

A Appendix

A.1 More experiments

Further demonstration of the advantages of real-to-sim way. Our R2SRepairer can mitigate camera noise which is beneficial for the grasp detector. To further validate our real-to-sim perspective, we also conduct experiment that adding R2SRepairer to the grasp detector [1] trained on real-world data. As shown in Table S1, after adding R2SRepairer (see Line 2), the grasp performance shows a modest improvement, but it still lags significantly behind our method that is trained on simulated data. This demonstrates that a large amount of simulated data can train a more robust grasp detector, and by utilizing our real-to-sim method, this capability can be effectively applied to real-world data.

Method	All	Seen	Similar	Novel
GSNet [1] w/o R2SRepairer	42.53	61.19	47.39	19.01
GSNet [1] w R2SRepairer	43.05	61.39	48.05	19.72
Ours	48.44	66.12	54.47	24.74

Table S1: Average precision comparison on real-world data captured by Kinect.



Figure S1: Comparison of camera noise before and after R2SRepairer. Different colors in the noise map represent different noise amplitude ranges, with amplitude measured in millimeters

Qualitative analysis of the Real-to-Sim Data Repairer. We further demonstrate the effectiveness of our Real-to-Sim Data Repairer (R2SRepairer) by presenting camera noise before and after refinement. As shown in Figure S1, it is evident that the correct noise image significantly reduce noise compared to the initial noise image. To further demonstrate the performance of R2SRepairer, we compared the single-view point cloud before and after camera noise repair. As shown in Figure S2, after noise repair, the positional drift and structural deformation of the point cloud are mitigated, bridging the gap between real and simulated data.

Qualitative analysis of structural features in memory bank. Our Real-to-Sim Feature Enhancer (R2SEnhancer) uses the precise structural features stored in the memory bank to enhance the real-world features. To visually demonstrate the semantic information of the stored structural features, we visualize the structures represented by these features. First, we calculate the cosine distance between the features of each graspable point and the stored features in the memory bank, assigning

