Investigating Voxel Carving for 3D Reconstruction of Occluded Objects in a Cluttered Container

Kateryna Pistunova
Stanford University
kpistunova@stanford.edu

Jacqueline Yau
Stanford University
jyau@stanford.edu

Shiva Badruswamy
Stanford University
shivalgo@stanford.edu

Abstract

Recently, object detection, recognition and retrieval has developed as one of the most important sub-fields of robotics and computer vision. While a lot of work has been focused on visible object recognition, detection and retrieval of occluded objects and objects in structured indoor environment, such as rooms, containers and shelves, is an ongoing problem. In this work, we apply voxel space carving to synthetically generated 3D objects on shelves to improve object recognition in such indoor environments. The resulting space carving can be used as a middle step in future neural network based 3D object detection techniques.

1. Introduction

In the modern era where robotics has become ubiquitous, one of the most common challenges that robots face is object manipulation and retrieval. For instance, Amazon has been using mobile robotic fulfillment systems since 2012 and is moving towards fully automated shipping warehouses. Their model relies on using mobile robots that follow bar-code stickers on the floor to retrieve stacks of objects and bring it to a human operator to pick the items. For a fully automated system, the robot would need to recognize the objects and be able to retrieve the correct target, subject to performance goals. In less structured environments, such a task may be further complicated by clutter that partially or completely occludes the target. Furthermore, the objects may not always be arranged stacks or in a table-top configuration, but may instead be located on a shelf or in a box.

In this project, we will explore improving mechanical search on dense, cluttered objects in a shelf, by using computer vision techniques (specifically space carving) to enhance the mid-level representations fed to the core reinforcement learning algorithms that issue actions to the robot.

2. Background/Related Work

Object detection, recognition and retrieval is a rapidly growing sub-field of robotics and computer vision. Recently, researches have made progress in detecting and retrieving objects from clutter/partially occluded objects. Previous work [6] included using reinforcement learning to teach a robot a sequence of movements to retrieve a partially occluded object that is detected with the help of a mid-level representation consisting of a position image masked by the segmentation of the target object. Another relevant paper [5] explores detection and recognition of occluded objects in organized indoor environments such as shelves. The authors employ a neural net architecture that consists of scene nodes containing the geometrical information about rooms, containers and objects as well as a natural language description of the scene. Some other previous works used either only semantics [1], [17], [12], [10] or geometry [2], [18] to generate scene graphs.

3. Approach

Our approach to enhance mechanical search is to perform space carving on the objects on the shelf so that the robot can better identify the objects. Our method is as follows: 1) Place objects onto a container shelf, 2) obtain multiple camera views of the scene, 3) retrieve relevant information for space carving (camera intrinsics, perspective matrix, view matrix for each new camera placement, each object's pose translation and pose rotation, etc.), and 4) perform space carving on the objects from various viewpoints.

3.1. Data

The raw data is a set of 3D containers and a set of 3D objects, which were provided by Andrey from [5]. Each container contains URDF, OBJ, MTL, and YAML files that describe the container's properties (origin, mass, mesh, shelf height, size, texture, etc.) that are loaded into iGibson.

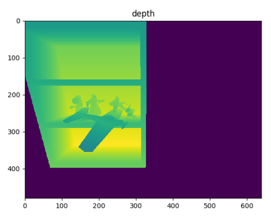
Likewise the objects' mesh, texture, and other properties are also defined by URDF, YAML, OBJ, and MTL files.

3.2. Methods

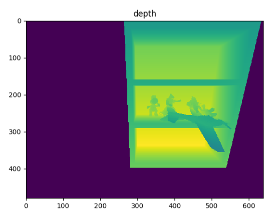
We adapted the hierarchical mechanical search (HMS) code from [5] to construct the shelf scene using the "generate_shelf_obj_placements.py" script, by having a robot from the iGibson and PyBullet environment attempt to place objects onto a shelf container. Given a container and a set of objects, the script will try various orientations of the objects in an attempt to place them onto the shelf (Fig. 4).

For our experiments, we placed five objects (action figurines) onto the container (a normal shelf). One example of how we placed some Nintendo action figures like Mario onto a normal shelf is visualized in the figure below (Fig. 5) with some of the objects placed onto the shelf. Initially, we had two camera views for each object: the camera placed at the mean location of all objects and the camera placed at the location of each specific object. We would retrieve the camera intrinsics, RBG image, depth map, segmentation mask for each object, and camera location. As we were unable to really carve with that information, we additionally retrieved the projection (perspective) matrix, view matrix, light perspective, and light view matrices. Also, we moved the camera to various other positions: translating along each x,y,z-axis by 0.2 and -0.2 (for 6 additional camera placements Fig. 1), and a randomly placed camera position.

