A Voxel-Based Region Growing Approach For Detecting Support Surfaces and Supported Objects in Point Clouds

Carleton University, Geometry Processing Fall 20201

Nicolas Perez
nickperez@cmail.carleton.ca

December 10, 2021

1 Objective

1.1 Context

The aim of this project was to develop an algorithm to efficiently and effectively detect horizontally oriented planes and segment the objects placed on the horizontal planes in 3D point clouds (for example, a 3D point cloud of a bookshelf with objects placed on it). In the literature, this is most commonly refferred to as support surface detection (detecting horizontal planes) and supported object detection/segmentation (the objects placed on the horizontal planes) [5, 7, 14, 13, 12]. The algorithm includes a user-controlled threshold for minimum surface size, to ensure accurate detection of support surfaces. With 3D Lidar scanning now accessible using smartphones, there are an increasing number of opportunities for geometry processing to make its way into everyday life [9], and with the implementation of the proposed algorithm, many of those opportunities will be realized.

The algorithm may be used to acquire 3D scans of objects, for either practical or entertainment purposes. For example, someone could place a collection of objects on their kitchen table, scan them, reorient them, and then scan them again, obtaining a full 360-degree 3D model of the objects. One could use these models to advertise the items on a buy and sell website, where users could rotate and view the object from any angle. The 3D models could also give potential buyers insight into the dimensions and volume of the objects, to see if they could fit them in their car or their backpack, or to verify the object they want is the right size for their needs. Someone could use these models as objects in a video game, or other media such as visual art. The proposed algorithm could even be used to automatically enable realistic interactions in a virtual environment given by a scan of a real life indoor scene (i.e. a character in a video game being able to move objects around, destroy them, etc...). Interactable virtual worlds where movable objects in indoor scenes are almost exclusively placed on horizontal surfaces are common in many video games, one well known game in particular being Skyrim.

The algorithm could also be used in robotics. An autonomous robot exploring an environment must know what surfaces it can safely land/stand/move on. Additionally, segmenting out supported objects would give the robot information on the types of objects it can move and manipulate, and obstacles to avoid while moving on or close to supported surfaces.

1.2 Related Work

In this section, algorithms that detect planes using a region growing approach, detect support surfaces, or detect both support surfaces and supported objects from point cloud data are re-

viewed. Out of all the works that focus specifically on support surface fitting, some of them include supported object segmentation as well [5, 7, 14, 13, 12]. The approaches are organized into subsections based on the technique used.

1.2.1 RANSAC Approaches

[14] uses a RANSAC based approach, and segments planes that are both horizontal and vertical, also providing a 2D segmentation. [13] uses a RANSAC based approach for segmenting both the surfaces, as well as the objects.

1.2.2 Region Growing Approaches

[12] uses a growing region technique for detecting horizontal planes, [4], and estimates supported objects using rectangular axis-aligned cuboids. Their technique is also confined to a very small environment, a robot placing objects in a box. The point cloud is given only by a bird's-eye view perspective. [6] uses a voxel-based region growing technique to detect planes, but does not make the same data assumptions as the algorithm in this project. The algorithm most closely related to the one in this project is seen in [15], which uses a voxel-based region growing approach to segment roof tops of buildings. It also does not work under the same assumptions as the algorithm presented in this project.

1.2.3 Other Approaches

[5] identifies support surfaces using a 'bottom up' approach which estimates classes of individual points and then aggregates neighbouring points together to form more coherent classifications. [7] provides 2D segmentation, their approach makes use of a SVM that takes as input the summation of features of the point cloud at different heights.

1.3 Selected Approach

The algorithm presented in this project is a voxel-based region growing approach, followed by a k-nearest neighbors approach for segmenting objects placed on support surfaces. For detecting surfaces, every point in the point cloud is placed into its respective voxel, followed by a region growing approach where a given region is defined by the voxels it spans. For detecting supported objects, a region growing algorithm will group points into clusters using each points' k-nearest neighbors.

The method in this project is the first known voxel-based region growing approach used for detecting support surfaces. The justification for developing this technique is clear: Voxel-based region growing approaches have been shown to effectively detect planes but these approaches haven't been adapted to the constrained problem of support surface detection. The region growing method also inherently allows for support surface detection regardless of the shape of the surface (ex: a U-shaped table).

2 Methodology

The support surface detection algorithm requires user specified parameters: v_x,v_y, and v_z are the length,width and height of each voxel, p_{min} is the minimum number of points that must be inside a voxel to be considered a part of a surface and ψ_{min} is the minimum size a region must be to be considered a surface. The supported object detection algorithm requires user specified parameters: k is the number of k-nearest neighbors each point uses for growing a given region

and s is the sampling fraction for sampling the points which will be used in the region growing algorithm.