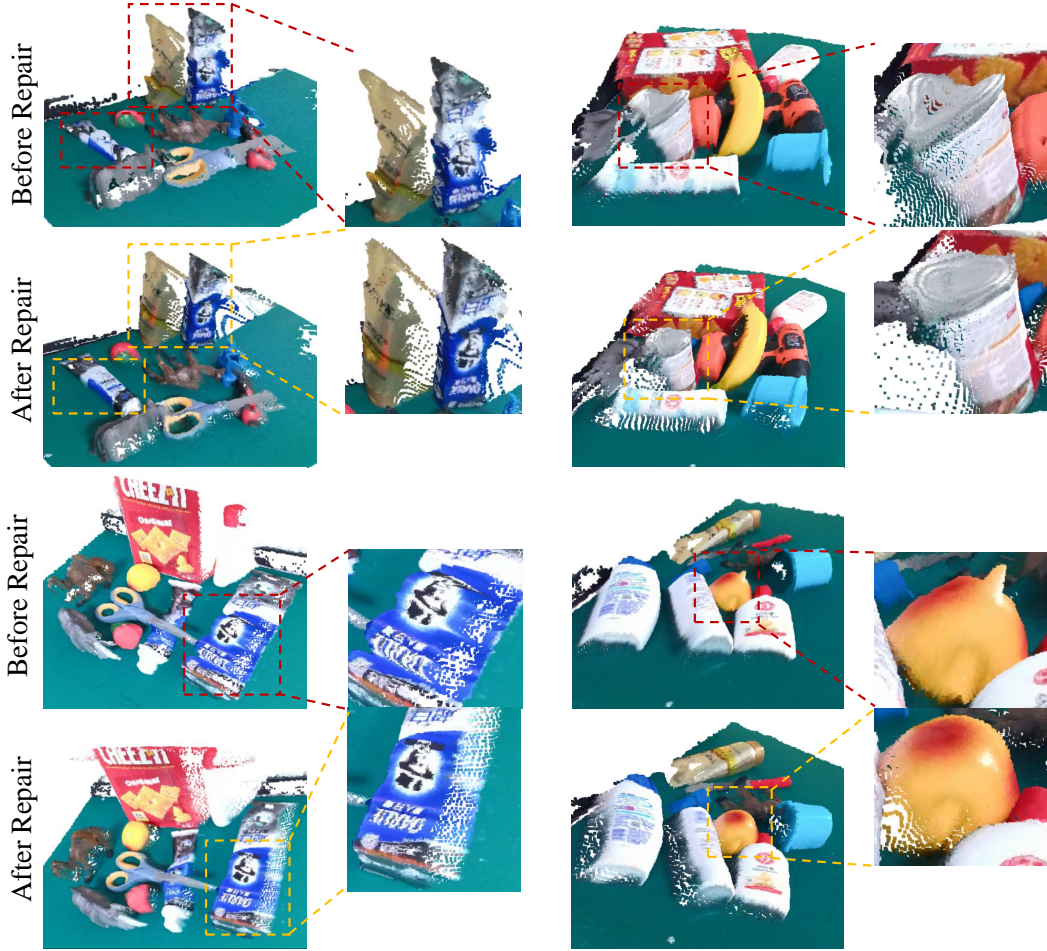


Figure S2: Visualization of single-view point cloud before and after repair. Zoom in better view.

each graspable point to the nearest stored structural feature. Then, we obtain a set of graspable points for each stored structural feature. Based on cosine similarity, we select three distinct features from the memory bank and visualize their corresponding sets of graspable points. As depicted in Figure S3, there are obvious differences in the structure represented by the three features. The first feature represents sharp object structures, as shown in the first row of Figure S3, which are commonly found on toy legs and heads. The second feature represents planar object structures, as depicted in the second row, primarily seen on square boxes. The third feature represents curved object structures, as illustrated in the last row, appearing on various curved surfaces, with bottles being the most prominent.

Qualitative analysis of grasping performance. To demonstrate that our R2SGrasp can adapt to real data, we use R2SGrasp to predict grasp poses on the GraspNet-1Billion test set and visualize the results, as shown in Figure S4. In seen and similar scenes, R2SGrasp predicts grasping poses with a success rate close to 100%. In novel scenes, there are some failed cases, which occur due to collisions or grasping at empty locations.

A.2 Implementation details of grasp detector.

Architecture design of grasp detector. The grasp detector in details is shown in Figure S5. We first randomly sample N points from the single-view point clouds generated from depth map, and then use a point cloud backbone to extract point-wise features with C_1 dimension. The point cloud



Figure S3: Semantic information of the stored structural features. “Blue”, “Cyan”, “Yellow” represent the structures corresponding to the three selected structural features.

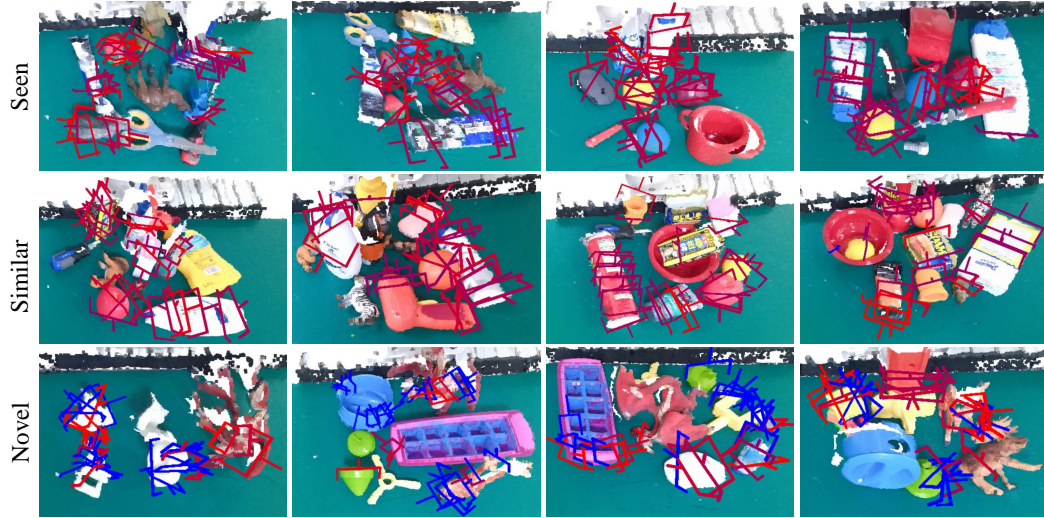


Figure S4: Top-30 grasp poses predicted by R2SGrasp on the test set of GraspNet-1Billion. The red gripper indicates successful grasp pose, while the blue gripper indicates failed grasp pose.

