C) \\ -Au'_0 - Bv'_0 - Cd_0 = 0 \longrightarrow \\ d = \frac{Au'_i + Bv'_i + Cd_i}{(\frac{A(u'-o_u)}{f_u} + \frac{B(v'-o_v)}{f_v} + C)} \\ = \frac{n_i \cdot p_i}{n_i \cdot \mathbf{K}^{-1}(u, v, 1)^T} \end{aligned} \tag{2}$$

where $\mathbf{K}^{-1}$ is the inverse camera matrix, followed by backprojection to get the final 3D plane coordinate:

$$P = \mathbf{K}^{-1} \cdot (u \cdot d, v \cdot d, d)^T. \tag{3}$$

2. Next, we estimate the distance between the plane vertex and surface point and clamp if it is larger than a disc diameter:

$$M = max(diam - ||p_i - P||_2, 0) \tag{4}$$

To ensure water-tightness we compute the diameter from the query location density: $diam = \min_{i \neq j} ||x_i - x_j||_2 \sqrt{3}$. Executing the above steps for each pixel yields a depth map $\boldsymbol{D}_i$ and a tangential distance mask $\boldsymbol{M}_i$ at point p_i.

Rendering Optimization Computing Eq. 2 and Eq. 3 for all image pixels can be extremely inefficient, especially for large image sizes. To address this issue, we limit computations to small pre-defined regions (7×7 in our case) centered at 2D positions of the surfels. This can easily be implemented using sparse tensors supported by PyTorch [1]. Once Eq. 2 and Eq. 3 are computed, we convert the collection of sparse tensors back to the dense format using known positions to regress a final rendering using our depth accumulation function as defined in the main paper.

To further optimize the performance, we avoid computing the above equations for all 3D points defining the object surface and limit the computations only to the points visible from the camera. To compute the set of visible points we utilize a hidden point removal (HPR) operator first introduced by Katz et al. in [9].

Figure 4: Example renderings from the PD dataset used for training background prediction: with instances (left) and without (right).

2 Optimization

In this section we provide further details on 2D and 3D optimization losses defined in Section 3.2 of the main paper. For each detected instance we run 100 iterations and use the SGD optimizer with a learning rate of 3×10^{-2} and a smaller learning rate of 10^{-5} for LF module weights.

3D losses We first use the predicted SDF/LF representation of the object to recover the full object surface in the form of a colored point cloud $\mathbf{c} = \{c_1, ..., c_k\}$ using a 0-isosurface projection. Then we estimate nearest neighbors between it and partial reconstruction based on predicted NOCS points $\mathbf{p} = \{p_1, ..., p_n\}$ and minimize the distance between the points by optimizing the feature vector z_{sdf}. At each iteration step nearest neighbors are recomputed and we only keep points that are closer than 0.2m to each other. The loss is then calculated as the mean distance over all correspondences C_{3D}.

$$L_{sdf} = \frac{1}{|C_{3D}|} \sum_{(i,j) \in C_{3D}} ||p_i - c_j||_1. \tag{5}$$

Our 3D Luminance Field loss is defined similarly to the SDF loss, but instead of point coordinates, we compare RGB values between the partial reconstruction based on predicted NOCS and input RGB colors against our recovered colored object point cloud.

$$L_{lf} = \frac{1}{|C_{3D}|} \sum_{(i,j) \in C_{3D}} ||p_i^{rgb} - c_j^{rgb}||_1. \tag{6}$$

Finally, our symmetry loss works by minimizing the difference between the left and right sides of our reconstructed colored object point cloud. Instead of finding nearest neighbors between two pointclouds, we perform the same operation for the left side of the car and inverted right side of the car, which equals to inverting the Y axis in our object reference frame as shown in Fig. 5.