Pseudocode for the support surface detection and the supported object detection is layed out in this Section. For a much more in depth understanding, the reader is referred to the source code, which contains commenting/documentation to help the reader understand it (the source code also resembles pseudocode, because it is Python).

Algorithm 1 Voxel-Based Region Growing($\mathcal{P}$, v_x, v_y, v_z, p_{min},ψ_{min})

```
organize all the points in P into their respective voxels
while every voxel hasn't been visited do
    pick an unvisited voxel as the current region
    mark this voxel as visited
    while unvisited voxels adjacent to the region contain more than p_min points do
        add those adjacent voxels to the region and mark them as visited
    end while
    if size of the region is ≥ ψ_min then
        add it to the list of detected surfaces
    end if
end while
return the list of detected surfaces
```

The voxel-based region growing approach is outlined in simple pseudo code in Algorithm 1. Depending on a user-specified Boolean parameter, points with estimated normals that are not close to parallel with the z axis will or will not be considered in the support surface detection algorithm. If normals must be estimated, the chosen method is to estimate the normals of points in the point cloud using the *estimate_normals_simple* function from the GeomProc library. The function organizes the points into a k-d tree, and then each point has its normal estimated by using a covariance analysis of its k-nearest neighbors. Next, horizontal planes are detected using a voxel-based region growing approach.

Algorithm 2 K-Nearest-Neighbor-Based Region Growing($\mathcal{P}'$, k, s)

```
sample s|P'| points from P' and store them in P''
while every point in P'' hasn't been visited do
    pick an unvisited point as the current region
    mark this point as visited
    while there are unvisited points adjacent to the region do
        add those adjacent points to the region and mark them as visited
    end while
    add the resulting region to the list of detected objects
end while
return the list of detected objects
```

The k-nearest-neighbors-based region growing algorithm is described in Algorithm 2. It segments objects that are placed on top of the detected horizontal planes. The input to the algorithm is the set of points not classified as a part of a support surface, denoted as $\mathcal{P}'$. The points 'adjacent' to a point are the k-nearest points to that point. A point is determined to be 'adjacent' to the current growing region if it is at least one of the k-nearest points of a point within the growing region.

After objects are segmented, a surface reconstruction algorithm can be used to create a mesh out of the points for a detected supported object [1]. Additional analysis of the mesh may then

be carried out, such as calculating the volume of the scanned object [16].

3 Implementation

3.1 Main Components of the Algorithm

The main components of the algorithm are the estimation of normals, organizing points into voxels, the voxel-based region growing, and the k-nearest-neighbors-based region growing segmentation of supported objects. The algorithm also relies on the use of k-d trees for normal estimation as well as the supported object segmentation.

3.2 Programming Languages and Libraries

Python 3.8.8 and the GeomProc library were used.

3.3 Dataset

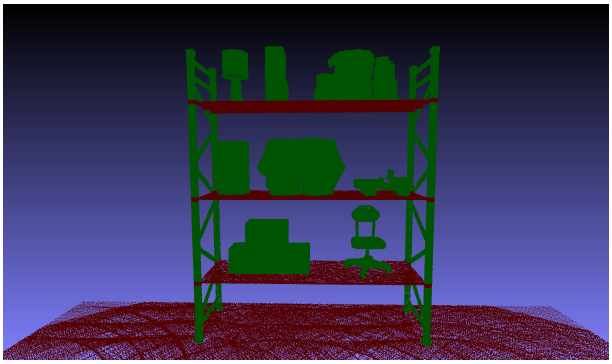
The assumption is that the point cloud the algorithm is operating on is given from a depth sensor that can also sense the direction of gravity (for example a lidar sensor with an IMU in it), so that support surfaces can be horizontal oriented prior to input in the proposed algorithm. This is not an unrealistic assumption, given there are now many modern SLAM and photogrammetry algorithms that include inertial sensors [3, 8, 2]. In the case that the data is not already aligned in such a manner, methods for aligning the data based off of estimated normals of the point cloud exist [7, 14]. The general formulation of the alignment algorithm is to rotate the point cloud based on some objective function that describes a good alignment. The objective function may be a squared error of the angle of each normal form the z axis, for example. This process may be done iteratively.

Each point cloud was acquired using AirSim's simulated lidar sensor [11] or by sampling points on the surface of a mesh. Ready-made 3D environments from Unreal Engine were used for lidar scanning using AirSim. Any mesh which was used for sampling points off of was downloaded from CGTrader.