backbone adopt a Unet [2] architecture with a ResNet14 [3] encoder built upon the Minkowski
 Engine [4]. Followed by a MLP layer, we predict the object point mask I_o and graspness heatmap
 I_h to select the graspable points along with their corresponding point-wise features with the shape of
 $M \times (C_1 + 3)$, where M is the number of graspable points and 3 denotes the cartesian coordinates of
 the points. We also select the grasp view of the graspable points from the predefined 300 approaching
 vectors based on the grasp view scores s_v which is also predicted by a MLP layer. Then, we perform
 cylinder grouping operation along the grasp view for each grasp point to aggregate the features of
 G neighboring points, followed by the MLP and max pooling operations to extract local structural
 features with C_2 dimension. Finally, our Real-to-Sim Feature Enhancer (R2SEnhancer) refines the

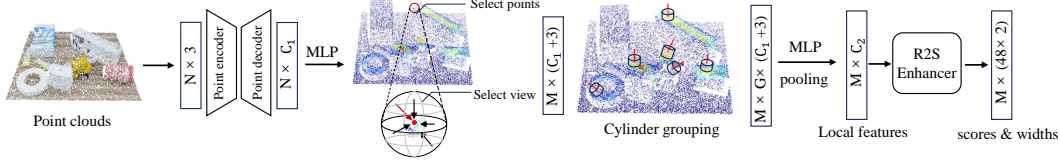


Figure S5: Grasp detector in details

local structural features using the stored simulated features and outputs the grasp scores s_g and widths s_w shaped as $M \times 48$, where 48 denotes the grasp candidates of the grasp points. For our network, we set $N = 20000$, $M = 1024$, $C_1 = 512$, $C_2 = 256$, $G = 16$.

Loss Design. The grasp detector is trained with the following loss function:

$$L_g = L_o(I_o, I_o^*) + \lambda_1 L_h(I_h, I_h^*) + \lambda_2 L_v(s_v, s_v^*) + \lambda_3 L_s(s_g, s_g^*) + \lambda_4 L_w(s_w, s_w^*), \quad (1)$$

where L_o, L_h, L_v, L_s, L_w are used to supervise the learning of object points, graspable points, grasp views, grasp scores and grasp widths respectively. $I_o^*, I_h^*, s_v^*, s_g^*, s_w^*$ is the ground truth of object point mask, graspness heatmap, grasp view scores, grasp scores and grasp widths. L_o adopts binary classification loss, while others use regression loss. Due to the simplification of our grasp annotations, some grasp poses may lack supervision signals. Therefore, when calculating the loss, we ignore any predicted grasp poses that lack supervision signals.

A.3 R2Sim dataset details

The overall process of dataset construction is shown in Figure S6. We start by selecting 256 daily household objects from the Google Scanned Objects dataset [5] and the GraspNet-1Billion training dataset [6]. Then, we generate scenes in blenderproc [7] and simultaneously label the grasp poses of the objects. After that, we project object-level grasp annotations into the scenes and detect the grasp annotations that result in collisions. Finally, we adopt our proposed grasp annotation simplification method to remove ineffective grasp poses. The remaining grasps serve as scene-level annotations. To sum up, our R2Sim dataset comprises 500 scenes with 76,800 RGB-D images. Each scene contains approximately 14.4 million grasp annotations and each frame in every scene is also annotated with object segmentation maps, 6-DoF poses of objects and camera, graspness heatmap and view graspness. In the graspness heatmap, brighter areas indicate a higher likelihood of successful grasps. Similarly, higher values of view graspness also represent a greater probability of successful grasps. Some examples of RGB, depth map, segmentation map and graspness map are shown in Figure S7.

A.3.1 Details of object level grasp annotation.

We use a sampling-evaluation approach to annotate the grasp poses of objects. Grasp poses are determined by downsampling high-quality mesh models to ensure that the grasp points are evenly distributed in the voxel space. For each grasp points, 300 approach directions are sampled uniformly on a spherical space. Grasp candidates of each approach directions are explored on a grid defined by 4 gripper depths and 12 rotation angles. To sum up, there are 48 grasp candidates along each approach direction and 14,400 grasp candidates on each grasp point. The gripper width is adjusted as necessary to prevent empty grasps or collisions. We adopt analytic computation method as [6, 8] to grade the sampled grasp poses. The grasp scores range from 0 to 1, with higher scores indicating a greater likelihood of successful grasps.

A.3.2 Details of scene level grasp annotation.

