Interactive Annotation of 3D Object Geometry using 2D Scribbles

Tianchang Shen[1,2⋆], Jun Gao[1,2,3], Amlan Kar[1,2,3], and Sanja Fidler[1,2,3]

[1]University of Toronto, [2]Vector Institute, [3]Nvidia
{shenti11, jungao, amlan, fidler}@cs.toronto.edu

In the supplementary material, we provide more details about our model and training, as well as additional qualitative and quantitative results. We first show the detailed structure of our backbone network in Section 1.1, then present training details for SIM (Section 1.2), Vox2Mesh (Section 1.3) and PIM (Section 1.4). For experimental results, we first provide details on evaluation metrics in Section 2.1, show additional qualitative examples for both SIM and PIM, as well as failure cases on ShapeNet in Section 2.2. Performance in terms of other evaluation metrics is shown in Section 2.3. Details of the user study are provided in Section 2.4. More alternative methods for shape generation are discussed in Section 2.5.

1 Model and Training Details

1.1 Network Architectures

GenRe: We provide a brief overview of GenRe and its illustration in Figure 1, and refer readers to the original paper for details. GenRe is composed of three modules. The first module predicts a depth map from an RGB image (cropped to contain a single object). In case depth is already available (such as depth obtained from a 3D sensor), one can omit this step without changing the rest of the model. The second module converts the depth map into a spherical map and inpaints the missing depth information with a 2D-UResNet. The third module back-projects the spherical map into a 3D occupancy grid, which is further refined to a volume of size 128^3 using a 3D-UResNet. In our experiments, we use the pretrained model provided by the official GitHub repo[1]. Details about the 2D-UResNet and 3D-UResNet are provided below.

2D-UResNet: The 2D-UResNet adopts Resnet-18 to encode the input, which can be either a 2D spherical map, scribble map or a projected 2D feature map. The decoder is composed of consecutive transposed convolution layers, which take both, the output from the previous layer and the corresponding feature map from the encoder as input. The output of the decoder is a one dimensional feature map with the same resolution as input. When multiple outputs are produced (*e.g.*, scribble's 2D-UResNet outputs the refined silhouette, attention and correction features), each prediction branch shares the same encoder but uses a different decoder.

⋆ equal contribution

[1] https://github.com/xiumingzhang/GenRe-ShapeHD

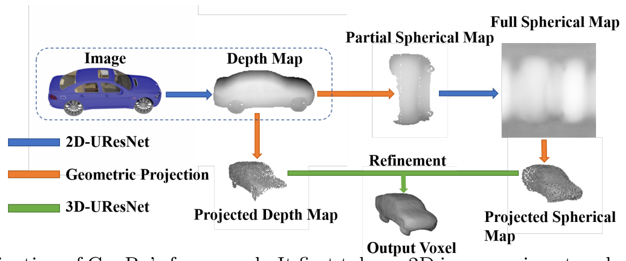


Fig. 1: Visualization of GenRe's framework. It first takes a 2D image as input and uses a 2D-UResNet to predict a depth map, which is further converted to a spherical map and inpainted with another 2D-UResNet. The spherical and depth maps are then projected into 3D, and a 3D-UResNet is used to refine the shape and predict the occupancy grid.

3D-UResNet: The 3D-UResNet leverages the same design concept as the 2D version, but consists of repeated 3D convolution and transposed convolution layers.

1.2 Scribble Interaction Module

Simulation of User Corrections: Since it is hard to collect large-scale human-in-the-loop data required for our scenario, we resort to **simulation** by exploiting synthetic 3D models. Synthetic data gives us perfect alignment (and thus ground-truth) between 3D shapes and their rendered images. The first step in our simulation is to simulate a proper annotation viewpoint, that is most informative to annotate during training. When the overall predicted shape is of low quality, we found that annotation in canonical views (front, side and top view) is most effective. For axis-aligned objects, canonical views contain many self-occlusions. Thus, a scribble in 2D affects a lot of the 3D volume, resulting in dramatic changes to the shape. When the predicted object shape is of better quality but missing fine details, annotating in a view with the least self-occlusion helps in order to reduce the depth ambiguity. We sample both types of viewpoints during simulation. GenRe predicts 3D shapes in a viewer-centric manner, thus the object pose (wrt canonical pose) is unknown. We thus use PCA to find the major axes of the object and obtain three orthogonal views. We also sample three diagonal views in between to increase the probability of exposing erroneous shape. From these six views, we compute IoU score between the projected object silhouette and the ground-truth silhouette, and select the view with the worst score.