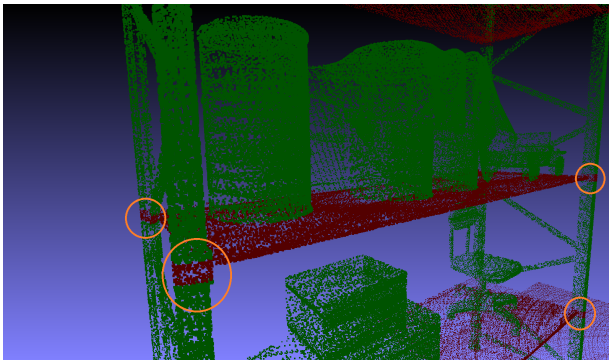
4 Results

To demonstrate the effectiveness of the method, resulting segmentation of point clouds are visually shown. Points classified as a part of a plane are colored red and points classified as not a part of a plane are colored green. Supported object detection is shown by comparatively excluding/including parts of the point cloud. There are many figures, and some of them are quite large, so the majority of the discussion is included as captions below the figures, to avoid making the reader scroll back and forth between the text and the corresponding figure(s). Unless otherwise specified, support surface detection results shown can be assumed to not have the filtering with estimated normals option enabled. Units for the point clouds given from simulated lidar are in meters. All tests were ran on a AMD Ryzen 5 3600, which is considered a mid to high tier processor at the time this report was written, December 2021.

Figure 1: AirSim environment of a shelf with miscellaneous items on it. Free assets from the Unreal Engine Marketplace were used. On the shelf, in order from left to right and top to bottom, there is: A lamp, a mini shelf, a stack of boxes, a barrel, a group of barrels with a tarp over them, a toy car, another stack of boxes and an office chair.

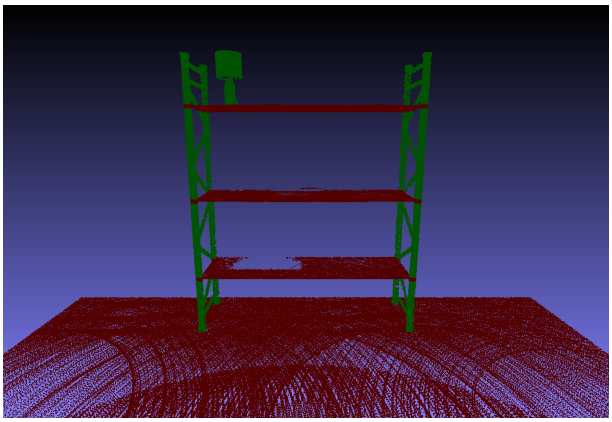


(a) Support surface detection results of a shelf with items on it.

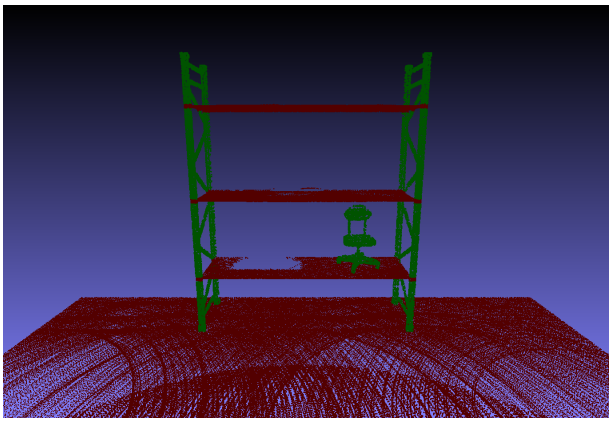


(b) A zoomed in view of the detection results showing false positives, indicated with orange circles.

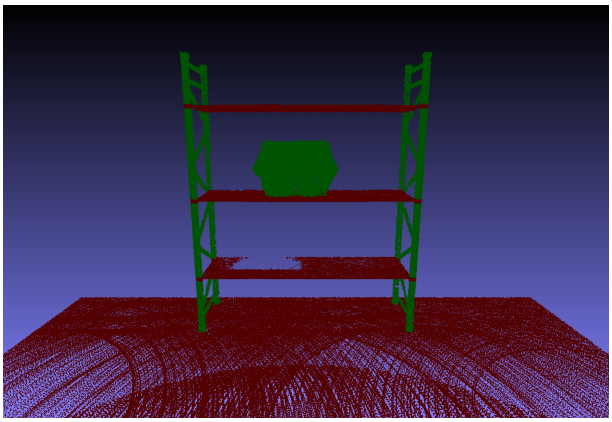
Figure 2: 2 figures showing different views for support surface detection results of a 3D point cloud of a shelf with items on it. The point cloud has $539,034$ points and was given by a simulated lidar scan of a virtual environment. The environment is shown in Figure 1. The algorithm ran without filtering points based on normals. It took 2.859 seconds to complete, with parameters $v_x = v_y = 0.1$ and $v_z = 0.07$, resulting in $363,957$ voxels to search through.



