Appendix

A. Proof of Equivalence

We show that the two optimization problems (5) and (7) are equivalent.

Lemma 5. *If p^* and f^* are the optimal values of the problems (5) and (7), respectively, then*

$$p^* = f^*. \tag{14}$$

Let $(\boldsymbol{Z}^*, \boldsymbol{h}^*, \boldsymbol{R}^*, \boldsymbol{t}^*)$ be the optimal solution to (7), and $\Delta = \boldsymbol{R}^*\boldsymbol{X} + \boldsymbol{t}^*\mathbf{1}_n^\mathsf{T} - \boldsymbol{Z}^*$, then $(\boldsymbol{Z}^* + \Delta, \boldsymbol{h}^*, \boldsymbol{R}^*, \boldsymbol{t}^*)$ is the optimal solution to (5).

Proof. First, note that the problem (7) is obtained from (5) by relaxing the hard equality constraint (*i.e.* $\boldsymbol{z}_i = \boldsymbol{R}\boldsymbol{x}_i + \boldsymbol{t}$). Therefore,

$$f^* \leq p^*. \tag{15}$$

Second, we are given that $(\boldsymbol{Z}^*, \boldsymbol{h}^*, \boldsymbol{R}^*, \boldsymbol{t}^*)$ is the optimal solution to (7). Let $\delta_i = \boldsymbol{R}^*\boldsymbol{x}_i + \boldsymbol{t}^* - \boldsymbol{z}_i^*$. Then,

$$\tilde{\boldsymbol{z}}_i = \boldsymbol{z}_i^* + \delta_i = \boldsymbol{R}^*\boldsymbol{x}_i + \boldsymbol{t}^*, \tag{16}$$

satisfies the equality constraint in (5). Furthermore,

$$f^* = \sum_{i=1}^{n} |f_d(\boldsymbol{R}^*\boldsymbol{x}_i + \boldsymbol{t}^* \mid \boldsymbol{h}^*)|^2 = \sum_{i=1}^{n} |f_d(\tilde{\boldsymbol{z}}_i \mid \boldsymbol{h}^*)|^2, \tag{17}$$

which implies that a feasible solution $(\tilde{\boldsymbol{Z}}, \boldsymbol{h}^*, \boldsymbol{R}^*, \boldsymbol{t}^*)$ in (5) attains the same objective value as f^*. Therefore, $p^* = f^*$. Note that

$$\tilde{\boldsymbol{Z}} = \boldsymbol{Z}^* + \Delta, \tag{18}$$

where $\Delta = [\delta_1, \delta_2, \ldots \delta_n]$, which equals $\boldsymbol{R}^*\boldsymbol{X} + \boldsymbol{t}^*\mathbf{1}_n^\mathsf{T} - \boldsymbol{Z}^*$. □

B. Architecture Details

Fig. 7 and Fig. 8 show the detailed architecture of our `CRISP`. We use the ViT-S variant of DINOv2 [29] as our ViT backbone, and keep it frozen during training.

The shape head (Fig. 7) is an MLP with layer normalization and ReLU activations. The network has two hidden layers, with hidden dimension equals to 512. We take the `[cls]` tokens from layers $(3, 6, 9, 12)$ as well as the patch tokens from layer 12, concatenate them into a 1920-dimensional vector and feed them into the shape head. The output of the shape head is a 2560-dimensional vector, which we use to condition the shape encoder. The shape decoder is a FiLM-conditioned MLP with sinusoidal activations [35, 41]. The network has 4 hidden layers, with hidden dimension equals to 256. The output of the shape head is split into 5 512-dimensional vectors, with each vector conditioning one layer of the shape decoder. The final layer is a linear layer with output dimension equals to 1.

The PNC head (Fig. 8) is based on DPT [37]. We take tokens from layers $(3, 6, 9, 12)$ and pass them independently through reassemble blocks (see [37] for details). We then pass the output of each reassemble blocks through fusion blocks respectively. We use projection as the readout operation and generate features with 256 dimensions. Note that different from the original DPT Implementation [37], we use group normalization in the residual convolutional unit in each of the fusion blocks. We note that this helps with the stability of the network in training. Lastly, we use a CNN to produce the PNC output. The CNN has 3 layers, with the first layer having 256 input channels and 128 output channels, the second layer having 128 input channels and 32 output channels, and the third layer having 32 input channels and 3 output channels. The kernel sizes of the first two layers are 3×3, and the kernel size of the third layer is 1×1. We use ReLU as the activation function.

