Sliding Shapes for 3D Object Detection in Depth Images

Shuran Song Jianxiong Xiao

Princeton University
Supplemental material

In this supplementary material, we first explain some details of the algorithm in Section 1, and then talk about the data used for training and testing in Section 2. We present more results in Section 3.

1 Algorithm Details

Identifying out of sight, missing depth cell, and occlusion source depth As is discussed in the paper, it is important to identify cells that are out of sight, occluded by other objects or having vast missing depth, and then set their features to zeros so that they do not bias the overall detection score. The way we identify these cells is the following: similar to TSDF feature computation, after each cell is divided into a $6 \times 6 \times 6$ voxel grid, we project cell center's 3D location (x_c, y_c, z_c) onto 2D image plane. If one cell has more than one voxel falls outside the image boundary, it is flagged as out of sight.

Otherwise, we round the voxel center positions into pixel coordinate and read the corresponding depth value D from the depth map. If D is missing, the voxel will be treated as missing as well. If more than 90% voxels of a cell are missing, the whole cell is treated as missing, and its features will be set to zeros.

For a threshold value μ, if $y_c < D - \mu$, the voxel is in front of a surface (this is possible because y_c is the average depth). If $D - \mu < y_c < D + \mu$, the voxel is on a surface. If $y_c > D + \mu$, the voxel is behind a surface thus occluded. If more than 90% voxels of a cell are occluded, the whole cell is treated as occluded, and the median of D over all voxels in this cell is stored as the depth of the occlusion source. At testing time, if the depth of occlusion source is smaller than the sliding windows minimum depth, then the occluder must be in front of the sliding box, thus we treat it as inter-object occlusion and set features of that cell to zero, otherwise the occluder is inside the window, implying self-occlusion and we do not change its feature, since we model this via rendering.

For out of sight, occluded or missing cells, after setting their features to zeros, we also append an extra bit flagging these special conditions to the end of feature vector, enabling SVM to learn a bias term.

Local Search We denote the camera is located at $(0, 0, 0)$, the CG model is rendered at (x_m, y_m, z_m) and it has size of (w_m, d_m, h_m) in meters. Then given a point cloud, to make the detector more robust, we only slide the windows on boxes whose center (x, y, z) is near the rendering position of the CG model, i.e., $x \in [x_m - 1.5 \times w_m, x_m + 1.5 \times w_m]$, $y \in [y_m - 1.5 \times d_m, y_m + 1.5 \times d_m]$, $z \in [z_m - 1.5 \times h_m, z_m + 1.5 \times h_m]$.

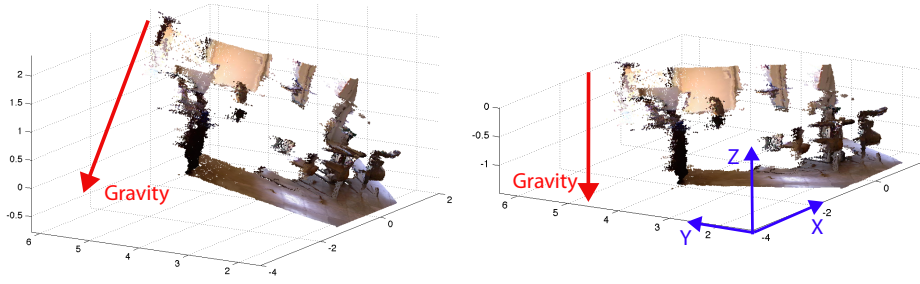


Fig. 1. Gravity alignment for the point cloud.

We further restrict the searching range with its relative location to the camera, such that the visibility of object surfaces is not dramatically changed.

If the model is rendered to the left of the camera, i.e. $x_m < 0$, the search range should not be on far right, so we require $x < 0.5w_m$; if $x_m >= 0$, then $x > -0.5w_m$. Similarly for the height, if $y_m < 0, y < 0 + 0.5h_m$, if $y_m >= 0, y > 0 - 0.5h_m$.