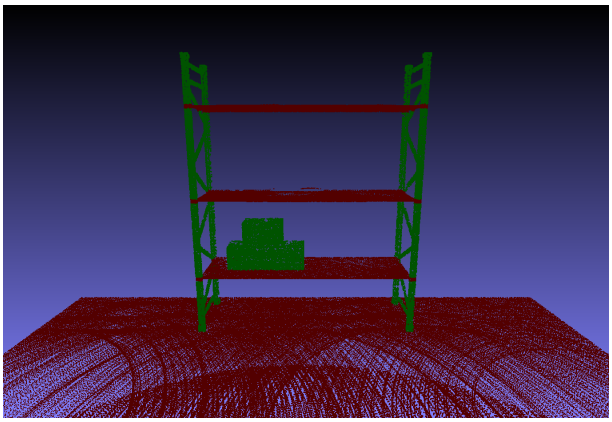
(a) A lamp being segmented.



(b) A chair being segmented.



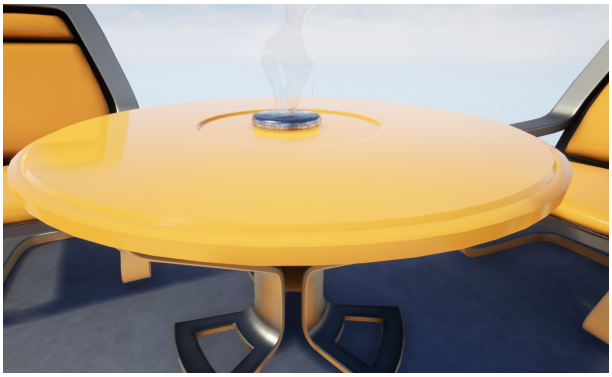
(c) A group of barrels being segmented.



(d) A stack of boxes being segmented.

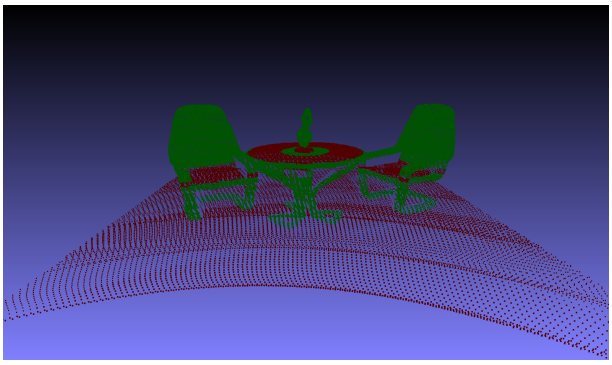
Figure 3: 4 Figures showing supported object detection of objects placed on a shelf. AirSim was used for acquiring the point cloud via a simulated lidar scan. The environment is seen in Figure 1. The only error in the results was that individual items were not segmented if they were touching each other, in particular, the group of barrels and the two stacks of boxes. The algorithm took 187.00 seconds to complete, with parameters $s = 0.3$ and $k = 50$. The k parameter had to be set very high, or else the lamp would be segmented into 2 separate parts. Excluding that one shape which is very irregular, the k can take on much lower values and still get perfect results.

(a) 2 chairs and a table in AirSim.

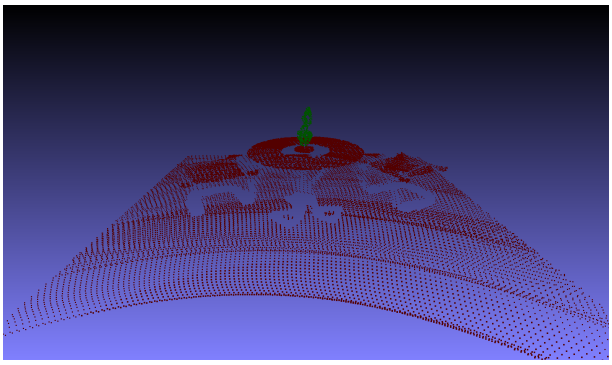


(b) A zoomed in view of an environment in AirSim showing a slight dip in the surface of the middle of a table.

Figure 4: 2 figures showing different views of an environment in AirSim that was used for acquiring a simulated lidar scan. The scene is to scale.

