Orthogonal Structure Detection in Point Clouds

Frankie Eder
EE290T: Advanced Topics in Signal Processing
Instructor - Avideh Zakhor

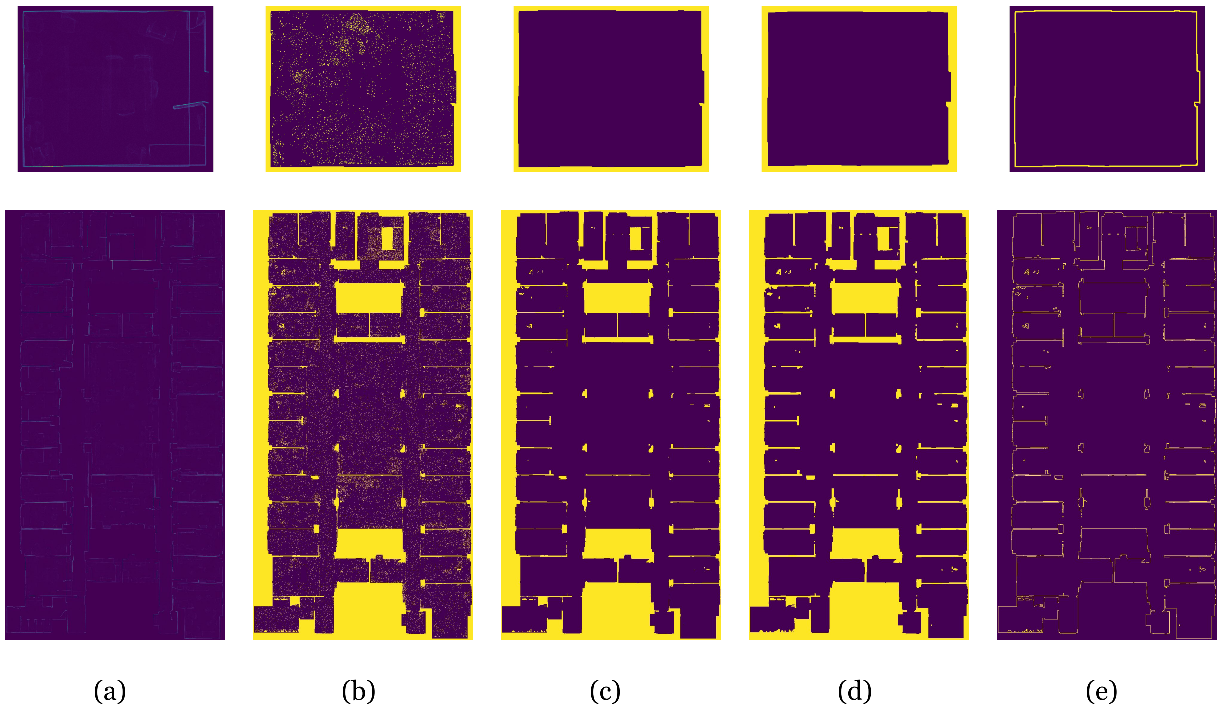

Figure 1. Wall detection using void space assumptions. We start with the 2D histogram-based density map of a point cloud projected to the xy-plane (a), detect void space (b), remove noise (c), dilate (d), and subtract to isolate walls (e). This process is visualized separately for an individual room (top), and for an entire building (bottom), and is described in detail in Section 2.

Abstract

The Manhattan assumption, which assumes rooms and buildings are rectangular, is common in 3D construction, detection, and reconstruction literature. However, this assumption inherently limits the types of scenes we can understand, excluding more creative architecture from modern advances in robotics, reconstruction, and augmented reality. To explore basic non-manhattan scene understanding, we propose a wall-detection method that instead exploits key void space assumptions, operating directly on an input point cloud with no training involved.

1) Introduction and Previous Work

Our work is heavily inspired in both motivation and methodology by the large scale semantic parsing methods and dataset proposed in *3D Semantic Parsing of Large-Scale Indoor Spaces* by Iro Armeni et al.[1] Our primary motivation is to extend the original results and goals of the aforementioned paper to be invariant to the manhattan assumption. In an effort to function as a portion of an existing pipeline, our implementation simply takes in an input point cloud and outputs the subset of points that are detected as walls. We had hoped to test our improvement on the remainder of the original implementation[1], in hopes that it would improve their results and conclusions. However, the authors had removed the publicly available code mentioned in their paper, and were unresponsive upon further inquiry. To compare accurately would require a ground-up implementation of their approach. For the purposes of this class, we have left a more in-depth comparison against Armeni et al.'s original approach for future work.

Luckily, wall detection is a general problem with applications beyond just the room segmentation proposed in the original paper; it is the first step to most scene understanding problems. Our approach works on a variety of input spaces, successfully testing against both individual rooms and large areas as proposed in the original paper (see Section 4). Furthermore, one could detect other room bounding structures such as floors and walls with straightforward modifications to this proposed wall detection methodology. However, this requires further fine tuning and assumptions left for future work (see Section 5), and would benefit from a fine-tuned room segmentation implementation such as the Voronoi method proposed in Bormann et al.[2] to differentiate separate floors in the input point cloud.

