Cubify Anything: Scaling Indoor 3D Object Detection

Supplementary Material

7. CA-1M

7.1. Additional Dataset Samples

We include additional panoramic projections of sample annotations in Figure 11.

7.2. Rendering CA-1M

We provide more details on the process to generate CA-1M per-frame ground truth. Given a video capture consisting of N frames, per-frame intrinsics K_i, possible distortion parameters d_i, and per-frame pose RT_i registered to the same coordinate system as the laser scans, we define a high-level rendering pipeline:

1. Transform 3D boxes from world to camera space using RT_i.
2. Apply frustum culling, removing boxes irrelevant to the current camera pose and frustum
3. Render each box independently at a suitable resolution (e.g., 320x240) using K_i and d_i.
4. Cut each 3D box to the **frustum** by backprojecting the independently rendered masks along rays terminating at the far position of the frustum.
5. Determine coarse visibility of each box by considering the the subset of each rendered mask whose depth is closest to the camera.
6. Cut each 3D box to the **scene** by backprojecting the visibility masks along rays terminating at the rendered *depth* to solve for visibility and occlusion.
7. Remove 3D boxes which have had too significant of a cut relative to the original box.

This process was shown in the main paper as Figure 4. This pipeline is implemented in PyTorch3D [13] and parallelized across multiple GPUs and nodes such that ground-truth can feasibly be generated for every frame over thousands of captures.

7.2.1. Importance of Handling Distortion

During rendering (of boxes), we must handle distortion (even in the main camera) within the handheld captures in order to maintain pixel alignment, as shown in Figure 8. While this is easier to implement when dealing with points or high resolution meshes, this is non-trivial to implement for 3D boxes.

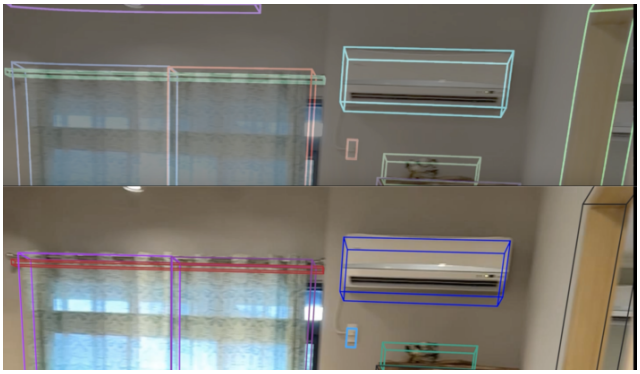

Figure 8. Rendering 3D boxes with distortion is important on CA-1M in order to maintain pixel accuracy and alignment for small, distant objects. The top image is rendered with distortion whereas the bottom is rendered purely using a Pinhole model.

8. Cubify Transformer

8.1. Additional Training Details

All image-based methods (CuTR, Cube R-CNN) are trained in a framework based on Detectron2. CuTR uses settings similar to ViTDet (AdamW optimizer, layer decay rates) while we use the same settings from Omni3D [3] for traning Cube R-CNN models. All point-based methods are trained in MMDetection3D using the default settings from each respective work.

Image-based models are trained across 8×4 V100 nodes using a total batch size of 64 (2 images per GPU). All data is processed into WebDataset format and streamed to each node — keyframing (with some randomness) is applied on the fly. Images are randomly augmented using the ViTDet style augmentations: a random factor between 0.25 and 1.25 is selected and multiplied by a base resolution of 1024×1024 to get the target image size. A random crop is taken (or the full image if the resolution is smaller). The depth map is augmented accordingly to preserve the same ratio of image to depth. We also select to randomly augment the image to depth ratio (1, 2 or 4).

Since we generate our training data at high frequency, we apply keyframing with a displacement of 30cm or a change in angle of 15 degrees.

8.2. Inference speed

In terms of model size, we trained ViT-B (90M), ViT-S (25M), and ViT-T (8M) models. We are releasing MPS-accelerated (Apple Silicon GPU) ViT-B checkpoints which provide comparable inference times to CUDA: 150 ms on a V100 versus 270 ms on an M1 Mac Studio. We emphasize that a 3D sparse convolution is not generally available on practical accelerators which makes them unusable for deployment on most mobile devices.