Given the predicted and ground truth silhouettes, we follow [9] to **simulate scribbles**. Error regions are computed by subtracting the two masks, and a series of morphological transformations are applied on the error areas. For training efficiency, we synthesize multiple scribbles in each view and update the shape using their union. At training time, the probability of a scribble being synthesized is proportional to the area of the corresponding false region. At test time, We use the same method to choose the viewpoint as in training, and simulate scribbles by choosing the largest false negative or false positive region.

Loss Function: We use binary classification loss for both the 3D occupancy grid, as well as 2D masks. To encourage SIM to produce predictions consistent with the corrected 2D view, we add a re-projection consistency loss between the predicted 3D shape and ground-truth 2D silhouette in the correction view. This avoids repeated predictions errors at the same location in the corrected view.

Training Details: Our SIM was trained with a batch size of 2 for 3 correction steps, with maximum 3 positive and 3 negative scribbles in each step. The input scribbles are drawn on an image plane with resolution 128×128, which equals the occupancy grid resolution. The widths of synthesized scribbles at each step are 4, 3 and 2 pixels, respectively. Note that at inference time, the scribble width also decreases by 1 pixel after each correction step, but stops at a minimum of 2 pixels.

We use the Adam optimizer with learning rate $= 3\times10^{-3}$, $\beta_1 = 0.5$ and $\beta_2 = 0.9$. The maximum number of training iterations is set to be 300k.

1.3 Vox2Mesh

To convert the predicted occupancy grid to a triangular mesh, we utilize a pipeline similar as in [3,12]. Specifically, we first produce an initial mesh using the *CUBIFY* operation, which divides faces of each occupied cell into two triangles. Shared vertices and edges are then merged, while internal faces are removed. Suppose that the mesh has N vertices and its topology is denoted as G. We refine the mesh using a Graph Neural Network (GCN), which predicts the offset of each vertex.

To be specific, we first feed the predicted occupancy grid after the scribble refinement to another 3D-UResNet. We then apply *CUBIFY* to get the mesh. For every vertex p_i in the output mesh, we extract a feature f_i from the first and last convolutional layers via bi-linear interpolation using vertex positions. The GCN then takes the concatenation of the features and vertex positions as input, and predicts the offset for each vertex:

$$f_i' = \text{concat}(f_i, p_i) \tag{1}$$

$$\Delta p_1, \Delta p_2, \cdots, \Delta p_N = \text{GCN}(f_1', f_2', \cdots, f_N'; G). \tag{2}$$

We use a similar network architecture as in [5] for GCN with a slight modification, which appends the features from the previous iteration step to the next step, while [5] only feeds in the vertex position. The state dimension in GCN is set to 128.

Loss Function: We train our GCN model with Chamfer distance, normal loss and the edge regularization loss as in [10,12]. The Chamfer loss is computed by comparing uniformly sampled points from the predicted and the ground truth mesh:

$$L_{chamfer} = \sum_{p \in pred} \min_{q \in GT} ||p - q||_2^2 + \sum_{q \in GT} \min_{p \in pred} ||q - p||_2^2. \tag{3}$$

We further add the normal loss as in [12], where:

$$L_{normal} = \sum_{p} \sum_{k \in \mathcal{N}(p)} ||(p-k) \cdot n_q||_2^2 \tag{4}$$

We additionally explored the *scale-dependent umbrella operator* [2] for its robustness to different edge lengths:

$$E = \sum_{k \in \mathcal{N}(p)} ||p-k||_2 \tag{5}$$

$$L_{lap} = \left\| \sum_{p} \sum_{k \in \mathcal{N}(p)} \frac{2(p-k)}{E \cdot ||p-k||_2} \right\|_2 , \tag{6}$$

where p is a predicted vertex and $\mathcal{N}(p)$ is the set of neighbouring vertices of p. Edge regularization loss is computes as:

$$L_{edge} = \sum_{e} ||e^s - e^t||_2, \tag{7}$$

where e are edges in the mesh and e^s, e^t are starting and ending points in the edge, respectively.