Using the object poses, the grasp pose in object coordinate system is projected onto the world coordinate system. We detect collisions for the projected grasp poses in the scene and set the scores of those that collide to zero. Assuming there are N grasp points projected into the scene, we calculate the success rate of grasp candidates at each point, resulting in N graspness values. Using the camera's intrinsic parameters, we convert the depth map into a single-view point cloud. We then use

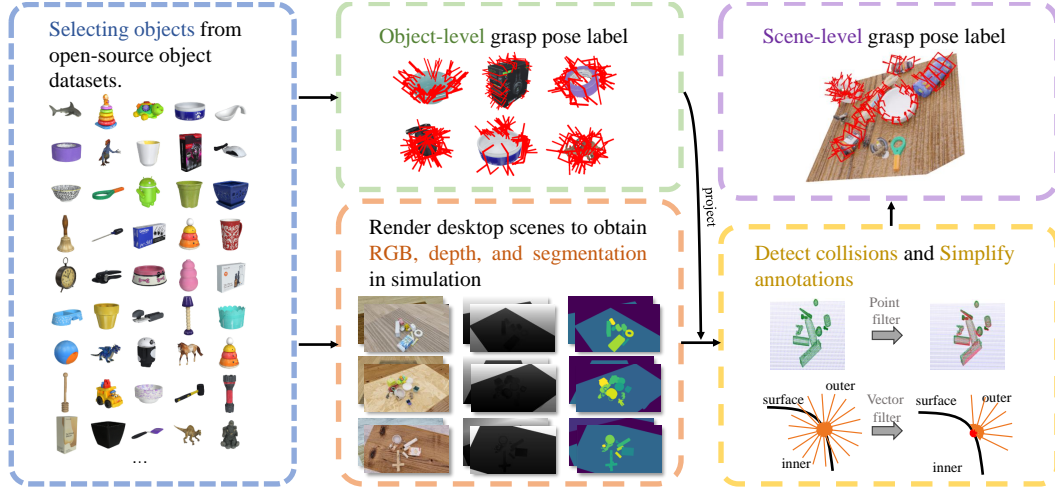


Figure S6: Overview of data generation pipeline.

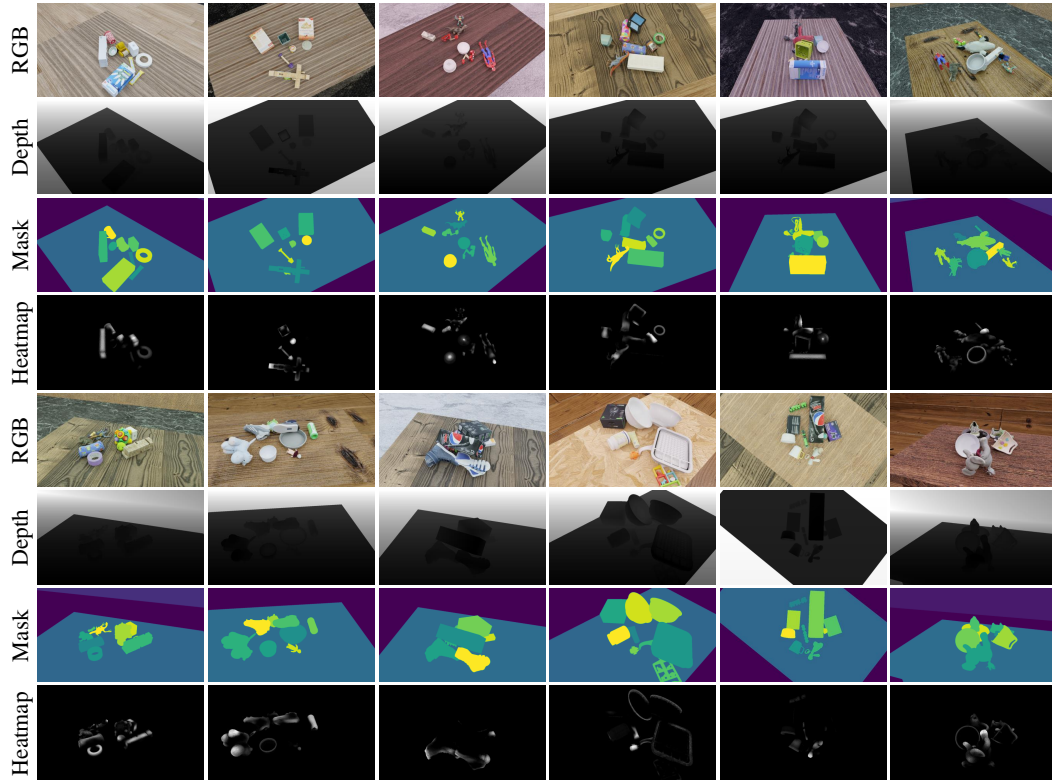


Figure S7: Display of RGB, depth map, segmentation mask and graspness heatmap in our simulated dataset.