9. ScanNet++

9.1. ScanNet++ versus CA-1M

ScanNet++ adopts the same FARO scanner and handheld capture approach as ARKitScenes/CA-1M and thus can, in theory, offer similar accuracy in 3D annotations to CA-1M. However, we note significant differences compared to CA-1M. First, ScanNet++ is *not a detection dataset*, only offering semantic instance segmentation of a large but not exhaustive taxonomy on the FARO meshes. While we can (and do) attempt to derive 6-DOF axis-aligned boxes from these, annotating with explicit 3D boxes ensures that *every object* can labeled, as done in CA-1M. This is because even a high quality scanner like FARO will still be unable to acquire points for many objects in a scene (field-of-view, material, etc). Points will inevitably be missing and so by only labeling acquired points as ScanNet++ does, a significant number of objects will still go unlabeled (see Figure 5). Second, ScanNet++ is an order of magnitude smaller than CA-1M in terms of scans and captures — having 230 captures versus 3,500 in CA-1M. Finally, the registration process in ScanNet++ is less complete — *only able to directly register a subset* of frames (264K out of 3.2M frames, i.e., 8%) to the FARO space. As mentioned in ScanNet++ [20], frames are considered not registerable when the rendering of depth from the FARO scans does not sufficiently agree with depth acquired from the handheld device (i.e., the LiDAR-equipped iPhone). Not only does this lead to less available frames, it *biases* the dataset towards frames where the handheld's acquired depth is "accurate" — partially entangling the dataset to the sensors of the handheld device. On the other hand, the registration process of ARKitScenes/CA-1M registers nearly every frame across all captures.

9.2. Experiments on ScanNet++

While we don't attempt to draw any conclusions dissimilar from the experiments on CA-1M, we, for completeness, include results of models trained on ScanNet++ data. Since ScanNet++ does not offer 3D box annotations, we must derive them. We use the 3D instance segmentation labels and use the normal of the first "wall" object in each scan to define the basic rotation of the scene and thus produce 6-DoF (axis-aligned) 3D boxes. Notably, this can be error prone if this wall happens to not adhere to some Manhattan assumption, which happens rarely. To generate per-frame ground-truth, we augment the same protocol defined in Section 7.2, but instead of using the rendered 3D boxes to generate a coarse instance mask (like in Figure 4), we render the modal instance segmentation from the underlying mesh annotations. The rest of the process (filtering, cutting, etc) proceeds the same as CA-1M. While ScanNet++ has a significant number of class labels, we do not consider them in experiments and treat this as a class-agnostic dataset to avoid complications that may arise due to the vocabulary size. Experimental results are presented in Table 6.

Method	AP25	AR25	AP50	AR50
3D point-based methods				
ImVoxelNet (RGB only)	21.3	40.7	4.4	11.7
FCAF	41.1	65.3	16.4	33.2
TR3D	38.1	68.3	12.8	30.9
TR3D + FF	40.3	66.6	14.1	31.4
2D based methods				
Cube R-CNN (RGB only)	15.5	33.2	3.5	10.6
CuTR (RGB only)	25.8	49.0	7.0	20.2
CuTR (RGB-D)	**48.7**	**71.5**	**18.5**	**36.6**

Table 6. Results on the ScanNet++ dataset when using ARKit depth and axis-aligned (i.e., to walls) ground-truth boxes derived from provided instance segmentation. We evaluate the class-agnostic precision and recall of each method.

Generally, we observe similar trends as we see on CA-1M: CuTR (both RGB and RGB-D variants) outperforms existing methods over the range of evaluation thresholds.

10. ARKitScenes

We present no specific results for ARKitScenes, however, we do use a CuTR model pre-trained on per-frame ARKitScenes data in order to conduct the experiments in 5. For uniformity, this per-frame ground-truth is generated using the same protocol as CA-1M, with only the underlying pose and 3D box annotations coming from the original ARKitScenes [2] dataset.

11. Further Ablations

We provide additional ablations which attempt to answer not only specific questions about CuTR and CA-1M but also to further ablate the shortcomings of point-based methods.