2 Data Details

In this section we provide more details on our training and testing data, as well as the ground truth labeling and classification.

Gravity direction Our algorithm takes an RGB-D image with the gravity direction as input. Aligning the point cloud and CG model with the gravity direction enables the axis-aligned sliding window for detection. The gravity direction can be obtained via many ways. For example, if a RGB-D camera is mounted on a robot, we know the robot's configuration and its camera tilt angle. For the cameras on the mobile device, like the cell phone we can use the accelerometer to obtain the gravity direction and camera's relative tilt angle during capturing. For this paper, the gravity direction for the RMRC data is provided. For dataset without ground truth gravity direction, it is not difficult to compute by fitting planes to the floor and walls. Fig. 1 shows the original point cloud and the point cloud with $-z$ axis aligned to gravity direction.

Depth refinement Raw depth image captured by Kinect has many missing values. To refine the depth, the RMRC data set uses the code from [1] to obtain a better depth map, integrating from other frames. The refinement step uses TSDF to voxelize the space and accumulate depth value from neighboring frames. After that, we use ray casting to obtain the refined depth map for the original camera pose. We define the neighboring frames to be frames at most 10 frames apart from current frame, so their time difference is less than a second, and the depth map can still be treated as a single image.

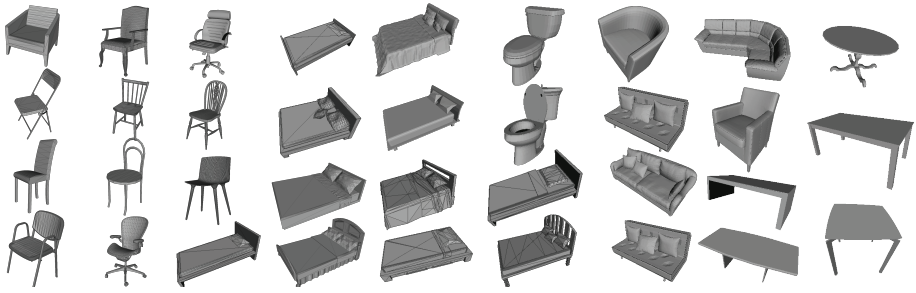


Fig. 2. All CAD models we used to train our sliding shape detector for the three category: chair, bed, and toilet

RGB-D ground truth We find that there are many inaccurate labeling in the ground truth of RMRC data [2, 3] , and many object categories contain a very small number of instances. Therefore, we only chose three categories that have more instances, namely chair, toilet, and bed, to evaluate our algorithm, and manually corrected all annotation mistakes in their ground truth labeling.

To better analyze the behavior of different algorithms, same as PASCAL VOC [4], we identify the difficult cases. We further divide these boxes into 3 types: out of sight, heavy occlusion and vast missing depth. The detailed criteria for such division is as follow:

Out of sight: Each 3D ground truth box is projected to 2D (denoted as B_{2D}). Then we define ratio $R_{in} = \frac{\text{area}(I \cap B_{2D})}{\text{area}(B_{2D})}$, where I is the 2D image. If $R_{in} < 0.5$, the ground truth box will be labelled as out of sight.

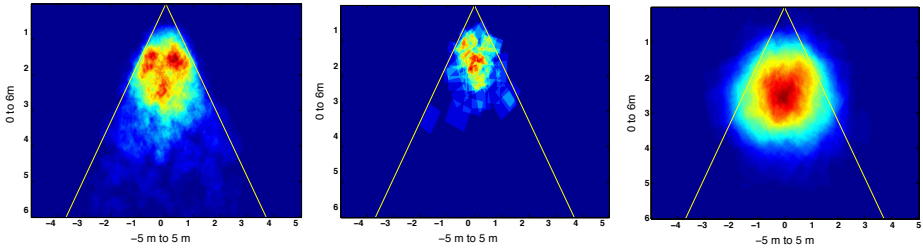
Heavy occlusion: We define occlusion ratio of a projected ground truth bounding box B_{2D} as $R_{\text{occ}} = \frac{||\{\mathbf{x} \in B_{2D} \cap d_{\mathbf{x}} < D_{\min}\}||}{||\{\mathbf{x} \in \cap B_{2D}\}||}$, where $D_{\min}$ is the minimal depth value of the 3D ground truth bounding box, $d_{\mathbf{x}}$ is the depth value of the 2D pixel $\mathbf{x}$, and $|S|$ is the cardinality of set S. If $R_{\text{occ}} > 0.5$, the ground truth box will be labelled as heavily occlusion.