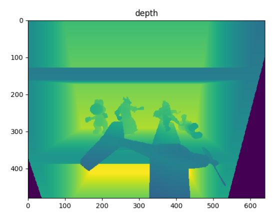
It is important at this point to briefly illustrate how the rendering engine, iGibson [14], returns various matrices needed for projection of 3D points back into 2D image space. The iGibson renderer is a light weight renderer that wraps around openGL renderer, and, as such returns matrices in the same format as openGL's. Essentially, iGibson-openGL returns a 4×4 matrix called the View matrix which is the object space matrix. It also returns a 4×4 rotational and a 4×4 translation matrix, which we use to compute the ModelView matrix 'M'. ModelView Matrix 'M' is the transformation of a 3D point in object space to camera space. iGibson-openGL also supplies the 3D camera space to 2D image space 'projection' matrix 'P'. We then compute the end-to-end 3D$\rightarrow$2D projection matrix by dotting 'P' and 'M'. The pipeline is illustrated below (Fig. 2):
The computation of the end-to-end projection pipeline was tricky as the matrix computations are based on the frustum camera model, a model whose impact we are unsure of on the overall carving technique. The end-to-end projection model for a frustum based camera according to openGL's



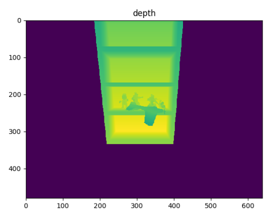
(a) x-axis translate add 0.2



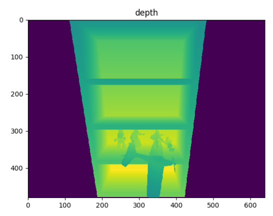
(b) x-axis translate subtract 0.2



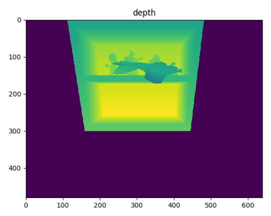
(c) y-axis translate add 0.2



(d) y-axis translate subtract 0.2



(e) z-axis translate add 0.2



(f) z-axis translate subtract 0.2

Figure 1: Camera placements from translations with respect to mean object location

literature is as follows:

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

where f and n are far plane and near plane distances, respectively, from the camera center along the optical axis as shown in figure 3 below. OpenGL projects a 3D point to the near plane. Further, depending on whether the viewing volume is symmetric or not, the frustum projection model changes. A challenge in iGibson was to be able to get the nature of the frustum viewing volume and its corresponding parameters. So, as an alternative, we computed the projection model as described in the pipeline in Fig. 2 , rather than directly through imputation of specific values for the 6 degrees of freedom required for the frustum perspective projection matrix. Further, this projection matrix is of shape 4×4, rather than the traditional matrix with a shape 3×4, and we would have to ensure that the dot product of the frustum matrix and the ModelView matrices were compatible. Given the unclear notations used in HMS code