The overall loss function is a weighted sum of the three above terms:

$$L = L_{chamfer} + \lambda_1 L_{lap} + \lambda_2 L_{edge} + \lambda_3 L_{normal} \tag{8}$$

Training Details: To convert the predicted occupancy grid to a mesh, we first use a threshold of 0.8 to binarize the grid, followed by downscaling to size $48 \times 48 \times 48$. The downscaling is needed to reduce the number of vertices in GCN. Since we do not downscale the skip-feature, we argue the information is preserved. We then apply the *CUBIFY* operation on the downscaled occupancy grid to obtain the initial mesh. Vox2mesh was trained with a batch size of 2 for 3 correction steps. We use the Adam optimizer with learning rate $= 3 \times 10^{-5}$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The weights for loss terms are set as $\lambda_1 = 0.0005$, $\lambda_2 = 0.5$, $\lambda_3 = 0.01$. We use a weight decay of 0.0001.

1.4 Point Interaction Module

Simulation of User Corrections: Following [5], we simulate the annotator by selecting the worst predicted vertex (largest L2 distance to the GT mesh), and simulate the drag-and-drop operation by moving it to the closest ground truth vertex, q_i.

Training Details: We use the same network architecture and training procedure as for the GCN in Vox2Mesh to train our PIM module with a slight modification. We decrease the weight for regularization loss to encourage movement of vertices. More specifically, we set $\lambda_1 = 5 \times 10^{-5}$, $\lambda_2 = 0.25$, $\lambda_3 = 0.005$. We train PIM with 3 point annotations in 3 separate steps, and affective length l is set to 3.

2 Experiments

2.1 Metrics

We use the Chamfer Distance, F1-score and the normal consistency score to evaluate our method, following [1,3,8,12,15]

The Chamfer Distance is computed as in Eq (3). The F1 score is calculated as

$$F_1 = 2 \cdot \frac{\text{completeness} \cdot \text{accuracy}}{\text{completeness} + \text{accuracy}} \tag{9}$$

where completeness is the percentage of points sampled from the ground truth mesh for which at least one sampled point from prediction is within a distance of 0.01. Similarly, accuracy is the percentage of points sampled from prediction for which at least one sampled point from ground truth is within the threshold.

The normal consistency loss, different from Eq (4), is calculated as:

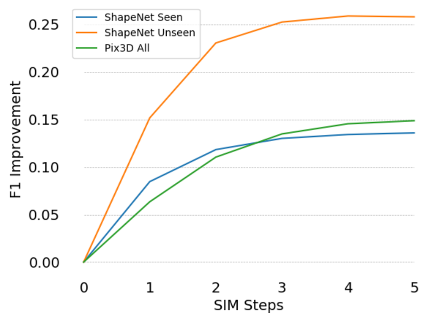
$$Normal\_Consistency = \sum_{p \in pred} |u_p \cdot u_q| / N, \tag{10}$$

where q is the closest ground truth point to p. Here, u_p and u_q are unit normal vectors of p and q, and N is the total number of points sampled from the predicted mesh, which is 10k in our case.

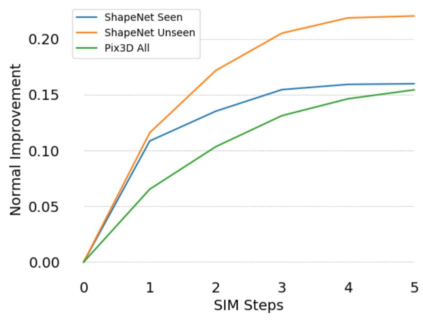
Chamfer Distance and F1-score evaluate first-order alignment to ground-truth surfaces and normal consistency evaluates how well our model reconstructs higher-order information. GenRe [15] focuses on the surface only (which can be incomplete) and does not evaluate Intersection-Over-Union, which we follow. We uniformly sample 10,000 points from the surface of both reconstructed and ground-truth meshes to evaluate these metrics. For a fair comparison, we convert volumetric prediction to mesh using the same Marching Cube [6] algorithm before sampling. We use different thresholds to binarize our and GenRe's [15] volumetric prediction. Ours uses a fixed threshold of 0.75 unless otherwise stated, while for GenRe, we search for the optimal threshold on the training classes. All metrics except for the normal consistency score depend on the scale of the object. Therefore, to evaluate Chamfer Distance and F1 Score, we first rotate all objects to be axis-aligned using ground-truth camera position, then normalize the sampled point cloud such that the longest dimension equals to one and the center is at zero. The F1 score is reported with the threshold equal to 0.01.