Overall, `CRISP` uses 709 MB of VRAM with 39.1M params with our PyTorch implementation. Each shape code is 2560-dimensional (51.2 KB) — 5.12 MB for 1K objects, a small amount comparing to the model itself.

C. Shape Degeneracy Check

The linear least square shape corrector (10) informs a degeneracy check on the shape estimator, i.e., a check if the shape estimation is unique or not. From the shape corrector (10) objective, we have

$$\|\boldsymbol{F}(\boldsymbol{Z})\boldsymbol{D}\boldsymbol{c}\|^2 = \boldsymbol{c}^\mathsf{T}\boldsymbol{D}\boldsymbol{F}(\boldsymbol{Z})^\mathsf{T}\boldsymbol{F}(\boldsymbol{Z})\boldsymbol{D}\boldsymbol{c}. \tag{19}$$

It follows that if the matrix $\boldsymbol{F}(\boldsymbol{Z})^\mathsf{T}\boldsymbol{F}(\boldsymbol{Z})$ is singular then the shape corrector problem (10) will have multiplicity of solutions. Thus for a given input, we can check the singularity of the matrix $\boldsymbol{F}(\boldsymbol{Z})^\mathsf{T}\boldsymbol{F}(\boldsymbol{Z})$ to determine shape estimation uniqueness. Fig. 9 plots the minimum eigenvalue $\lambda_{\min}$ of the matrix $\boldsymbol{F}(\boldsymbol{Z})^\mathsf{T}\boldsymbol{F}(\boldsymbol{Z})$ as a function of keyframes N. In the example, as the handle of the mug begins to show up, the minimum eigenvalue $\lambda_{\min}$ increases drastically.

D. Training Details

D.1. Supervised Training Details

For PNC, we adopt a loss similar to the soft-L_1 used in [47]:

$$L_{PNC}(\boldsymbol{Z}, \boldsymbol{Z}^*) = \frac{1}{n}\sum_{i=1}^{N} \begin{cases} (\boldsymbol{z}_i - \boldsymbol{z}_i^*)^2/(2\zeta), & |\boldsymbol{z}_i - \boldsymbol{z}_i^*| \leq \zeta \\ |\boldsymbol{z}_i - \boldsymbol{z}_i^*| - \zeta/2, & |\boldsymbol{z}_i - \boldsymbol{z}_i^*| > \zeta \end{cases} \tag{20}$$

where ζ is the threshold to control switching between L_1 and quadratic loss. We use $\zeta = 0.1$ for all datasets.

For shape head and decoder, we adopt a loss similar to [41]:

$$\mathcal{L}_{SDF} = \frac{\gamma_1}{M_1} \sum_{i \in \Omega_0, |\Omega_0| = M_1} |f_d(\boldsymbol{x}_i \mid \boldsymbol{h}) - f_d(\boldsymbol{x}_i \mid \boldsymbol{h})^*| \tag{21}$$

$$+ \frac{\gamma_2}{M_2} \sum_{i \in \Omega_e, |\Omega_e| = M_2} \phi\left(f_d(\boldsymbol{x} \mid \boldsymbol{h})\right) d\boldsymbol{x}$$

$$+ \frac{\gamma_3}{M_1 + M_2} \sum_{i \in \Omega_0 \cup \Omega_e} |\|\nabla_{\boldsymbol{x}} f_d(\boldsymbol{x}_i \mid \boldsymbol{h})\| - 1| \tag{22}$$