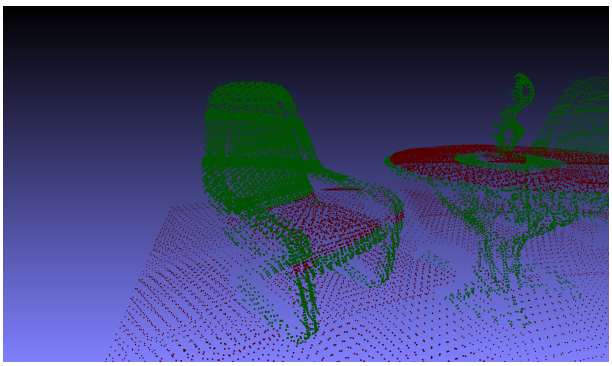


(a) An example of support surface detection on a point cloud with $108,025$ points. The algorithm took 0.625 seconds to complete, with parameters $v_x = v_y = 0.1$ and $v_z = 0.05$, resulting in $55,768$ voxels to search through.

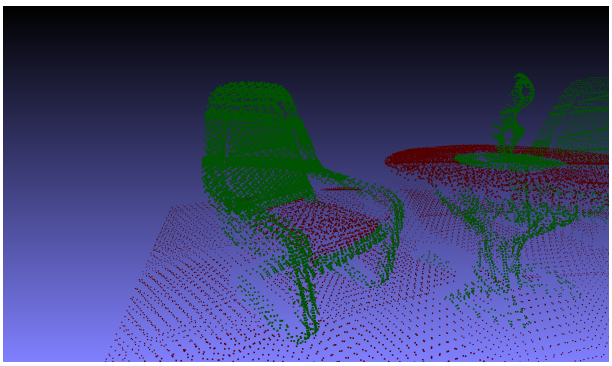


(b) An example of supported object segmentation. The object on top of the table is segmented out perfectly. The algorithm took 12.75 seconds to complete, with parameters $s = 0.3$ and $k = 20$. The sparseness of the object is due to the down sampling of the algorithm.

Figure 5: 2 figures showing the (a) support surface detection and (b) object segmentation of the object placed on top of the table. The point cloud was acquired using AirSim and an environment with 2 chairs and a table (see Figure 4). In (a) we can see that the dip in the table, as seen in Figure 4b may give false negatives. Increasing v_z can help handle cases like this.

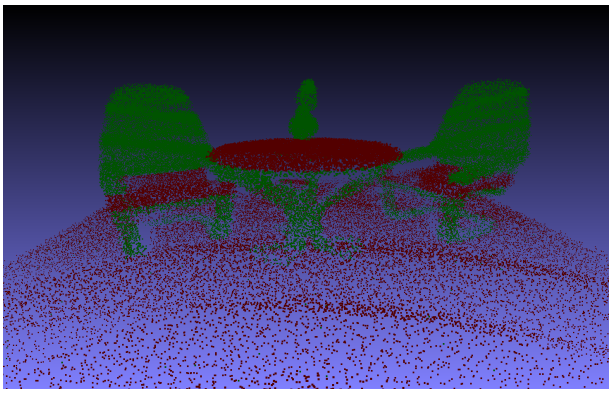


(a) An example of support surface detection on a point cloud without filtering points based on estimated normals.

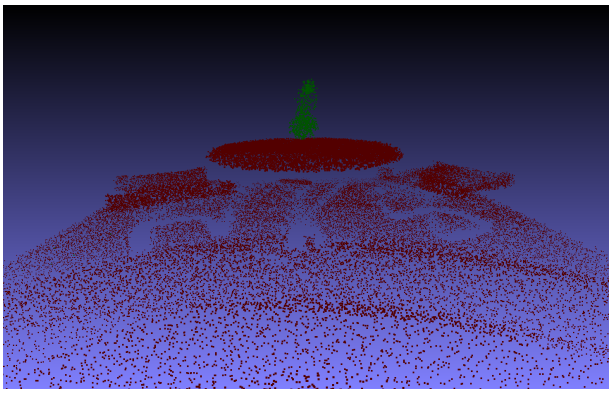


(b) An example of support surface detection on a point cloud with filtering points based on estimated normals.

Figure 6: 2 figures showing the same support surface detection as in 5, but zoomed in to show the difference between (a) not filtering points based on estimated normals and (b) filtering points based on estimated normals. Estimating all the normals prior to support surface detection took 160.41 seconds, using $k = 30$ neighbors for each point to estimate the normals. (a) has false positive points on the legs of the chair. (b) has no false positives but has more false negatives on the table.