2.2 Additional Qualitative Examples on ShapeNet

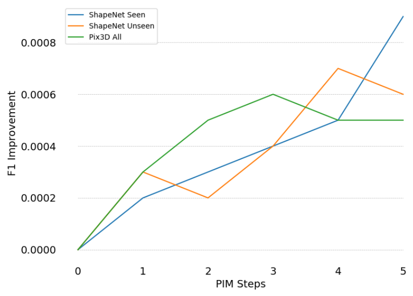
SIM Examples: We show more examples of SIM in action in Fig. 3. In addition, we visualize the simulated annotation process on ShapeNet images and ScanNet scans in *simulation_tool.mp4*. The scribble generation and view selection are simulated as we described in the main paper.



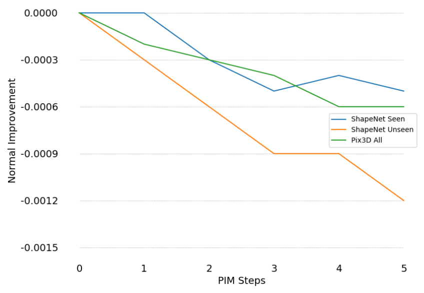
(a)



(b)

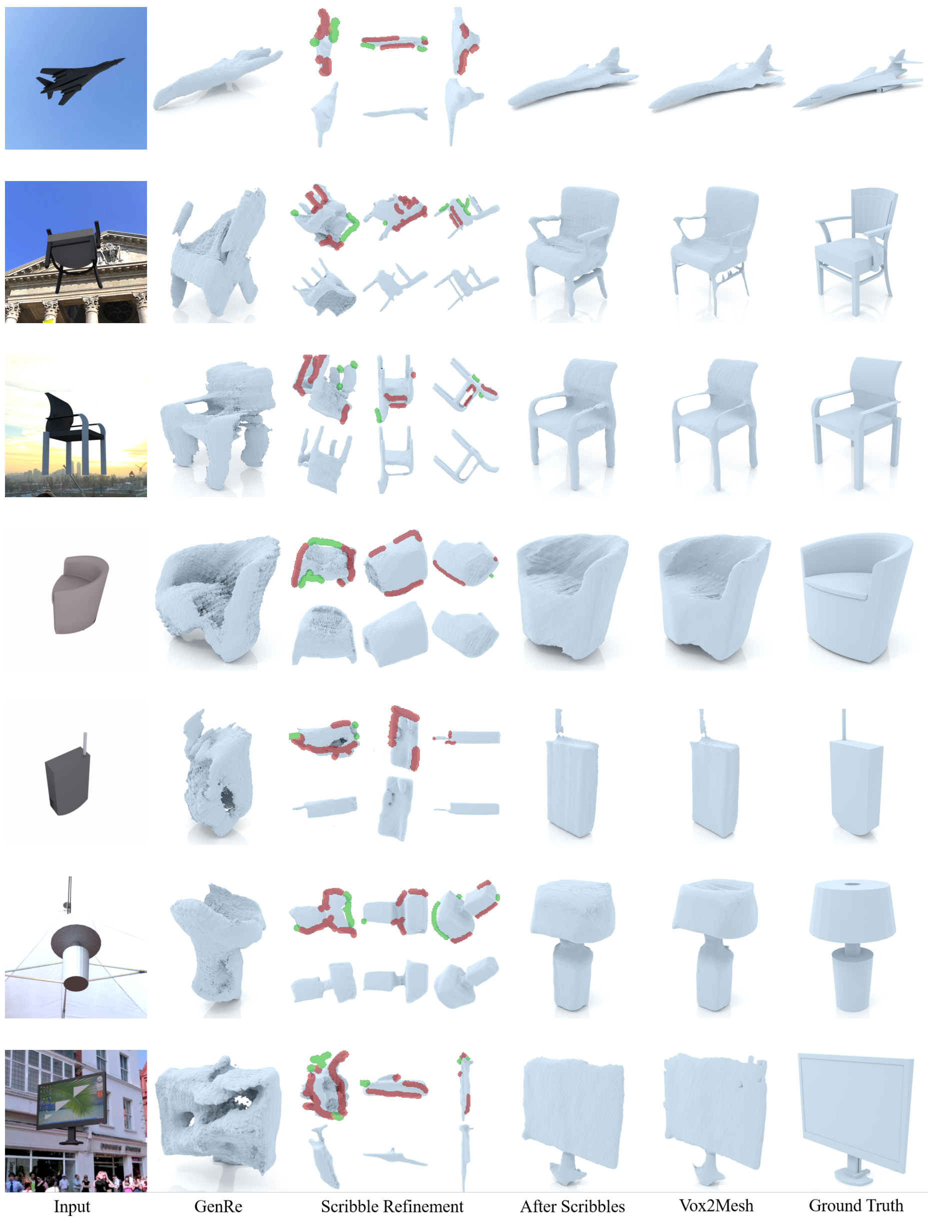


(c)



(d)

Fig. 2: Number of Corrections vs. Improvements on F1 and Normal Consistency Score


Input
GenRe
Scribble Refinement
After Scribbles
Vox2Mesh
Ground Truth

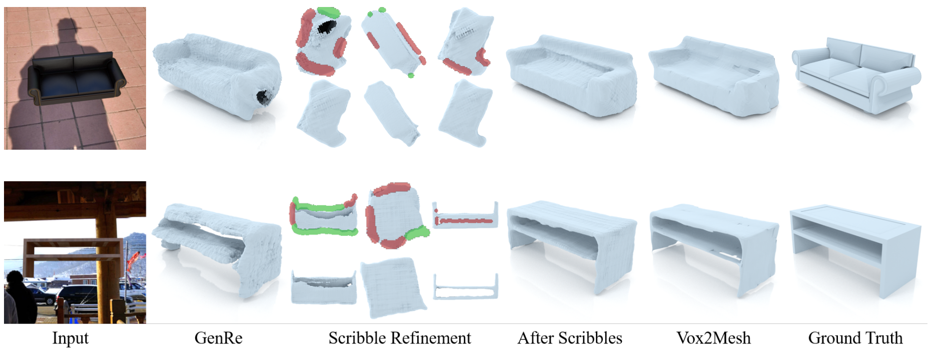


Fig. 3: Example shapes (ShapeNet) annotated using our framework. First two rows show seen classes while the other rows show unseen classes. Output shape from each intermediate step is shown from left to right, followed by ground truth mesh. The additive/deletion scribbles are shown in green/red. In the scribble refinement steps, although the additive scribbles are drawn in 2D, SIM can accurately infer the depth of the added shape (*e.g.