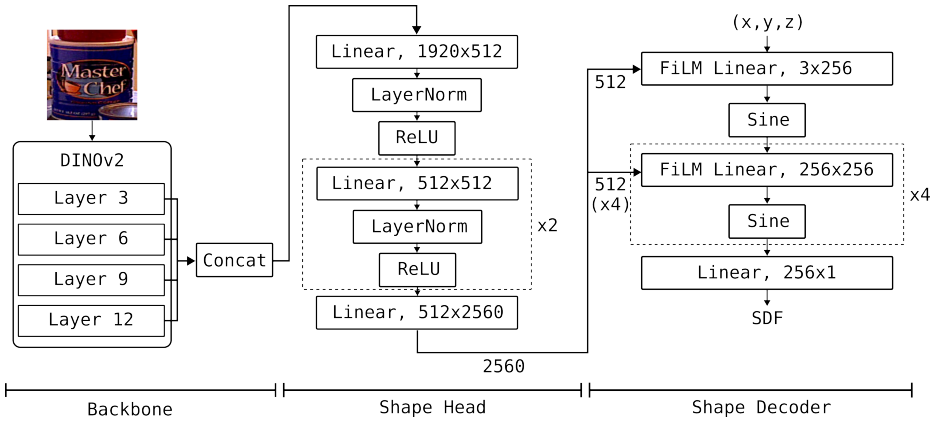


Figure 7. Diagram of our architecture for CRISP's shape head and shape decoder.

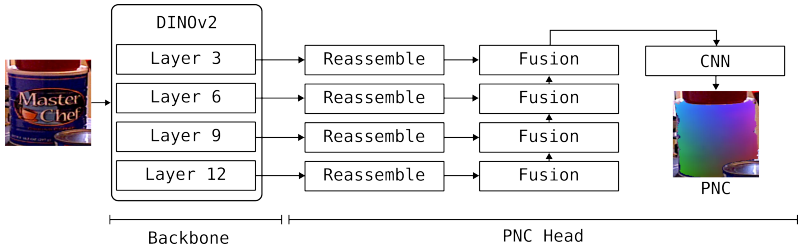


Figure 8. Diagram of our architecture for CRISP's PNC head.

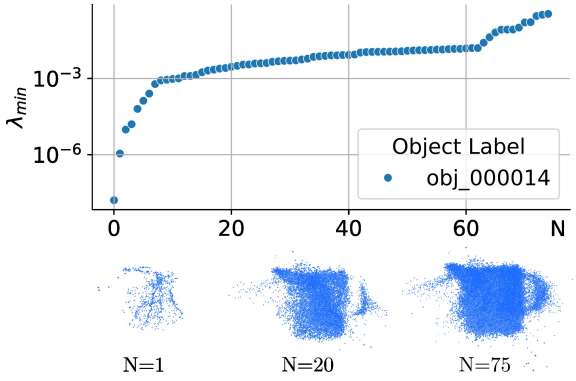


Figure 9. The minimum eigenvalue of the matrix $\boldsymbol{F}(\boldsymbol{Z})^{\mathsf{T}}\boldsymbol{F}(\boldsymbol{Z})$ as a function of keyframes N. Each keyframe captures the mug from a different viewing angle. $\boldsymbol{F}(\boldsymbol{Z})$ is computed using the estimated PNC $\boldsymbol{Z}$, aggregated over all keyframes till N.

where $\psi(\mathbf{x}) = \exp(-\alpha \cdot |\Phi(\mathbf{x})|)$ and $\alpha = 100$, Ω_0 is a set of points sampled on the shape manifold, equivalent to the zero-level set of SDF (with size M_1) and Ω_e is the set of points sampled outside the shape manifold, but inside the domain in which we want to reconstruct the SDF values (with size M_2). The first term utilizes supervision directly on the SDF values. The second term penalizes off-manifold points to have close-to-zero SDF values. The last term in (22) is the Eikonal regularization term where we ensure the norm of spatial gradients to be 1 almost everywhere in Ω. γ_1, γ_2, and γ_3 are hyperparameters. We compute the ground truth SDF values using `libigl`[2].

[2]https://github.com/libigl/libigl

The overall loss used is the following:

$$\mathcal{L}_{total} = \alpha\mathcal{L}_{PNC} + \beta\mathcal{L}_{SDF} \tag{23}$$

where α and β are hyperparameters.

YCBV Dataset For the YCBV dataset, we train two supervised models: `CRISP-Syn` and `CRISP-Real`. For `CRISP-Syn`, we train using synthetic rendered images of the YCBV objects using BlenderProc [9]. We generate a total of 4200 images, with 200 images per object. We use $\alpha = 5 \times 10^3$, $\beta = 0.1$, $\gamma_1 = 3 \times 10^3$, $\gamma_2 = 2 \times 10^2$, and $\gamma_3 = 50$. We train `CRISP-Syn` with an Adam optimizer and learning rate of 3×10^{-4} for 500 epochs. We use a batch size of 16 and weight decay of 10^{-5}.