Vast missing depth : The missing ratio is defined as $R_{\text{miss}} = \frac{||\{\mathbf{x} \in B_{2D} \cap d_{\mathbf{x}} \text{ is missing}\}||}{||\{\mathbf{x} \in \cap B_{2D}\}||}$. If $R_{\text{miss}} > 0.6$, the ground truth box will be labelled as vast missing depth.

Ground truth boxes not labelled by above criteria are defined as normal and are included in the 2D and 3D evaluation. All ground truth labelled as out of sight, heavy occlusion or vast missing depth are regarded as difficult cases and are only included in the 2D+ and 3D+ evaluation.

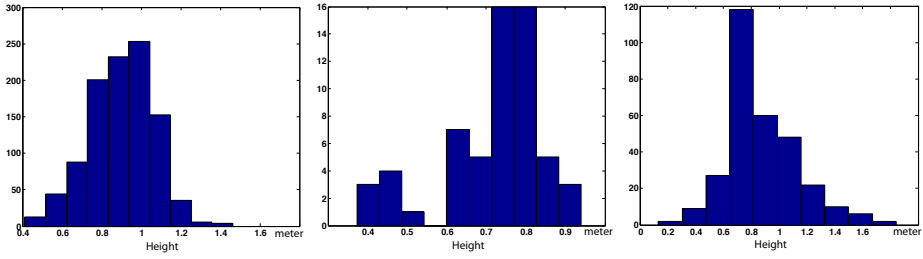
CAD models Fig. 2 shows all CAD models we used to train our sliding shape detector for the three categories, which are downloaded from google warehouse.

RGB-D statisitics To bridge the gap between CAD models and RGB-D image, we want to render the CAD models so that it represents the typical locations, sizes, and viewpoints in the RGB-D domain. Therefore, using the RGB-D training set, we obtain the statistics for object locations related to the camera (Fig. 3), object heights (Fig. 4),



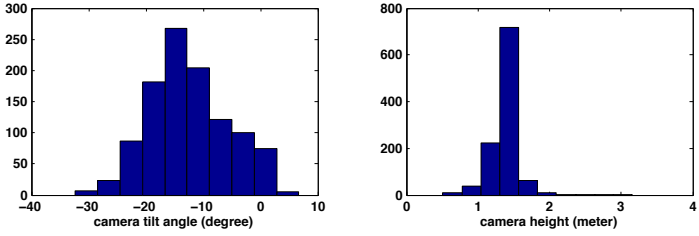
(a) chair (b) toilet (c) bed

Fig. 3. Top view of the location distribution of different category in RMRC training set.



(a) chair (b) toilet (c) bed

Fig. 4. Height distribution of different category in RMRC training set.



(a) camera tilt (b) camera height

Fig. 5. Camera tilt angle distribution in RMRC training set.

camera tilt angle and camera height (Fig. 5). And we render the CAD models using the modes of these distributions.

3 More Results

For testing, it takes about 2 second per detector to test on an RGB-D image using Matlab. The computation is naturally parallelizable except the non-maximal suppression at the end of detection. For training, it takes 4 to 8 hours to train a single detector with single thread in Matlab. But again, the training is naturally parallelizable, and can be easily done using Amazon EC2. In our case, we use a cluster with 500 CPU cores.