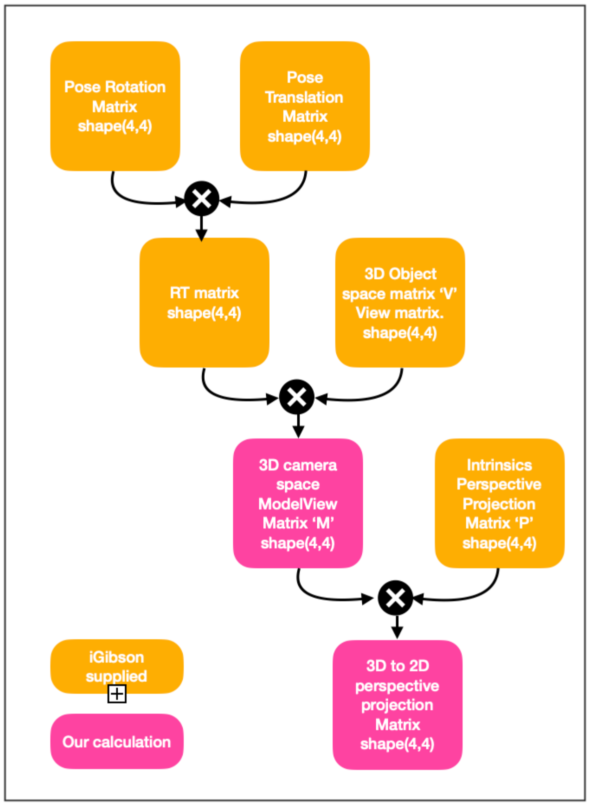

Pose Rotation Matrix shape(4,4)
Pose Translation Matrix shape(4,4)
RT matrix shape(4,4)
3D Object space matrix 'V' View matrix. shape(4,4)
3D camera space ModelView Matrix 'M' shape(4,4)
Intrinsics Perspective Projection Matrix 'P' shape(4,4)
iGibson supplied
Our calculation
3D to 2D perspective projection Matrix shape(4,4)

Figure 2: Computation of 3D to 2D perspective projection matrix

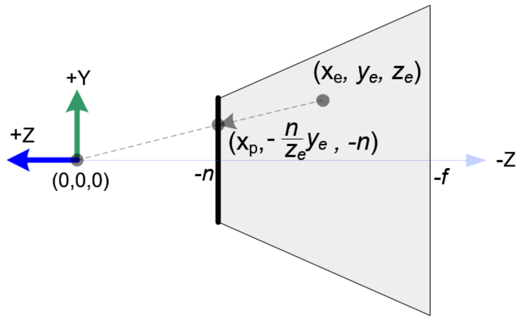


Figure 3: Side view of the frustum camera model in openGL. Image courtesy: http://www.songho.ca/opengl/gl_projectionmatrix.html

as well as in iGibson renderer, our computation of such dot products needs more validation checks.

Given that our projection matrix computation methodology required collection of several intermediate matrices and

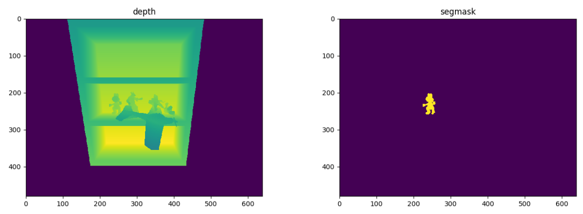


Figure 4: Example of depth map and segmentation mask for one object of the shelf scene from HMS output

careful assessment of shapes, our final tweaks to HMS code was to recover verifiable matrices such as each object's pose in the camera reference frame, the pose translation, pose rotation, and the 6 degrees of freedom pose array [Note: iGibson returns a 7 element array] itself. As explained in the computation chart above, the ModelView matrix 'M' (eq. 1) is then computed as follows:

$$M = VT^T R^T \tag{1}$$

where V is the view matrix, T is the object's pose translation, and R is the object's pose rotation.

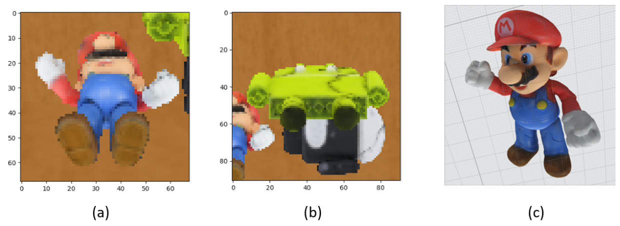


Figure 5: Mario figure object: (a), (b) examples of objects (Mario action figure, Android action figure) generated on a shelf. (c) the 3D model of the Mario action figure

We perform space carving for our experiments, using the information from the HMS code output. To implement space carving, we would first create a voxel prism around the 3D object. We would then place cameras with known camera matrices (obtained from the HMS output camera intrinsics matrix) around the object. After that, we find the voxels that are within the bounds of the silhouette and the final voxels that are within the silhouette itself. Our goal is to combine the views from multiple cameras, to obtain a reasonably good baseline space carving of the object. The accuracy will be limited by the fact that we can only utilize cameras from one side of the voxel cube, so some parts of the object will be unreachable to the camera. However, as expanded on in the Experiments section later, there were some difficulties with carving from multiple viewpoints using the output from iGibson.

Carved voxels are visualized by finding 3D surfaces from the carved volumetric data using Lewiner et al's approch to Marching Cubes algorithm [8].The algorithm is an improved version of Chernyaev's Marching Cubes 33 algorithm [15]. In contrast with Lorensen et al. approach [11] to the same algorithm, Lewiner et al. algorithm is faster, resolves ambiguities, and guarantees topologically correct results.