For `CRISP-Real`, we train with the provided real-world train set images. We use $\alpha = 5 \times 10^3$, $\beta = 0.1$, $\gamma_1 = 3 \times 10^3$, $\gamma_2 = 2 \times 10^2$, and $\gamma_3 = 50$. We train `CRISP-Real` with an Adam optimizer with a learning rate of 3×10^{-4} for 50 epochs, We use a batch size of 10 and weight decay of 10^{-5}.

SPE3R Dataset We train `CRISP` on the train set of SPE3R. We use an Adam optimizer with a learning rate of 10^{-4}, weight decay of 10^{-6}, a batch size of 16 and for 100 epochs. We use a cosine annealing scheduler with restarts per 4000 iterations and a multiplication factor of 2. We use $\alpha = 50$, $\beta = 1$, $\gamma_1 = 3 \times 10^3$, $\gamma_2 = 2 \times 10^2$, and $\gamma_3 = 50$.

NOCS Dataset We train `CRISP` on the train sets of CAMERA and REAL275. We use an Adam optimizer with a learning rate of 10^{-4}, weight decay of 10^{-6}, a batch size of

16 and for 50 epochs. We use a cosine annealing scheduler with restarts per 4000 iterations and no multiplication factor. We use $\alpha = 50$, $\beta = 1$, $\gamma_1 = 3 \times 10^3$, $\gamma_2 = 2 \times 10^2$, and $\gamma_3 = 50$.

D.2. Self-Training Details

Correction For self-training, we use Alg. 1 and Alg. 2. There are two main components to both Alg. 1 and Alg. 2: the solver for $\boldsymbol{Z}$ (Line 2 in Alg. 1 and Alg. 2) and the solver for $\boldsymbol{h}$ (Line 3 in Alg. 1 and Alg. 2).

For the gradient descent solver for $\boldsymbol{Z}$ in both Alg. 1 and Alg. 2, we use a step size of 1×10^{-3} and a maximum number of iterations of 50 for all datasets.

For the solver for $\boldsymbol{h}$ in Alg. 1, we use projected gradient descent as described in Section 4.1 with a step size of 1×10^{-2} with a maximum number of iterations of 25 for all datasets. For the solver for $\boldsymbol{h}$ in Alg. 2 on the YCBV dataset, we additionally enforce $\boldsymbol{h}$ to be a one-hot vector. In addition, as described in Remark 3, we store $\boldsymbol{Z}$ in a buffer from multiple views. The buffer has a maximum size of 50 frames.

Certification We use (12) as a boolean indicator function to generate pseudo-labels. For the YCBV dataset, we use $\epsilon = 1 \times 10^{-2}$ and $p = 0.98$. For the SPE3R dataset, we use $\epsilon = 2 \times 10^{-2}$ and $p = 0.97$.

Self-Training We use (13) for the loss function for self-training. We use two separate stochastic gradient descent optimizers for the shape head and PNC head. For the shape head, we use a learning rate of 4×10^{-4} for YCBV and 3×10^{-4} for SPE3R. For the PNC head, we use a learning rate of 3×10^{-4} for both YCBV and SPE3R. We use a weight decay of 10^{-5} for both shape head and PNC head for all datasets. We use a batch size of 3 for the YCBV dataset and 10 for the SPE3R dataset.

E. Additional Results

Figure 10. Two sample images from our YCBV synthetic dataset.

Similar to Section 6, for tables, we use bold fonts for the best results and color the top three results with , and , respectively. For tables fewer than three entries, we label only the top entry with and bold font.

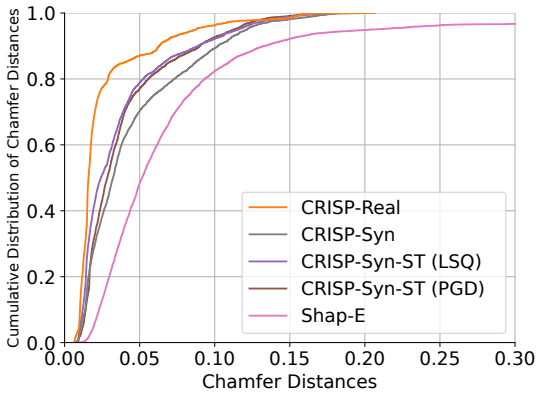


Figure 11. Cumulative distribution function of the Chamfer distances of the reconstructed shapes on the YCBV dataset.