*, bipods in the third example). With a few scribbles, our model predicts shape close to ground truth. Vox2Mesh further converts the discrete voxel grid to a fine mesh.

PIM Examples: In the main paper, we showed that PIM is capable of creating concave surface with users dragging one vertex on the surface. Here, we show another usage of PIM which is to refine local details. As shown in Figure 4, the user can correct a local surface using PIM without re-predicting the whole shape. This is favourable over SIM when the overall shape is good.

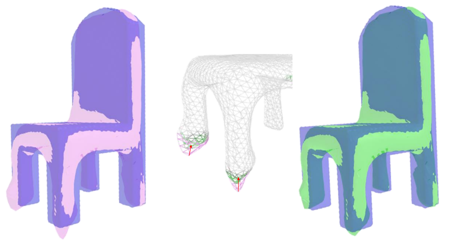


Fig. 4: An example of PIM correcting local the surface. Ground truth shape, prediction before PIM, prediction after PIM are represented in blue purple and green, respectively. The red arrow represents human correction.

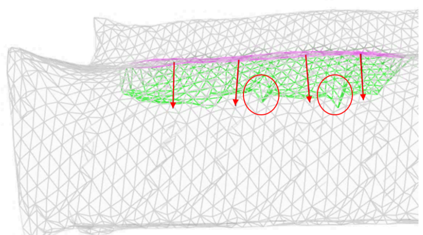


Fig. 5: A failure case of PIM. The vertices in red circles are affected by two corrections, denoted as red arrows. The deviation accumulates and leads to a gap between these points and surrounding vertices, which are only affected by single correction.