4. Experiments

We ran our experiments by attempting to perform space carving on the shelf scene with objects generated from the HMS code, using the various camera placements we had tried. For the most part, computing the projection matrix to transform from 3D to 2D took most of our efforts in the following experiments. The dataset used, as mentioned in the Approach section, is from the set of containers and objects provided to us from [5]. Currently we measure our results qualitatively, by looking at the carving result visually. If we were able to successfully carve, we would have wanted to compare against a baseline from performing the space carving on Blender (expanded on in the Conclusion section).

Some possible quantitative metrics we were considering were some type of reprojection loss between the two carving results, like silhouette-based loss functions, and/or some volumetric loss [4]. For the silhouette-based loss, the idea is that a 2D silhouette projected from the reconstructed volume, under certain camera intrinsic and extrinsic parameters, should match the ground truth 2D silhouette of the input image. The loss is then (eq. 2)

$$L_{proj}(I) = \frac{1}{n}\sum_{j=1}^{n} d\left(P(f(I);\alpha^{(j)}), S^{(j)}\right) \tag{2}$$

where I is the input image, $S^{(j)}$ is the $j-th$ ground truth 2D silhouette of the original 3D object, n is the number of silhouettes/views used for each 3D model, $P(.)$ is a 3D to 2D projection function, and $\alpha^{(j)}$ are the camera parameters of the j-th silhouette, from [4]. The distance metric $d(.)$ could be L2 or L1 distance.

For the volumetric loss (eq. 3), the idea is to capture the distance between the reconstructed and ground-truth volumes

$$L_{vol}(I) = d(f(I), X) \tag{3}$$

where I is the input image and X is ground-truth volume of the 3D object for some distance metric (like L2) [4].

We implemented the space carving algorithm described in the previous section using the objects generated from the HMS code. One constraint to note is that since space carving is conservative, objects must not be occluded from the camera view by the shelf, otherwise they will be carved out in the final carving. Fig. 6 shows a typical camera view when objects are occluded by the shelf. As seen from the figure, the segmentation mask is missing the occluded objects and such camera views therefore need to be excluded for conservative space carving.

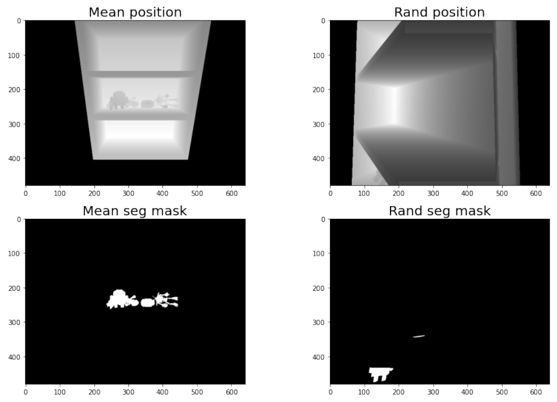


Figure 6: Occluded example of depth maps and segmentation masks for different object placements.

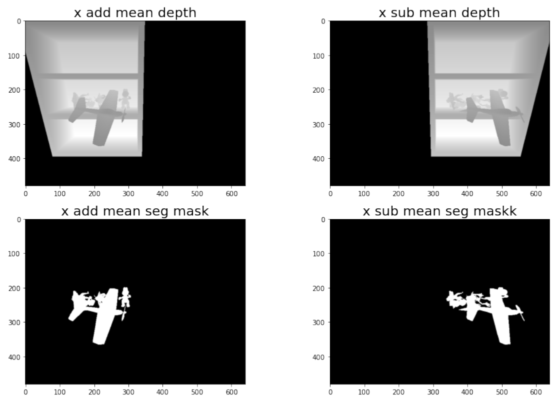


Figure 7: Non-occluded example of depth maps and segmentation masks for different object placements.

On the other hand, Fig. 7 shows camera views where all objects are visible. Such views can be included in the final space carving and are the type of views we generated and selected.