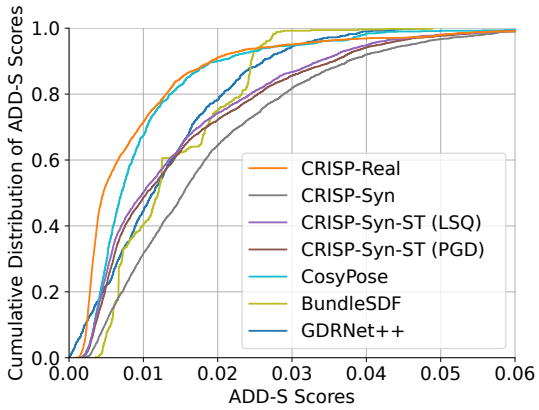


Figure 12. Cumulative distribution function of the ADD-S scores on the YCBV dataset.

E.1. YCBV Dataset

Fig. 10 shows two sample images from our synthetic dataset. Fig. 11 shows the cumulative distribution function (CDF) curves of Chamfer distance between the reconstructed shape and the ground truth CAD model. We note that `CRISP-Real` achieves the best performance, with self-training improving the shape reconstruction performance of `CRISP-Syn`. Fig. 12 shows the CDF curves of ADD-S scores. We note that `CRISP-Real` again achieves the best performance, with GDRNet++ achieves a slight performance edge at low ADD-S scores. Self-training improves the performance of `CRISP-Syn`, with `CRISP-Syn-ST (LSQ)` slightly outperforming `CRISP-Syn-ST (BCD)`.

E.2. SPE3R Dataset

Our corrector (Alg. 1) can be used to improve performance during inference as well, not only for self-training. Tab. 10 shows the performance of `CRISP` with and without the corrector (Alg. 1). With the corrector, we see improved shape reconstruction performance and pose estimation performance.

Methods	e_{shape} (mm) ↓						
	Bottle	Bowl	Camera	Can	Laptop	Mug	Avg
SPD [44]	3.44	1.21	8.89	1.56	2.91	1.02	3.17
SGPA [7]	2.93	0.89	5.51	1.75	1.62	1.12	2.44
CASS [6]	0.75	0.38	0.77	0.42	3.73	0.32	1.06
RePoNet [53]	1.51	0.76	8.79	1.24	1.01	0.94	2.37
CRISP	**0.55**	**0.31**	**0.37**	**0.18**	**0.51**	**0.16**	**0.35**

Table 8. Shape reconstruction results on the NOCS dataset.

	Methods	mAP						
		IoU_{25}	IoU_{50}	IoU_{75}	5° 5 cm	5° 10 cm	10° 5 cm	10° 10 cm
Category-level	NOCS [47]	**84.8**	78.0	30.1	10.0	9.8	25.2	25.8
	Metric Scale [21]	81.6	68.1	–	5.3	5.5	24.7	26.5
	SPD [44]	81.2	77.3	53.2	21.4	21.4	54.1	54.1
	CASS [6]	84.2	77.7	–	23.5	23.8	58.0	58.3
	SGPA [7]	–	80.1	61.9	39.6	–	70.7	–
	RePoNet [53]	–	81.1	–	**40.4**	–	**68.8**	–
	SSC-6D [34]	83.2	73.0	–	19.6	–	54.5	56.2
Category -agnostic	DualPoseNet [24]	–	76.1	55.2	31.3	–	60.4	–
	FSD [27]	80.9	77.4	61.9	28.1	**34.4**	61.5	**72.6**
	CRISP	84.2	**83.5**	**70.5**	22.5	23.7	62.8	64.8

Table 9. Pose estimation results on the NOCS dataset.

Method	$e^{L_1}_{shape}$ ↓		$e^{L_1}_{pose}$ ↓	
	Mean	Median	Mean	Median
CRISP	0.163	0.159	0.292	0.211
+ Corrector	**0.139**	**0.120**	**0.191**	**0.176**

Table 10. Evaluation of CRISP with and without the corrector (Alg. 1) on the SPE3R dataset.