More results and quantatative comparison between Sliding Shape (ours), DPM, and Kinect (our algorithm trained on Kinect data) is shown in Fig. 6, 7 and 8. We set the

threshold on their detection scores so that different algorithms have the same recall on 2D+ or 3D+ ground truth. The recall threshold is set to be 0.2 for chair, 0.5 for toilet and 0.25 for bed. In general detectors has a higher recall for toilet than chair and bed. The thickness of the bounding box is proportion to the detection score (thicker box indicates higher score). The results demonstrate that under the same recall, Sliding Shape detector achieves much higher precision. Note that DPM fails when the image is too dark or there is background clutter.

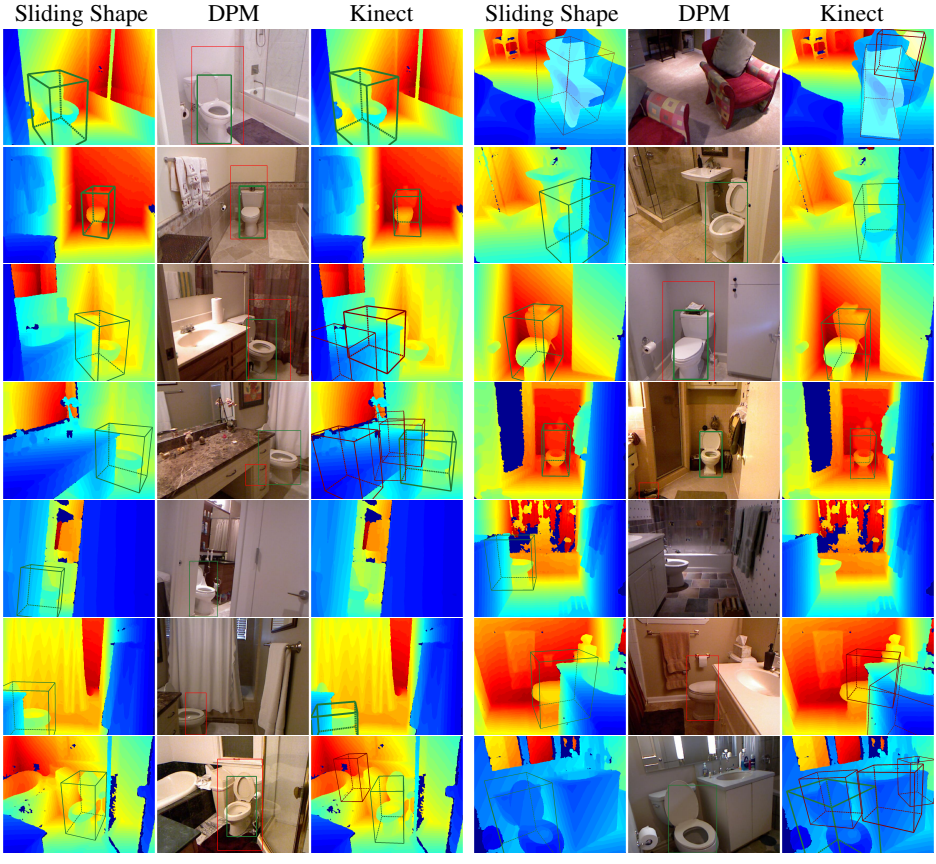


Fig. 6. Comparision for toilet detectors.