$$L_{symm} = \frac{1}{|C_{3D}|} \sum_{(i,j) \in C_{3D}} ||l_i - r_j||_1.$$

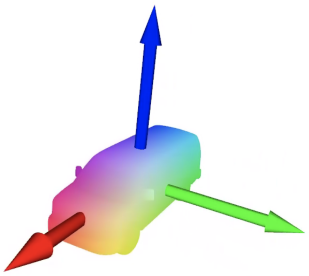


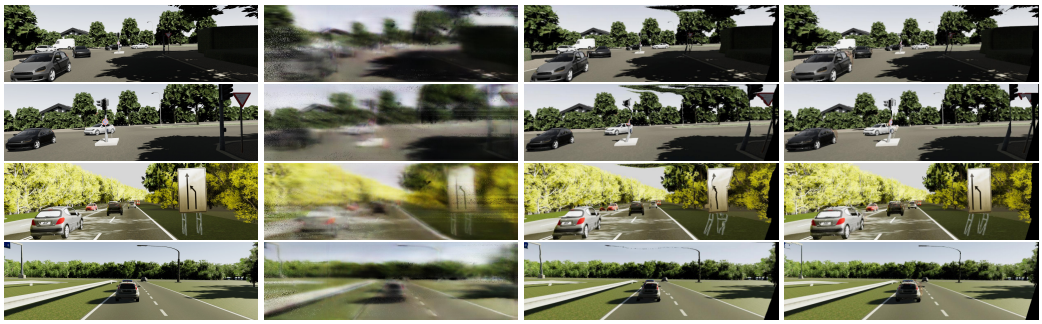
Figure 5: **Object reference frame**: X, Y, X axes are rendered as red, green, and blue arrows respectively. Mirroring the right side of the car for L_{symm} loss are achieved by simply inverting the Y axis.

The final 3D loss is a weighted sum of its components:

$$L_{3D}^{all} = w_{sdf} * L_{sdf} + w_{lf} * L_{lf} + w_{symm} * L_{symm}, \tag{7}$$

where in our implementation $w_{sdf} = 1$, $w_{lf} = 0.1$, and $w_{symm} = 0.1$.

2D losses Our surfel differentiable renderer allows to effectively define 2D losses for shape, pose, and appearance optimization. In our pipeline, we define three 2D losses: $L_{nocs_{2D}}$, $L_{lf_{2D}}$, and L_{mask}. All three are per-pixel loss functions comparing predicted and rendered 2D maps.



(a) GT (b) PixelNeRF (c) PackNet (d) Ours

Figure 6: **Qualitative view synthesis results.** The view synthesis is performed on images not seen during training. PixelNeRF trained on monocular sequences can not generalize to a novel view point. PackNet depth estimation method distorts objects due to imprecisions in predicted geometry. Our method, while retaining PackNet-like artifacts for the background, preserves the objects' geometry much better.

The 2D NOCS loss compares the predicted NOCS map coming from the NOCS head of the 3D reasoning block with the NOCS output of the renderer. Moreover, we limit the loss to the subset pixels contained in the union of predicted and rendered masks.

$$L_{nocs_{2D}} = \frac{1}{|P|} \sum_{p \in P} ||nmap_p^{pred} - nmap_p^{ren}||_2, \tag{8}$$

where $p \in P$ indexes all pixels contained in the union of predicted and rendered masks $m^u = m^{pred} \cup m^{ren}$ over all input images.

The 2D Luminance Field loss compares the input RGB image with the RGB output of the renderer

$$L_{lf_{2D}} = \frac{1}{|P|} \sum_{p \in P} ||lfmap_p^{pred} - lfmap_p^{ren}||_2. \tag{9}$$

Finally, the 2D mask loss compares the predicted mask coming from 3D reasoning block with the mask output of the renderer for the pixels contained in the union of both masks.

$$L_{mask_{2D}} = \frac{1}{|P|} \sum_{p \in P} ||mmap_p^{pred} - mmap_p^{ren}||_2. \tag{10}$$

Similarly to the 3D case, the final 2D loss is defined as a weighted sum of three 2D losses:

$$L_{2D}^{all} = w_{nocs_{2D}} * L_{nocs_{2D}} + w_{lf_{2D}} * L_{lf_{2D}} + w_{mask_{2D}} * L_{mask_{2D}}, \tag{11}$$

where in our implementation $w_{nocs_{2D}} = 1$, $w_{lf_{2D}} = 1$, and $w_{mask} = 0.1$.