Ablations	ADD-S (AUC) ↑ 1 cm	ADD-S (AUC) ↑ 2 cm	ADD-S (AUC) ↑ 3 cm
Normalized $\boldsymbol{Z}$ (NOCS)	8×10^{-4}	0.036	0.095
Unnormalized $\boldsymbol{Z}$ (PNC)	**0.22**	**0.42**	**0.54**

Table 11. Comparing pose estimation performance between normalized $\boldsymbol{Z}$ (NOCS) with PNC for self-training on the YCBV dataset.

E.3. NOCS Dataset

Tab. 8 shows the shape reconstruction results on the NOCS dataset for all categories. CRISP outperforms the baselines in all categories as well as the average in terms of e_{shape}. Tab. 9 shows the pose estimation results on the NOCS dataset. As noted in Section 6.3, CRISP achieves comparable performance with the state-of-the-art category-agnostic methods. Note that CRISP's rotation error performance can be potentially improved by incorporating more advanced loss functions such as the symmetry-aware loss used in [47].

E.4. Qualitative Results

We show qualitative results on the YCBV, SPE3R and NOCS datasets in Fig. 13. Empirically, we observe that shape reconstruction is better if objects are consistently aligned — CRISP sometimes confuses the two clamps in YCBV which are offset by 90°.

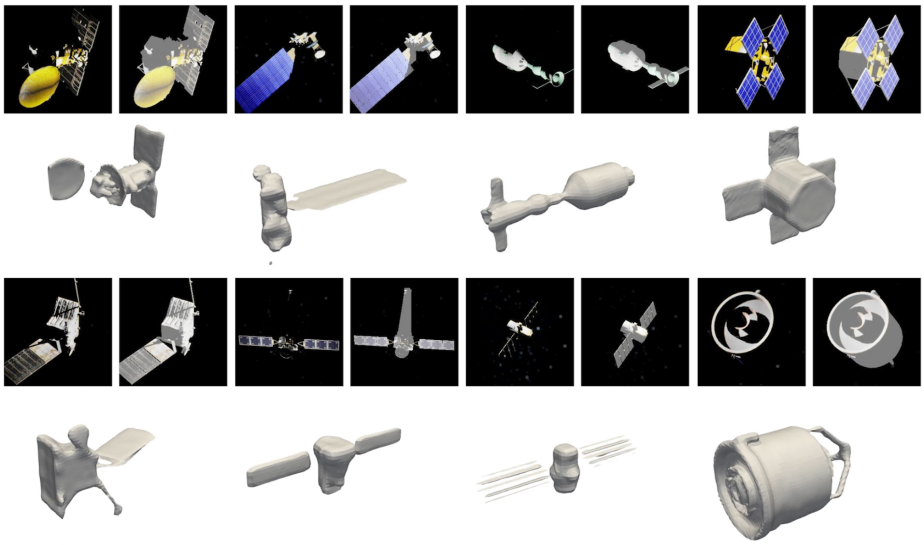
E.5. Additional Ablation Experiments

Scale Degeneracy. Comparing with NOCS [47], we see that PNC achieves significantly better pose estimation performance through the prevention of scale degeneracy (Tab. 11). To understand why the scale degeneracy is a problem, consider the pose optimization problem used with NOCS:

$$(\hat{s}, \hat{\boldsymbol{R}}, \hat{\boldsymbol{t}}) = \underset{(s,\boldsymbol{R},\boldsymbol{t})}{\operatorname{argmin}} \sum_{i=1}^{n} \|s\boldsymbol{R}\boldsymbol{x}_i + \boldsymbol{t} - \boldsymbol{z}_i\|^2 . \quad (24)$$

Assume noiseless depth $\{\boldsymbol{x}_i\}$, for a given $\boldsymbol{z}_i$, we have $(\hat{s}, \hat{\boldsymbol{R}}, \hat{\boldsymbol{t}})$ as the solution to (24). If we scale $\boldsymbol{z}_i$ by a factor of b, we can achieve the same loss for (24) if we let $s = b\hat{s}$, $\boldsymbol{R} = \hat{\boldsymbol{R}}$, and $\boldsymbol{t} = b\hat{\boldsymbol{t}}$. In practice, we observe such phenomenon (see Fig. 14): after self-training, the predicted z_i becomes significantly smaller than the ground truth z_i (Fig. 14a). If we transform the CAD model using the estimated pose, we observe that the transformed CAD model is much larger than the ground truth CAD model (Fig. 14b). We note that this might be one of the reasons why prior works [27, 34] need to keep access to synthetic data during

(a) Qualitative results on the YCBV dataset.