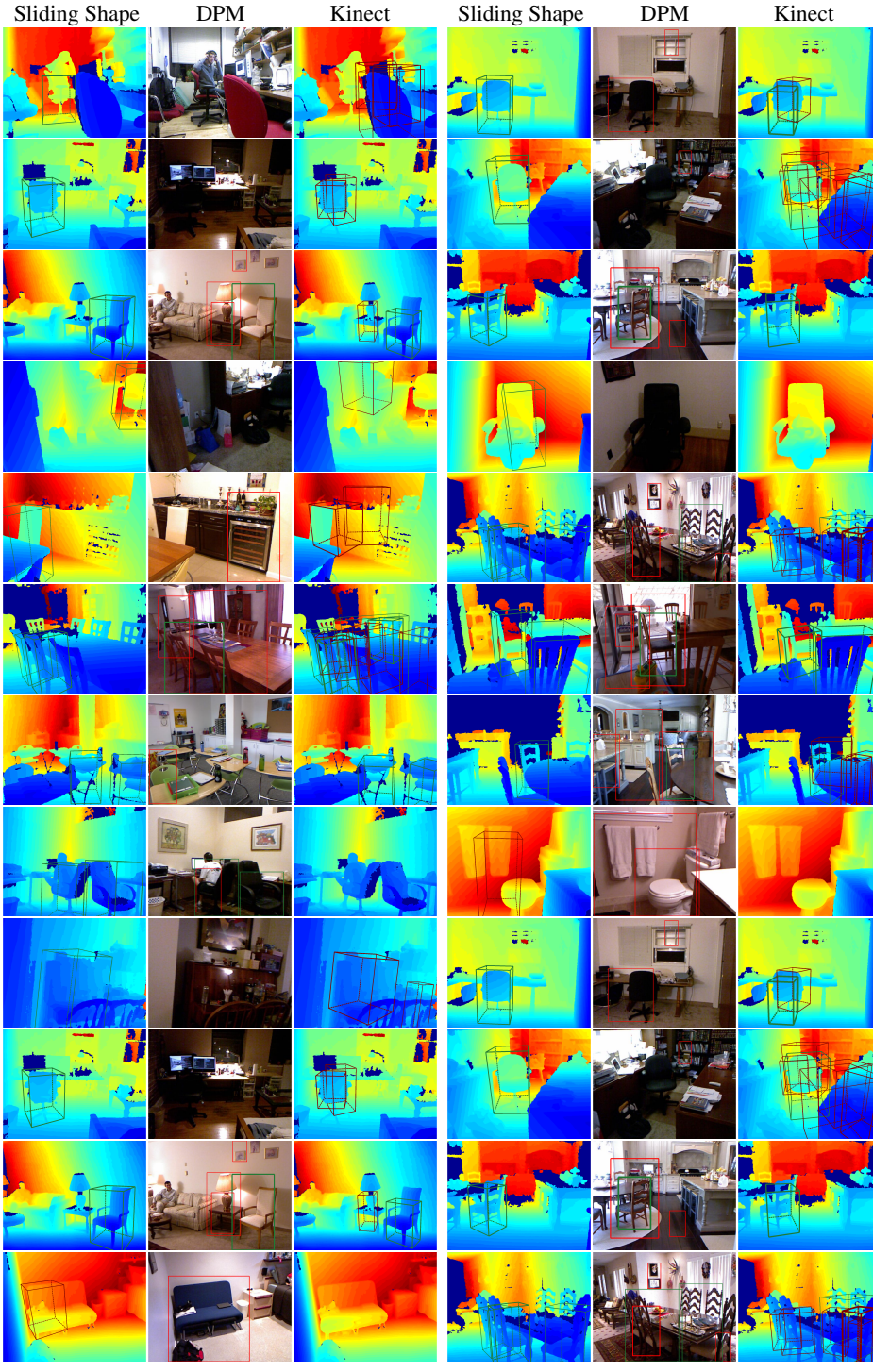


Fig. 7. Comparision for chair detectors.