3 Additional Evaluations

3.1 Pose/ Shape Optimization

We note that our initial pose, as regressed by an outlier-robust P*n*P solver, is usually already very accurate for a given NOCS prediction. However, since we optimize both pose and shape at the same time, allowing pose to change ensures the maximal alignment. To confirm this we ran an additional ablation for four 2D optimization modalities - no optimization, only shape, only pose, both pose and shape. We quantify the pose regressed using (i) the median angular error

	No opt.	Shape	Pose	Shape + Pose
Angle (deg)	5.99	5.99	6.55	**5.97**
Chamfer (mm)	14.72	9.90	14.72	**8.80**
Translation (m)	4.00	4.00	**3.98**	3.99

Table 1: **Optimization ablation.** The effect of optimization on 2 tasks: full object shape estimation (bidirectional Chamfer distance) and pose estimation.

Train	Method	Test	*Lower is better*				*Higher is better*		
			AbsRel	SqRel	RMSE	RMSE_{log}	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
VKITTI	Monodepth2	VKITTI	0.132	3.754	15.261	0.443	0.805	0.823	0.912
	PackNet-SfM		0.127	3.471	14.988	0.420	0.826	0.841	0.925
	Ours		**0.122**	**3.064**	**14.297**	**0.388**	**0.859**	**0.912**	**0.929**
	Monodepth2	KITTI	0.239	3.217	8.425	0.354	0.469	0.655	0.767
	PackNet-SfM		0.211	2.746	7.701	**0.319**	0.546	0.704	0.780
	Ours		**0.175**	**2.375**	**7.384**	0.360	**0.642**	**0.756**	**0.803**
PD	Monodepth2	PD	0.089	1.122	6.828	0.268	0.899	0.949	0.976
	PackNet-SfM		0.081	1.112	6.527	0.259	0.909	0.933	0.974
	Ours		**0.066**	**0.942**	**6.126**	**0.212**	**0.933**	**0.972**	**0.982**
	Monodepth2	DDAD	0.187	3.951	6.465	0.184	0.429	0.506	0.586
	PackNet-SfM		0.165	**2.500**	**6.017**	**0.165**	0.366	**0.591**	**0.620**
	Ours		**0.139**	3.808	6.612	0.231	**0.497**	0.559	0.590

Table 2: **In-Domain Depth estimation results** Our results suggest that our method does not just transfer between synthetic and real-world domains (as demonstrated in the main paper, Table 1), but also enables a better in-domain performance for foreground objects. For easy comparison, we also include out-of-domain estimation results from Table 1 of the main paper.

$\theta(q_i, \hat{q_i}) = 2 \cdot arccos(|q_i \cdot \hat{q_i}|)$ comparing the ground truth pose q and the optimized pose $\hat{q}$; and (ii) for translation, we estimate the difference between the ground truth translation and the optimized one considering only X and Y components. Additionally, we show the performance of shape estimation using the median bidirectional Chamfer distance to see how different modalities perform in this context. We performed this experiment on the PD *test* set.

3.2 View-Synthesis: Qualitative Results

We present qualitative results of the view synthesis experiment in Figure 6. To generate output images, we first render predicted hole-filled backgrounds by warping RGB values using predicted depth given respective camera transformation. Then, we transform extracted partial instances given camera poses and render them using our surfel-based renderer. As can be seen from the results, PixelNeRF trained on monocular sequences fails to generalize well to a stereo view-point providing a highly impaired noisy and blurry result. The PackNet-SfM baseline generates output by warping original RGB images using regressed full-scene depth maps. This baseline results in sharp images, but significantly distorts foreground objects due to predicted depth inaccuracy. Finally, our method preserves object geometry by using our differentiable renderer, yielding overall best results, but still retains PackNet-like artifacts on the background due to similar depth-estimation limitations.

3.3 In-Domain Depth Estimation

Table 2 extends the depth evaluation experiment presented in Section 4.4 of the main paper to demonstrate in-domain performance of our pipeline. The results suggest that our method does not just transfer between synthetic and real-world domains (as demonstrated in Table 2 of the main paper), but also enables a better in-domain performance for foreground objects. We attribute this behavior to the fact that depth networks operate at a per-pixel level, minimizing an objective that takes into consideration the entire image. This includes mostly background structures (e.g., road and buildings), making the depth network learn to accurately predict these structures at the expense of foreground objects. Our NOCS head, on the other hand, aims to only estimate partial shapes of objects of interest yielding superior results.