We have successfully implemented carving with one camera as shown in Fig. 8 where the top two insets represent a single view voxel carving from different angles while the bottom two insets show the corresponding segmentation mask and depth map. We can clearly see that carving represents the silhouette of the placed objects, generated by

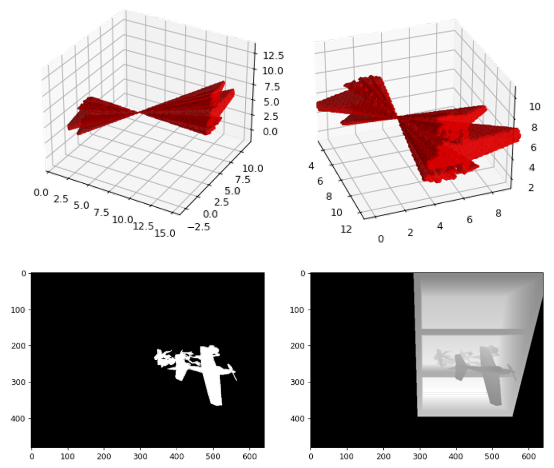


Figure 8: Single-view voxel carving from different angles. Bottom: corresponding segmentation mask and depth map.

the segmentation mask.

Our next step was to generalize space carving to many different cameras that contain objects unclouded by the shelf. However, we ran into some issues when trying to generate carving from multiple viewpoints. Specifically, we have had a lot of trouble trying to calculate the projection matrix to transform the points from 3D to 2D. Initially, we had tried using the projection (perspective) matrix from the iGibson renderer, one of the first outputs retrieved from the HMS output, but that unfortunately did not work, as we found out that when the camera positions shifted, the perspective matrix did not change. Next, we had discovered that the view matrix V did change with each new camera placement, so we assumed using V as the camera extrinsics and tried to calculate the projection matrix by multiplying the intrinsics (K) and extrinsics together (eq. 4)

$$P = KV \tag{4}$$

but as V is a 4×4 matrix, we removed the final column, which seem like homogeneous coordinates, to make V into 3×4. However, we ran into the issue of the camera seeming to stay in place while the objects moved around. We experimented with numerous other methods to attempt to calculate the correct projection matrix. In the next attempt, we discovered that each object, its own Instance class in iGibson, has its pose information stored as a 4×4 matrix (p_i for object i), along with the object's pose translation (T) and rotation (R), so we tried calculating the projection matrix as (eq. 5)

$$P = Kp_i \tag{5}$$

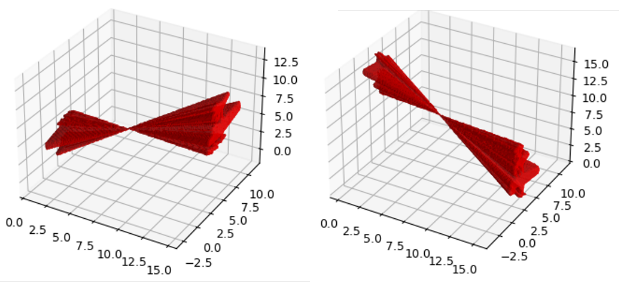


Figure 9: Single-view voxel carving from different angles. Bottom: corresponding segmentation mask and depth map.

where p_i is the object pose matrix for some object i for $i \in [1, 5]$. Similarly, we remove the last column of p_i to make it a 3×4 matrix. As this was unsuccessful as well, we next tried using the ModelView matrix (eq. 1) to calculate the projection matrix (eq. 6)

$$P = KM \tag{6}$$

which unfortunately was still unable to carve properly. Our final attempt was to use the perspective matrix (P') with the ModelView matrix to calculate the projection (eq. 7)

$$P = P'M \tag{7}$$

but this ultimately led to erroneous results as well.

Fig. 9 shows a single view carving done from multiple cameras. We can see that the camera seems to stay in place while the objects are moved around while one would expect the opposite. This same issue occurred for all of our attempts at computing the projection matrix as listed before when trying to carve from multiple viewpoints and thwarted our efforts to try any of the other future ideas we had originally planned (using color for segmentation, etc.). A possible solution to this confusion would be to look for a better camera projection matrix we can get from the HMS code, or use manually created cameras and 3D scenes as outlined in the Conclusion section below.