User Study Examples: In user study, we created a real dataset from Pix3D images using our annotation tool. Here we show examples of annotated shapes in Fig. 6. We include screen capture of real user annotating ShapeNet and Pix3D images using our web tool in *real_tool.mp4*.

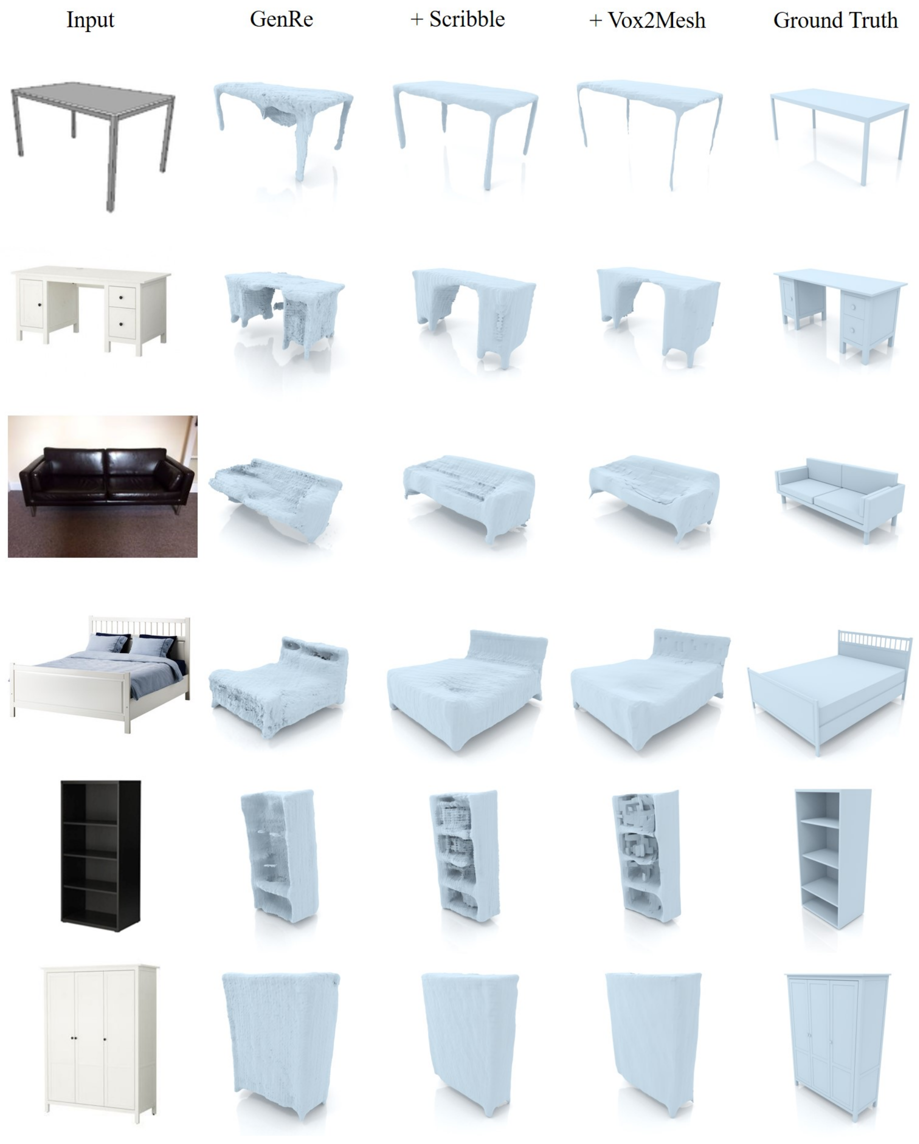


Fig. 6: Example shapes (Pix3D) annotated by real human using our web tool. Each example is from a different object category which is unseen to our base model.

Failure cases: With SIM, we found that it is difficult to annotate complex shapes from bad initial predictions, or to annotate small but fine details, as shown in the first two examples in Figure 7. Vox2Mesh sometimes creates artifacts along thin edges (*e.g.*, chair leg, airfoil), as shown in the third example of Figure 7.