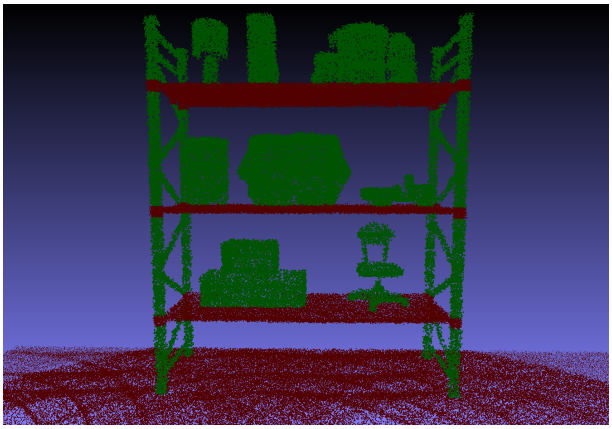


(a) An example of support surface detection on a noisy point cloud. Superb results!

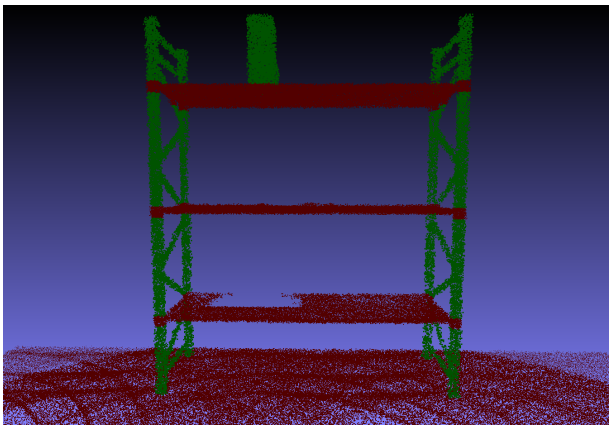


(b) An example of supported object detection on a noisy point cloud. The object is perfectly segmented. The increase in sparseness is due to the sample fraction used for the object segmentation.

Figure 7: 2 figures showing the same scenario described in Figure 5 but with added noise to each point. Each point was moved along each x, y and z axis by some random amount between 0 and 0.05 (units in meters as mentioned before).



(a) An example of support surface detection on a noisy point cloud. Superb results!



(b) An example of supported object detection on a noisy point cloud. The mini shelf is perfectly segmented. All other supported objects were also perfectly segmented.

Figure 8: 2 figures showing the same scenario described in Figure 2 but with added noise to each point. Each point was moved along each x, y and z axis by some random amount between 0 and 0.05 (units in meters as mentioned before).

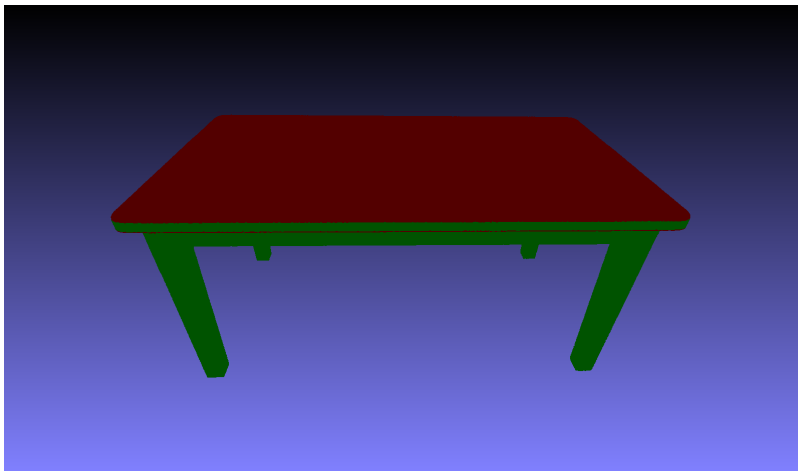
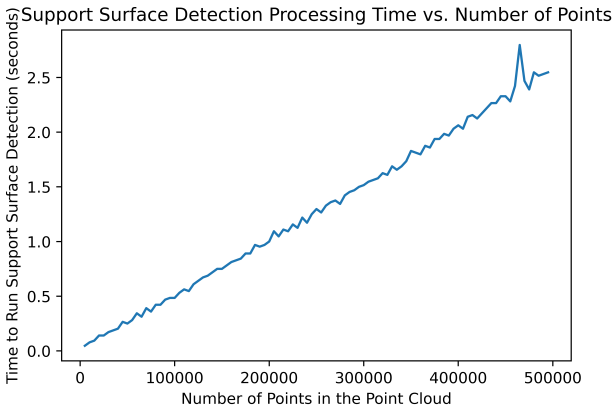
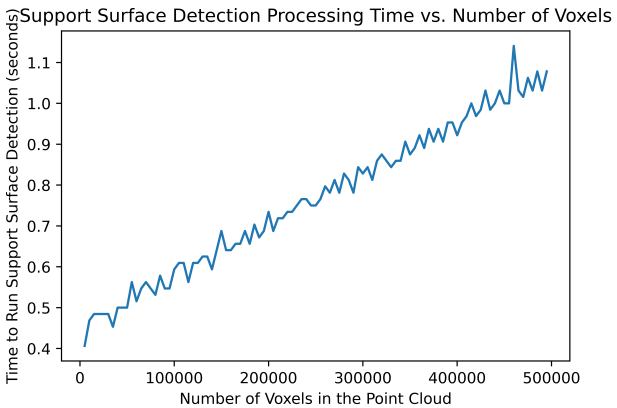