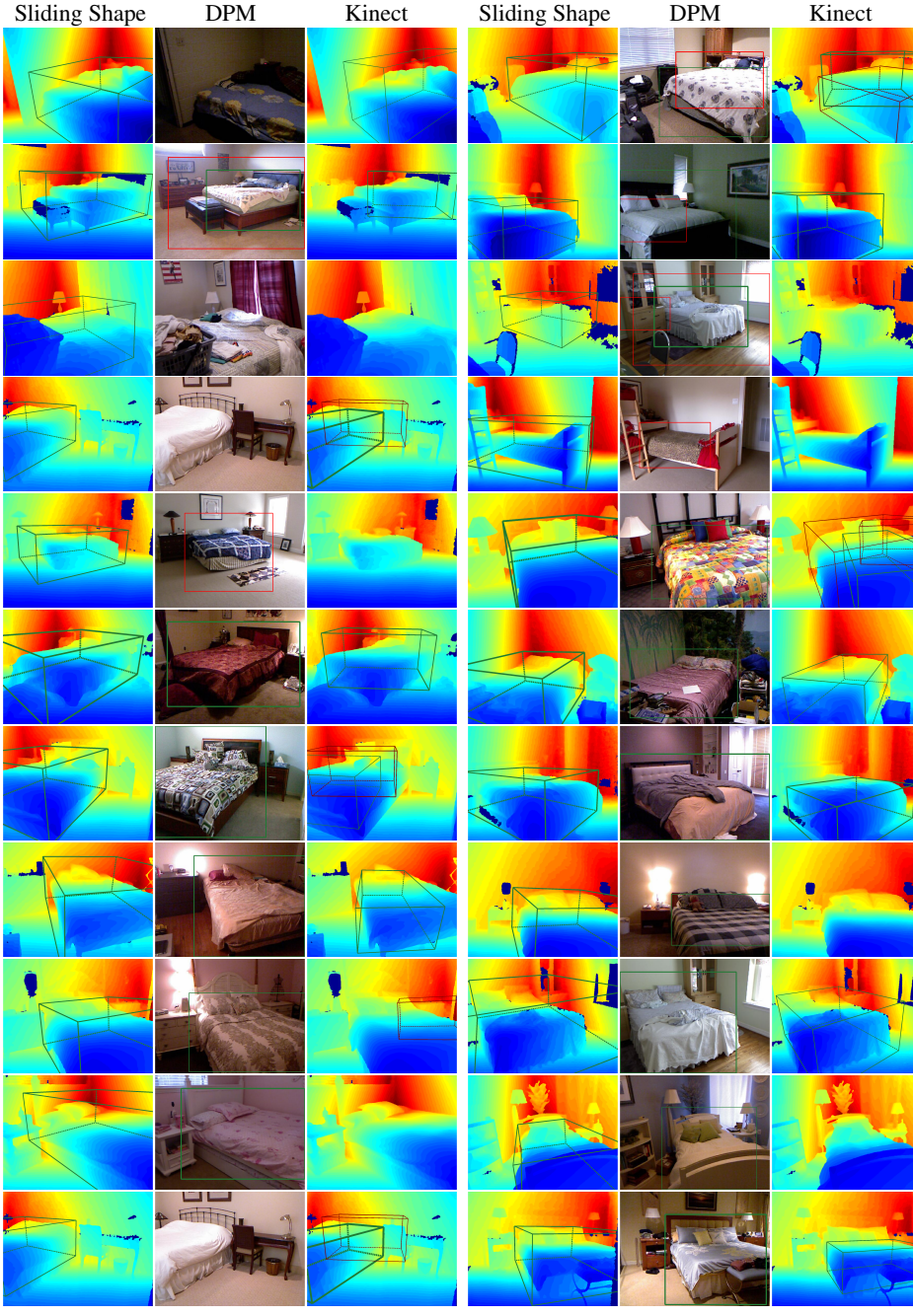


Fig. 8. Comparision for bed detectors.

References

1. Xiao, J., Owens, A., Torralba, A.: SUN3D: A database of big spaces reconstructed using sfm and object labels. In: ICCV. (2013)
2. Nathan Silberman, Derek Hoiem, P.K., Fergus, R.: Indoor segmentation and support inference from rgbd images. In: ECCV. (2012)
3. Guo, R., Hoiem, D.: Support surface prediction in indoor scenes. In: ICCV. (2013)
4. Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The pascal visual object classes (voc) challenge. (2010)