Model Size While all experiments of CuTR are done using a ViT-B backbone, we additionally ablate how CuTR responds to smaller backbones: a ViT-S (6 heads, 384 dimension embedding) and a ViT-T (3 heads, 192 dimension embedding). We observe that CuTR scales to these smaller backbones gracefully, still retaining performance competitive with point-based methods even at the ViT-T size.

Depth (Modality) Fusion CuTR adopts a MultiMAE-like backbone: a separate depth patch embedding and joint encoding of both RGB and depth tokens within a Transformer encoder. This can be initialized by a pre-trained MultiMAE (i.e., on pseudo-labeled ImageNet) or we could use a strong

ViT	AP25	AR25	AP50	AR50
Tiny	28.8	54.3	7.3	22.0
Small	34.8	58.3	9.6	25.4
Base	40.9	62.3	12.7	29.1

Table 7. CuTR can still retain competitive performance on CA-1M while using significantly smaller model sizes like ViT-S and ViT-T.

RGB only backbone like DINOv2 or Depth-Anything and learn the depth fusion (from scratch) during training. In Table 8, we show the benefit of initialization from MultiMAE pre-training over DINOv2 [10] or Depth-Anything [19], despite these being considered much stronger backbones than MultiMAE. We compare both the 2D and 3D box accuracy which helps explain whether performance differences come from 2D box quality or from 3D.

Backbone	AR50 (2D)	AR75 (2D)	AR25 (3D)	AR50 (3D)
MultiMAE [1]	83.6	55.1	**62.4**	**29.0**
DINOv2 [10]	**83.9**	**55.2**	60.5	27.2
Depth-Anything [19]	83.8	55.0	59.8	26.7

Table 8. CuTR initialized with a pre-trained MultiMAE backbone shows strong performance on CA-1M against pre-trained DINOv2 and Depth-Anything backbones which must learn the depth modality fusion during training. While all share similar 2D box performance, MultiMAE shows significant improvements with respect to 3D accuracy. This may show the importance of depth fusion at *pre-training* time for 3D box prediction.

"Pixel for point" All point-based methods are inherently limited by the depth provided by the underlying sensor, i.e., they can only backproject at a resolution limited by the depth (256×192 in CA-1M). In practice, depth is usually significantly lower resolution than what the RGB sensor is capable of delivering (1024×768 in CA-1M). Therefore, an advantage of CuTR could be seen as efficiently fusing lower resolution depth data with higher resolution RGB data. In Table 9, we purposefully decrease the RGB resolution given to CuTR to understand how "pixel for point" it compares to point-based methods. Even at the exact "pixel for point" setting of RGB and depth at 256×192, CuTR can match the performance of point-based methods. Restoring this resolution even to 512×384 shows minimal losses over the full 1024×768 evaluation setting of CuTR.

We further understand how the resolution affects performance over distance by decomposing the 3D evaluation of Table 9 by **box distance**. We group all boxes into *near* (0-2 meters), *medium* (2-4 meters), and *far* (4-5 meters) buckets.

We observe that the most significant drops in performance as RGB resolution decreases, comes from the further objects (medium and far distances), which may stem from the possibility that both depth uncertainty increases and size in image decreases as objects are further away. Higher RGB resolutions would better allow the network to learn to overcome these shortcomings.

RGB Res.	AR50 (2D)	AR75 (2D)	AR25 (3D)	AR50 (3D)
1024×768	83.6	55.2	62.3	29.1
512×384	81.7	52.2	59.3	26.0
256×192	81.1	51.9	50.7	19.5
3D point-based methods				
FCAF	N/A	N/A	49.5	22.6
TR3D	N/A	N/A	51.9	20.0
TR3D + FF	N/A	N/A	52.9	21.0

Table 9. CuTR is evaluated on CA-1M at a variety of RGB resolutions with a fixed depth resolution of 256×192. We reproduce the point-based method results here for convenience. Both 2D and 3D recall suffers at lower RGB resolutions, however, CuTR remains competitive with point-based methods.

RGB Res.	AR25n	AR25m	AR25f
1024×768	68.4	58.8	40.0
512×384	66.4	54.5	35.4
256×192	59.1	44.5	24.8

Table 10. We view the results of Table 9 with the 3D evaluation broken down by **n**ear, **m**edium, **f**ar box distances.