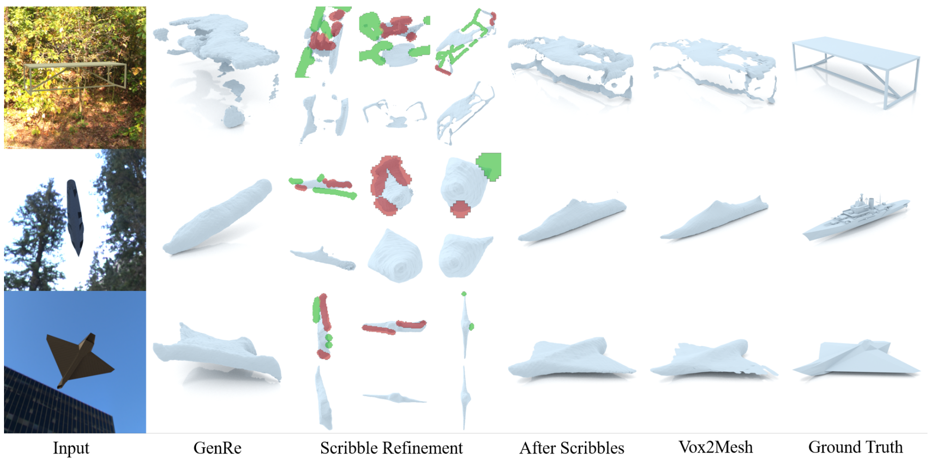


Fig. 7: Failure cases of SIM and Vox2Mesh

For PIM, we observed that when correcting a large surface with multiple drag-and-drops, the artifacts would appear in the overlapping region. As shown in Figure 5, the overlapping region in-between two corrected vertices are pushed lower than vertices affected by only 1 correction.

2.3 Number of Corrections vs. Improvements on F1 and Normal Consistency Score

In the main paper, we reported improvements in terms of Chamfer Distance. Here we further show results in terms of the F1 score (see Figure 2 (a) and (c)) and the Normal Consistency Score (see Figure 2 (b) and (d)). Notice that the normal consistency score drops slightly for PIM because we used a smaller regularization loss. Also, the problem discussed above lowers this score.

2.4 User Study

Full Scene Annotation: Our model first densely samples points from the observed surface within an object bounding box, normalizes this point cloud into a unit cube, and constructs an occupancy grid as input to GenRe. During annotation, the input point cloud is rendered transparently on the predicted shape as a visual clue. 15 scribbles on avg. were drawn to annotate each object.

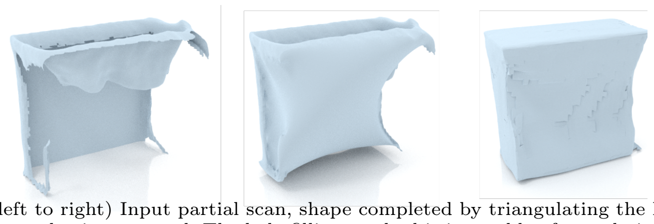


Fig. 8: (From left to right) Input partial scan, shape completed by triangulating the hole boundary and shape annotated using our tool. The hole-filling method is incapable of completing large unseen surface (the back of the bin). We use the CGAL [7] implementation for hole filling with triangulation.

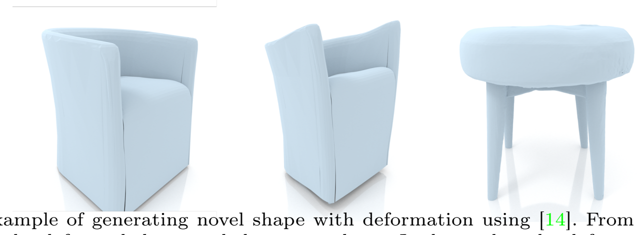


Fig. 9: An example of generating novel shape with deformation using [14]. From left to right: the source shape, the deformed shape and the target shape. It shows that the deformation result is not faithful to the target shape when no similar shape template exists in database.

2.5 Alternative Methods for Shape Annotation

Here we discuss two alternative methods and their limitations as annotation tool. Recent work [4,13,14] explored deformation of 3D shape for fast content creation. This approach can preserve the high-quality geometric details in the source shape while matching the general structure of the reference shape or image, making it suitable for various graphic applications, *i.e.* animation. However, they output shapes that are not faithful to the target when no similar template exists, compared to SOTA 3D reconstruction methods. We illustrate this limitation in Fig 9 using the same example as in Fig. 10 from the main paper. Note that we use the closest shape (in Chamfer Distance) in ShapeNet to generate the target shape in Pix3D.