5. Conclusion

From our experiments and experience tracing through how the iGibson renderer works, we have learned that converting from iGibson to the projection matrix we required understanding some subtleties between the matrices that iGibson produces and the projection and camera matrix we recognize and need for space carving. One important learning was that voxel carving using a projection matrix under the frustum camera model requires an understanding of how different formats of projection matrices impact the carving process. For instance, as can be seen from the partial carves, the camera appears to be always located in between two image planes. We would have to investigate how the near-far plane nature of the projection matrix impacts this result, and if a successful voxel carving is at all possible in openGL's camera models. If iGibson-openGL render is not amenable for carving, then we could use other objects and renderers to do space carving on. Given that we would also need the camera matrices from multiple views, we could accomplish it in several ways:

a) Obtain 3D objects suitable for a shelf from either the dataset Andrey provided (see previous section for trouble with running the code) or from an open source dataset, simulate our own camera matrices (or use the ones in the carving homework problem), get the 2D projections of the 3D object for each camera, input the 2D images into the frames.mat code to replace the bird images, achieve space carving and compare it to the original 3D object.

b) A potential future work involves the use of Blender [13] as a renderer to set up flexible object scenes and cameras with different camera matrices in its enabling GUI. We installed an experimental pipeline to recreate in Blender scenes generated with the HMS code including the chosen camera positions. Fig. 10 shows a 3D generation of the recreated scene. Further advantages of Blender include simply adjusting light positions to readily generate silhouetted images, as shown in Fig. 11, and the ability to generate bounding boxes on the fly to generate the initial coarse voxel hull, as shown in Fig. 12.

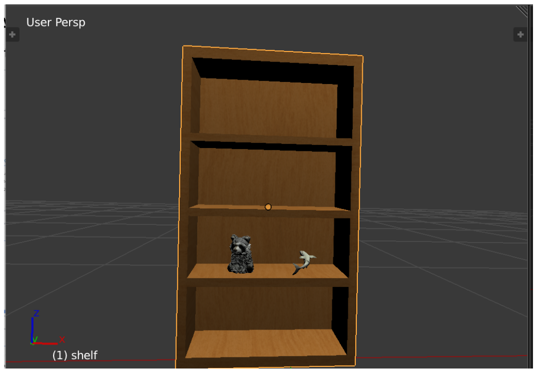


Figure 10: Recreated scene in Blender with objects and container from the HMS code repository

c) calibrate our own cameras and get real life images, similar to the ones used to carve out the bird structure

Our next goal would be to extend space carving to multiple objects, as well as objects clustered together. Space carving, however, would not be able to distinguish between

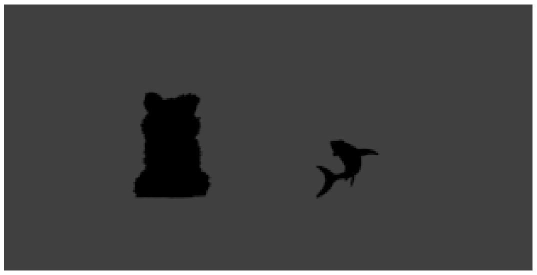


Figure 11: A generated silhouette in Blender

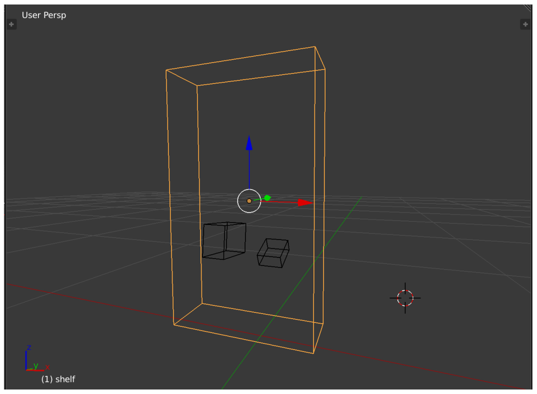


Figure 12: A generated bounding box in Blender

objects that are placed too close to each other. One solution is to use voxel coloring, provided the objects are lambertian and of different color. Another possibility is to use shadow carving by placing light sources around the object and calculating the shadows. Such an approach would not only help with tightly clustered objects, but also with objects that have multiple concavities, as traditional space carving would fail to identify them. Once we have a reliable algorithm to carve the shapes of different objects in different orientations, we would feed the data into a 3D detection algorithm that we would either find an open source implementation of or code ourselves. One way to do this is to convert our carved voxels into a point cloud by using very small voxels and using the coordinate of each as a point since 3D detection algorithms typically use point clouds [16], [7]. While a logical step would be to use a 3D convolutional network algorithm, such methods are typically slow and require high computational power without prior pre-processing [3], [9]. More recent works organize voxels in sets of columns and encode each column into a feature encoding to form a pseudo-image which can then be used in traditional detection algorithms. A state-of-the art end-to-end approach called Voxelnet [19] directly operates on point clouds and produces 3D bounding boxes for the detected structure. The summary of the algorithm is diagrammed in Fig. 13.