Figure 9: The output of the support surface detection algorithm on a point cloud with $2,000,000$ points, using $2,924,660$ voxels for detection. The point cloud was given by sampling points on the surface of a mesh of a table. The algorithm took only 14.75 seconds to complete. Even point clouds with millions of points organized into millions of voxels can be processed in a matter of seconds. The point cloud is so dense that it looks like a solid object.



(a) In these experiments, the total numbers of voxels used was fixed at $14,974$.



(b) In these experiments, the total numbers of points in the point cloud was fixed at $100,000$.

Figure 10: 2 figures showing the time it takes for the support surface detection to complete. The tests were ran by sampling points from the surface of the same table mesh used in Figure 9. The processing time increases linearly both with the number of voxels used for detection, and the number of points in the point cloud.

5 Documentation

There are 2 main programs to run. The 2 programs have a shared helper file, *utilities.py*. They also both rely on a slightly modified version of the GeomProc library (due to some minor bug fixes). Each program was tested using Python 3.8.8. There is no compilation required, as they are ran exclusively using Python! The following subsections explain the usage for each of the 2 main programs.

5.1 Support Surface Detection

The program to run the support surface detection is called *plane_detection.py* and is located in the base of the Code folder. Just open a terminal and run the file without changing where it is located in the Code folder. The user specified parameters must be changed within the file before running it, and examples of how to use and change the parameters are seen at the end of the file. Example point clouds are included in the same folder. User-specified parameters in the file correspond to the parameters discussed in Section 2 of this written report, and are described in the source code using comments. The program outputs 2 point clouds, one called $'X$ *not detected.obj*$'$ and one called $'X$ *detected.obj*$'$, where X is the user specified string input to the program.

5.2 Supported Object Detection

The program to run the supported object detection is called *nearest_neighbors_segmentation.py* and is located in the base of the Code folder. Just open a terminal and run the file without changing where it is located in the Code folder. The user specified parameters must be changed within the file before running it, and examples of how to use and change the parameters are seen at the end of the file. Example point clouds are included in the same folder. User-specified parameters in the file correspond to the parameters discussed in Section 2 of this written report, and are described in the source code using comments. The program outputs and saves each segmented object as a point cloud in a directory called $'X$ *segmented shapes*$'$, where X is the user specified string input to the program. The point clouds are saved in order of biggest to smallest, and are indexed corresponding to that order.

6 Conclusion

This project validated the initial hypothesis: A voxel-based region growing approach can detect support surfaces efficiently and effectively. The voxel-based region growing approach naturally allows detecting support surfaces of any shape. The implemented method does suffer from having a few user parameters to fine tune. Although, the parameters allow the user freedom to decide on a trade trade off between efficiency and accuracy.

The supported object detection algorithm depends on the support surface detection algorithm being ran first. The support surface detection algorithm has a user-specified parameter for determining whether or not it should have to compute estimated normals for each point to filter out points which do not have vertically oriented estimated normals. The normal estimation and filtering does not have to occur to effectively segment and detect each supported object. This may be acceptable if the application of the algorithm is to segment and detect supported objects. However, the support surface detection algorithm suffers from false positives when normals are not estimated for each point. For an application in robotics, where a robot is using a support surface detection algorithm to determine where it can land/stand/move on, false positives can be

disastrous. The robot may try to operate on an object which is not planar, resulting in it wiping out and damaging itself. Therefore, the normal estimation is important in certain situations.

6.1 Future Improvements

6.1.1 Using Parallel Processing to Improve Computation Speed

The support surface detection detects planes in each layer of the point cloud one at a time. This could easily be improved using parallel computing, i.e. multiple threads running at the same time, each one detecting the horizontal planes at a different height in the point cloud.

6.1.2 Improving Filtering

Due to time constraints, the normal estimation and filtering logic was not optimally implemented. When enabled, normals for every single point are estimated prior to support surface detection. This is not necessary, first the supported surfaces could be detected without normal estimation and filtering. Then, the points classified as a part of a support surface could have their normals estimated. This would greatly improve efficiency, and potentially also improve the quality of the normal estimation. Then, the support surface detection would be ran again but this time with filtering via normal estimation, and only on the points detected from the previous support surface detection. When normal filtering was enabled, the vast majority of computational effort was spent on the normal estimation. Reducing the amount of normals to estimate would greatly increase computational efficiency.