93 the K-NN algorithm to match the grasp points in the scene, assigning each point in the point cloud
 94 the corresponding graspness value. This value is then back-projected into the image to create the
 95 graspness heatmap. Similarly, we calculate the success rate of grasp candidates along each approach
 96 direction and using this as the view-graspness. We also make further simplification of grasp pose
 97 annotation to improve training efficiency.

A.3.3 Scene generation.

We automated the construction of cluttered desktop scenes using Blenderproc [7]. We first construct a simple indoor scene with a table placed at the center. Textures for the table, floor, and walls are chosen randomly from specific material categories provided on AmbientCG. The lighting setup of the scene is randomized as well, with practical adjustments to intensity and variations in light color to improve visual clarity and accuracy. Then, we choose a variable number of objects ranging from 7 to 10 from our object pool and place on the table. To create sufficiently cluttered scenes, we place the objects 1.5 meters above the table and allow them to fall naturally onto the surface. Finally, we set 128 camera poses to capture RGB-D images from multi views, where the poses are randomly sampled on the upper hemisphere, with a radius of 1.1 meters centered on the objects’ region. Based on the above setup, we obtain RGB-D images, object segmentation maps, object poses, and camera poses from different angles efficiently.

A.3.4 Analysis of simplified annotation.

To demonstrate the advantage of simplified annotation, we compare the simplified and non-simplified annotations across multiple metrics. The average precision measures the performance of grasp detector trained on 300 scenes selected from R2Sim dataset. Except for the average accuracy, all other metrics are evaluated on the entire R2Sim

Metrics	Simplified	Non-Simplified
Average Precision(%)	35.94	33.68
Run Time(epoch/h)	5.64	21.17
Memory Usage(GB)	7.50	48.41
GPU Memory Usage(GB)	4.43	9.77
Storage Usage(GB)	15	72.44

Table S2: Compare metrics between simplified and non-simplified annotations.

dataset. As shown in Table S2, following the simplification of annotations, there are a notable increase of 2.26 AP. We believe that this improvement is due to the simplified annotations alleviating the imbalance between positive and negative samples present in the original annotations, where positive samples made up less than 2% of the total according to statistics. Moreover, after simplifying the annotations, the program’s runtime, memory usage, and GPU usage decrease by 73.36%, 84.51%, and 54.66%, respectively, and the storage usage for the entire dataset annotation decrease by 79.29%. This simplified annotation method is crucial for constructing a large-scale dataset, as it reduces the burden of data storage and neural network training.

References

- [1] C. Wang, H.-S. Fang, M. Gou, H. Fang, J. Gao, and C. Lu. Graspness discovery in clutters for fast and accurate grasp detection. In *IEEE/CVF International Conference on Computer Vision*, 2021.
- [2] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention*, 2015.
- [3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE conference on computer vision and pattern recognition*, 2016.
- [4] C. Choy, J. Gwak, and S. Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.
- [5] L. Downs, A. Francis, N. Koenig, B. Kinman, R. Hickman, K. Reymann, T. B. McHugh, and V. Vanhoucke. Google scanned objects: A high-quality dataset of 3d scanned household items. In *IEEE International Conference on Robotics and Automation*, 2022.
- [6] H.-S. Fang, C. Wang, M. Gou, and C. Lu. Graspnet-1billion: A large-scale benchmark for general object grasping. In *IEEE/CVF conference on computer vision and pattern recognition*, 2020.
- [7] M. Denninger, D. Winkelbauer, M. Sundermeyer, W. Boerdijk, M. Knauer, K. H. Strobl, M. Humt, and R. Triebel. Blenderproc2: A procedural pipeline for photorealistic rendering. *Journal of Open Source Software*, 2023.
- [8] V.-D. Nguyen. Constructing force-closure grasps. *The International Journal of Robotics Research*, 1988.