In practice, the partial scan of the scene can also be completed using hole-filling algorithms as in [11]. However, such method can only fix small holes or gaps, and can not complete large unseen parts as shown in Fig 8.

In summary, we believe our comparison with current annotation approach and user study proves the practical value and necessity for learning-based annotation.

References

1. Chen, W., Gao, J., Ling, H., Smith, E., Lehtinen, J., Jacobson, A., Fidler, S.: Learning to predict 3d objects with an interpolation-based differentiable renderer. In: Advances In Neural Information Processing Systems (2019) 5
2. Fujiwara, K.: Eigenvalues of laplacians on a closed riemannian manifold and its nets. Proceedings of the American Mathematical Society **123**(8), 2585–2594 (1995) 4

3. Gkioxari, G., Malik, J., Johnson, J.: Mesh r-cnn. arXiv preprint arXiv:1906.02739 (2019) 3, 5
4. Hanocka, R., Fish, N., Wang, Z., Giryes, R., Fleishman, S., Cohen-Or, D.: Alignet: Partial-shape agnostic alignment via unsupervised learning. ACM Trans. Graph. (2018) 11
5. Ling, H., Gao, J., Kar, A., Chen, W., Fidler, S.: Fast interactive object annotation with curve-gcn. In: CVPR. pp. 5257–5266 (2019) 3, 4
6. Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3d surface construction algorithm. In: ACM siggraph computer graphics. vol. 21, pp. 163–169. ACM (1987) 5
7. Loriot, S., Rouxel-Labbé, M., Tournois, J., Yaz, I.O.: Polygon mesh processing. In: CGAL User and Reference Manual. CGAL Editorial Board, 5.0.2 edn. (2020), https://doc.cgal.org/5.0.2/Manual/packages.html#PkgPolygonMeshProcessing 11
8. Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., Geiger, A.: Occupancy networks: Learning 3d reconstruction in function space. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4460–4470 (2019) 5
9. Oh, S.W., Lee, J., Xu, N., Kim, S.J.: Fast user-guided video object segmentation by interaction-and-propagation networks. CoRR **abs/1904.09791** (2019), http://arxiv.org/abs/1904.09791 2
10. Smith, E., Fujimoto, S., Romero, A., Meger, D.: GEOMetrics: Exploiting geometric structure for graph-encoded objects. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 97, pp. 5866–5876. PMLR, Long Beach, California, USA (09–15 Jun 2019), http://proceedings.mlr.press/v97/smith19a.html 3
11. Straub, J., Whelan, T., Ma, L., Chen, Y., Wijmans, E., Green, S., Engel, J.J., Mur-Artal, R., Ren, C., Verma, S., Clarkson, A., Yan, M., Budge, B., Yan, Y., Pan, X., Yon, J., Zou, Y., Leon, K., Carter, N., Briales, J., Gillingham, T., Mueggler, E., Pesqueira, L., Savva, M., Batra, D., Strasdat, H.M., Nardi, R.D., Goesele, M., Lovegrove, S., Newcombe, R.A.: The replica dataset: A digital replica of indoor spaces. CoRR **abs/1906.05797** (2019), http://arxiv.org/abs/1906.05797 11
12. Wang, N., Zhang, Y., Li, Z., Fu, Y., Liu, W., Jiang, Y.G.: Pixel2mesh: Generating 3d mesh models from single rgb images. In: ECCV (2018) 3, 4, 5
13. Wang, W., Ceylan, D., Mech, R., Neumann, U.: 3dn: 3d deformation network. In: CVPR (2019) 11
14. Yifan, W., Aigerman, N., Kim, V.G., Chaudhuri, S., Sorkine-Hornung, O.: Neural cages for detail-preserving 3d deformations. In: CVPR (2020) 11
15. Zhang, X., Zhang, Z., Zhang, C., Tenenbaum, J.B., Freeman, W.T., Wu, J.: Learning to Reconstruct Shapes from Unseen Classes. In: NeurIPS (2018) 5