6.1.3 Improving Supported Object Segmentation

The object segmentation was not the focus of this work. Developing another voxel-based approach for the object segmentation approach could drastically reduce the processing time required. The processing time required for the object segmentation was fairly high.

6.1.4 Reducing Incorrect Classification with Support Surface Detection

There were some instances of false positives and false negatives in the results shown. In particular, the uneven surface of one of the tables was problematic. Further developments to better handle cases like these would make the support surface detection incredibly accurate. Some ideas may be to: Run the support surface detection algorithm multiple times, with multiple different settings and aggregate the results somehow, integrate machine learning into the method or have some sort of additional refinement step after planes have been detected (something RANSAC inspired, perhaps, or involving voxel-based region growing in the vertical direction).

6.2 Future Applications

6.2.1 Prepocessing for, or Combining with, Machine Learning

This algorithm may be used in parallel with a machine learning approach. There exist machine learning approaches for solving point cloud classification, the most popular one probably being Pointnet [10]. Many machine learning approaches benefit from some sort of classical algorithm that preprocesses the input. One approach in particular that would likely work well alongside the voxel-based region growing technique is Voxelnet [17].

6.2.2 Annotating Data for Machine Learning Approaches

Annotating data for supervised learning can be very laborious. The methods explained in this written report could greatly increase the speed at which data can be annotated, by providing an 'initial guess' that is later refined (if needed) by a user through some sort of interface.

References

[1] Matthew Berger, Andrea Tagliasacchi, Lee Seversky, Pierre Alliez, Joshua Levine, Andrei Sharf, and Claudio Silva. State of the art in surface reconstruction from point clouds. In *Eurographics 2014-State of the Art Reports*, volume 1, pages 161–185, 2014.

[2] Carlos Campos, Richard Elvira, Juan J Gómez Rodríguez, José MM Montiel, and Juan D Tardós. Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, 2021.

[3] Davide Antonio Cucci, Martin Rehak, and Jan Skaloud. Bundle adjustment with raw inertial observations in uav applications. *ISPRS Journal of Photogrammetry and Remote Sensing*, 130:1–12, 2017.

[4] Zhipeng Dong, Yi Gao, Jinfeng Zhang, Yunhui Yan, Xin Wang, and Fei Chen. Hope: Horizontal plane extractor for cluttered 3d scenes. *Sensors*, 18(10):3214, 2018.

[5] Anna Eilering, Victor Yap, Jeff Johnson, and Kris Hauser. Identifying support surfaces of climbable structures from 3d point clouds. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6226–6231. IEEE, 2014.

[6] Cedrique Fotsing, Nareph Menadjou, and Christophe Bobda. Iterative closest point for accurate plane detection in unorganized point clouds. *Automation in Construction*, 125:103610, 2021.

[7] Ruiqi Guo and Derek Hoiem. Support surface prediction in indoor scenes. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2144–2151, 2013.

[8] Thien-Minh Nguyen, Muqing Cao, Shenghai Yuan, Yang Lyu, Thien Hoang Nguyen, and Lihua Xie. Viral-fusion: A visual-inertial-ranging-lidar sensor fusion approach. *IEEE Transactions on Robotics*, 2021.

[9] Polycam. How to lidar scan – crazy iphone 12 pro trick. `https://www.youtube.com/watch?v=7yXDY25C0hI`, November 2021.

[10] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[11] Microsoft Research. Airsim. `https://microsoft.github.io/AirSim/`, November 2021.

[12] Alessandro Rizzo, Joshua A Haustein, and Carolina Bianchi. *Extracting contact surfaces from point-cloud data for autonomous placing of rigid objects*. PhD thesis, Thesis, 2020. URL: http://webthesis. biblio. polito. it/id/eprint/14410, 2020.

[13] Radu Bogdan Rusu, Nico Blodow, Zoltan Csaba Marton, and Michael Beetz. Close-range scene segmentation and reconstruction of 3d point cloud maps for mobile manipulation in domestic environments. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1–6. IEEE, 2009.

[14] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *European conference on computer vision*, pages 746–760. Springer, 2012.

[15] Yusheng Xu, Wei Yao, Ludwig Hoegner, and Uwe Stilla. Segmentation of building roofs from airborne lidar point clouds using robust voxel-based region growing. *Remote Sensing Letters*, 8(11):1062–1071, 2017.

[16] Cha Zhang and Tsuhan Chen. Efficient feature extraction for 2d/3d objects in mesh representation. In *Proceedings 2001 International Conference on Image Processing (Cat. No. 01CH37205)*, volume 3, pages 935–938. IEEE, 2001.

[17] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4490–4499, 2018.