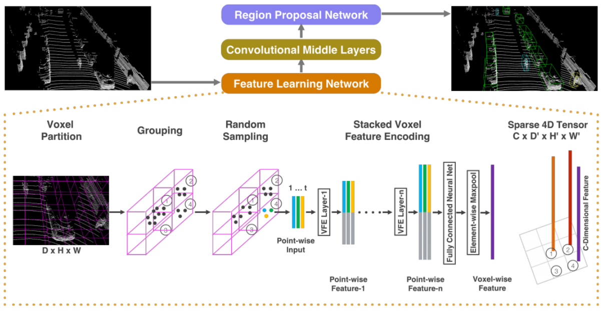


Figure 13: VoxelNet architecture. The feature learning network takes a raw point cloud as input, partitions the space into voxels, and transforms points within each voxel to a vector representation characterizing the shape information. The space is represented as a sparse 4D tensor. The convolutional middle layers processes the 4D tensor to aggregate spatial context. Finally, a RPN generates the 3D detection.

Inputting voxels already pre-processed by space carving could potentially speed up the algorithm. Our final steps would be to compare the detection speed, accuracy and localization with our space carving model and without it. We could use one of the open source algorithms mentioned above to benchmark the performance with and without the additional space carving step.

References

[1] J. Crespo, J. C. Castillo, O. M. Mozos, and R. Barber. Semantic information for robot navigation: A survey. *Applied Sciences*, 10(2), 2020.

[2] M. Danielczuk, A. Angelova, V. Vanhoucke, and K. Goldberg. X-ray: Mechanical search for an occluded object by minimizing support of learned occupancy distributions, 2020.

[3] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks, 2017.

[4] X.-F. Han, H. Laga, and M. Bennamoun. Image-based 3d object reconstruction: State-of-the-art and trends in the deep learning era, 06 2019.

[5] A. Kurenkov, R. Martín-Martín, J. Ichnowski, K. Goldberg, and S. Savarese. Semantic and geometric modeling with neural message passing in 3d scene graphs for hierarchical mechanical search, 2020.

[6] A. Kurenkov, J. Taglic, R. Kulkarni, M. Dominguez-Kuhne, A. Garg, R. Martín-Martín, and S. Savarese. Visuomotor mechanical search: Learning to retrieve target objects in clutter, 2020.

[7] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom. Pointpillars: Fast encoders for object detection from point clouds, 2019.

[8] T. Lewiner, H. Lopes, A. W. Vieira, and G. Tavares. Efficient implementation of marching cubes cases with topological guarantees. *Journal of Graphics Tools*, 8(2):1–15, december 2003.

[9] B. Li. 3d fully convolutional network for vehicle detection in point cloud, 2017.

[10] J. K. Li, D. Hsu, and W. S. Lee. Act to see and see to act: Pomdp planning for objects search in clutter. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5701–5707, 2016.

[11] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, Aug. 1987.

[12] Y. Qiu, A. Pal, and H. I. Christensen. Learning hierarchical relationships for object-goal navigation, 2020.

[13] rfabbri (https://blender.stackexchange.com/users/16590/rfabbri). 3x4 camera matrix from blender camera. Blender Stack Exchange. https://blender.stackexchange.com/questions/38009/3x4-camera-matrix-from-blender-camera (version: 2011-08-21).

[14] B. Shen, F. Xia, C. Li, R. Martın-Martın, L. Fan, G. Wang, S. Buch, C. D'Arpino, S. Srivastava, L. P. Tchapmi, K. Vainio, L. Fei-Fei, and S. Savarese. igibson, a simulation environment for interactive tasks in large realistic scenes. *arXiv preprint*, 2020.

[15] E. Tcherniaev. Marching cubes 33: Construction of topologically correct isosurfaces. 01 1996.

[16] Y. Wang and J. Ye. An overview of 3d object detection, 2020.

[17] Y. Wu, Y. Wu, A. Tamar, S. Russell, G. Gkioxari, and Y. Tian. Bayesian relational memory for semantic visual navigation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[18] Y. Xiao, S. Katt, A. t. Pas, S. Chen, and C. Amato. Online planning for target object search in clutter under partial observability. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8241–8247, 2019.

[19] Y. Zhou and O. Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection, 2017.

6. Supplementary Materials

Our modified HMS code, originally from [5]:
https://github.com/jhyau/HMS