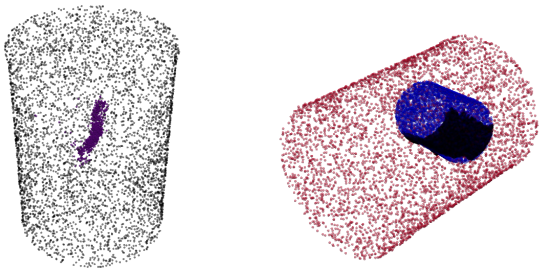


(b) Qualitative results on the SPE3R dataset.

(c) Qualitative results on the NOCS dataset.

Figure 13. Qualitative results on the YCBV, SPE3R and NOCS datasets. For each example, we show the input RGB image (*top left*), masked RGB image (*top right*) and reconstructed meshes transformed with estimated transformations (*bottom*).

self-supervised training.



(a) (b)

Figure 14. (a) *Purple*: $\boldsymbol{Z}$ predicted by the PNC head after self-training if we adopt normalization for $\boldsymbol{Z}$ (equivalent to NOCS [47]), *Black*: ground truth CAD model. (b) *Blue*: ground truth transformed CAD model. *Black*: depth point cloud. *Red*: CAD model transformed using estimated pose with normalized $\boldsymbol{Z}$.

Use of DINOv2 as a backbone. We use DINOv2 as the backbone of `CRISP` due to its highly generalizable features. To validate this, we perform an ablation study where we randomly initialize the backbone weights, while keeping the same architecture, and train on the synthetic data. We then test the model performance on the real YCBV test set, and comparing that with `CRISP-Syn`. We observe that with randomly initialized weights, both the shape reconstruction and pose estimation performance drastically degrade comparing to using DINOv2 weights (Table 12).

Ablations	ADD-S (AUC) ↑ 2 cm	e_{shape} (AUC) ↑ 5 cm
Randomly Initialized	0	0.075
`CRISP-Syn` (DINOv2)	**0.42**	**0.39**

Table 12. Ablation studies on the use of DINOv2 as a backbone.

Use of $\boldsymbol{h} = f_e(\mathcal{I})$ in (9). The use of $\boldsymbol{h} = f_e(\mathcal{I})$ in (9) can be seen as a way to incorporate the shape encoder's output into Alg. 2. We empirically observe that doing so improves the shape reconstruction performance after self-training (see Tab. 13).

Ablations	e_{shape} (AUC) ↑ 3 cm	e_{shape} (AUC) ↑ 5 cm	e_{shape} (AUC) ↑ 10 cm
Without $\boldsymbol{h} = f_e(\mathcal{I})$	0.21	0.35	0.55
With $\boldsymbol{h} = f_e(\mathcal{I})$	**0.25**	**0.43**	**0.65**

Table 13. Comparing shape reconstruction performance between using and not using $\boldsymbol{h} = f_e(\mathcal{I})$ in (9) on the YCBV dataset.

Normalization with $\boldsymbol{D}$. As noted in Section 4.2, we apply normalization to $\boldsymbol{F}$ through the use of the diagonal matrix $\boldsymbol{D}$. To be more precise, we calculate d_k (the diagonal entries of $\boldsymbol{D}$) as the inverse of the diameter of the bounding box for each $f_d(\cdot \mid \boldsymbol{h}_k)$ in (9). Empirically, this improves the shape reconstruction performance after self-training (see Tab. 14).

Ablations	e_{shape} (AUC) ↑ 3 cm	e_{shape} (AUC) ↑ 5 cm	e_{shape} (AUC) ↑ 10 cm
Without normalization	0.23	0.40	0.63
With normalization	**0.25**	**0.43**	**0.65**

Table 14. Comparing shape reconstruction performance between using and not using $\boldsymbol{D}$ for normalization on the YCBV dataset.