3.4 Failure Modes and Limitations

In this section we discuss common failure modes of the algorithm. Starting from the first stage of our pipeline, we can only reconstruct objects if we detected them - therefore false negatives are treated as background. However, in our scenes this usually happens when objects are significantly far away from the camera and representing them as background is a sufficiently good approximation. A second failure case that can affect distance scene objects is noisy NOCS map prediction. Erroneous NOCS maps directly affect the performance of P*n*P-based pose estimation. Despite using an outlier-robust RANSAC scheme, this still sometimes causes objects to be misaligned with respect to the

Figure 7: Example reconstructions on the PD dataset.

road surface and could even place objects behind the camera. While these problems can be resolved by manually estimating the road surface and aligning cars accordingly, usually in this case the quality of NOCS maps and, therefore, resulting partial shape reconstruction, does not allow for further optimization.

Another set of problems comes from the choice of the underlying background representation. While fairly robustly predicting geometry behind the detected objects, our method fails in the case of non-detected objects and structures limiting view-synthesis abilities. Moreover, luminance prediction of the regressed occluded surfaces does not always yield satisfactory results due to poorer generalization capabilities of the background module (when compared with NOCS prediction), which additionally impairs the visual quality. Finally, having a disjoint foreground-background representation sometimes causes alignment problems and self-occlusions, which could be resolved by representing background also as an implicit surface and allowing for further optimization.

Our *PriorDB* in its current implementation also has a number of limitations. First of all, we currently only consider average size vehicles, which restricts our ability to generalize to other types of shapes. Another limitation comes from the fact that *PriorDB* stores only rigid objects. We leave extensions to non-rigid objects as as well as other shape types as future work.

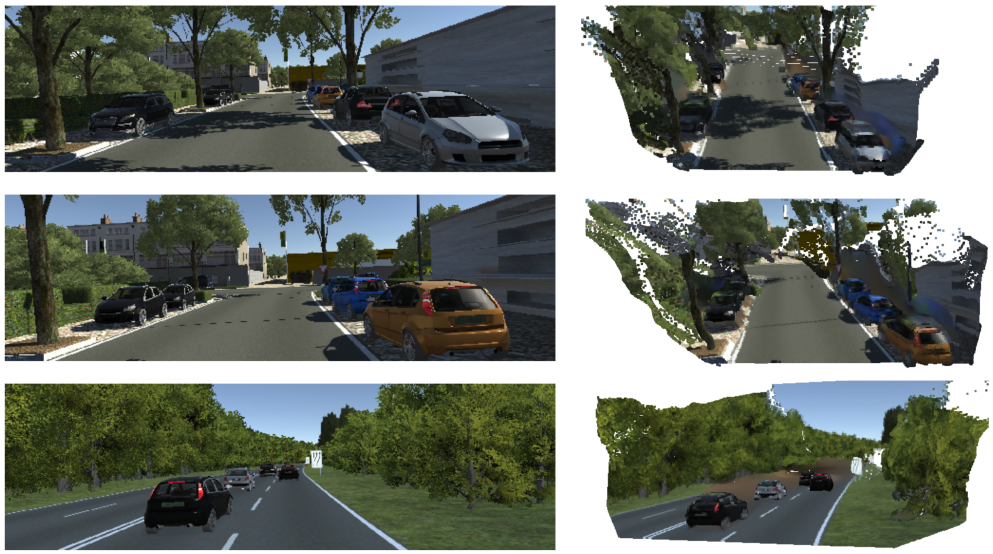


Figure 8: Example reconstructions on the VKITTI2 dataset.

Figure 9: Example reconstructions on the KITTI dataset.

3.5 Scene Reconstructions

Figures 7,8, and 9 provide a diverse set of reconstructions demonstrating the performance of our system on a variety of different scenes from different datasets.

References

[1] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[2] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end detection with transformers. In *ICCV*, 2020.

[3] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.

[4] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv*, 2017.

[5] Parallel domain. `https://paralleldomain.com/`, May 2021.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[7] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. *NeurIPS*, 2020.

[8] S. Zakharov, W. Kehl, A. Bhargava, and A. Gaidon. Autolabeling 3d objects with differentiable rendering of sdf shape priors. In *CVPR*, 2020.

[9] S. Katz, A. Tal, and R. Basri. Direct visibility of point sets. In *SIGGRAPH*. 2007.