Our methodology exploits the following **key assumptions**:

1) In the input point cloud, adjacent to room bounding structures, there is a section of zero density void space where no data has been collected at any point in the gravity direction.
2) Walls are parallel to the gravity direction.
3) The gravity direction is denoted in the input point cloud along the standard axis z.

To exploit these assumptions, we propose the following **key ideas**:

1) Replacing the 1D histogram wall detection method proposed in Armeni et al.[1] (which is dependent on the manhattan assumption) in favor of a 2D histogram (which by nature preserves more locality information) allows us to see non-manhattan wall geometry in the xy-plane.
2) Operating under the above assumptions, walls lie adjacent to low-density volumes in the point cloud, which can be directly isolated from the pixels of the density map created by the 2D histogram.

2) Methodology and Implementation

We implement in Python to utilize third-party packages helpful for visualizing and manipulating 3D data. The methodology itself is divided into five simple parts: density map creation, void space isolation, noise-reducing morphology, dilation, and remapping. We have left discussion of a more efficient C++/Halide implementation for future work (see Section 5).

2.1) Density Map Creation

To create the **density map**, we drop the z axis and input the remaining x, y coordinate pairs into a standard 2D histogram method, such as that included in NumPy. We chose the bin size of the 2D histogram such that the output histogram has an equivalent pixel width of ½” x ½”. This allows space between walls in a building-scale point cloud to map onto a range between 3-8 pixels (1½” - 4”) and the thickness of the walls themselves to map onto a range between 1-3 pixels (½” - 1½”). We base this choice from standard wall thicknesses and gaps present in most modern architecture. This also allows ample pixel width between walls that remain after undergoing the noise reducing morphology (see section 2.3) while avoiding the additional computational complexity of a finer pixel mapping. This wall, void space, wall pattern can then be isolated by directly finding the void space of the density map. To thoroughly validate this pixel size parameter would require further testing on a more extensive dataset with a wider variety and extent of room structures (see section 5).

2.2) Void Space Isolation

We can directly isolate the void space from the 2D histogram by creating a new, related image that we will call the **noisy void space image**. This void space image is an exact copy of the density map, but with all nonzero values mapped to a value of zero (False), and zero values mapped to a value of one (True), and padded by a 10-pixel thick border of true values to pad the subsequent morphological operations. However, this yields many “false” void values in the assumed floor and ceiling space of the pixel mapping. Luckily, in practice, these false points do not form large clusters, and can be removed easily using classical 2D image processing approaches.

2.3) Noise Reducing Morphology

To remove the false void values, we utilize the morphological “opening” operation, which consists of a sliding-window kernelized erosion operation followed by an inverting dilation operation. We chose the standard implementation from OpenCV[3], with a square 3x3 input kernel and one iteration. This removes the small, sparse clusters in the floor area without removing too much of the desired void space that lies behind walls. The results of these are qualitatively evident in Figure 1. We will call the “opened” noisy void space image the **clean void space image.**

2.4) Dilation

After we have removed the false void space values, we can simply dilate the remaining space to encapsulate the walls, as shown in Figure 2. Again, we chose the standard OpenCV implementation with a square 3x3 kernel. To dilate sufficiently to encapsulate the entire wall thickness, we run three iterations, expanding to encapsulate all corresponding points within 1.5” of the clean void space. Subtracting the clean void space image leaves us with the **wall image**. We conclude that all (x, y, z) points from the input point cloud that mapped to the wall image are walls, and only must invert the histogram operation to create our desired output.

2.5) Point correspondence mapping

The goal of this stage is to find all points in the original point cloud that lie within the “true” subset of the wall image. For each (x, y, z) point in the point cloud, we can find its corresponding pixel location in the wall image. If it's location in the wall image is 1, that point has been detected as a wall and we add it to the new set of points that is our desired output. This is conceptually quite straightforward, and we chose to implement it using a SQL-style merge using the Pandas package in Python.

3) Non-Manhattan Modification and Experiments

To test robustness to non-manhattan, curved wall geometry in the point cloud, we propose a point cloud **twisting transformation**. The transformation is defined in Algorithm 1. The transformation takes in a point cloud and a parameter s, and outputs a "twisted" version of the input point cloud. Figure 2, which is a visual representation of the density map after the twisting transformation at different values of s, gives a more intuitive sense of the transformation. In the Section 4, we quantitatively test our methodology on point clouds that have undergone the twisting transformation at $s = 0$ (unmodified), $s = \frac{\pi}{2}$, and $s = 2\pi$.

```
Define a function twist(point cloud, $s$):
    1. Find the centroid of the point cloud, and subtract the centroid from all points in the point
       cloud, elementwise, so that the centroid of the transformed point cloud lies at the origin.
    2. Convert the $(x, y)$ components of the point cloud into polar coordinates $(r, \theta)$.
    3. Find the maximum $r$ in the polar space, $\widehat{r}$.
    4. Define a new angle $\theta' = \theta + s\,(r / \widehat{r})$ for each $(r, \theta)$ in the polar space.
    5. Convert each new polar point $(r, \theta')$ back into cartesian coordinates $(x', y')$
    6. Return the resulting point cloud of points of the form $(x', y', z)$.
```

Algorithm 1. Twisting transformation definition as pseudocode.

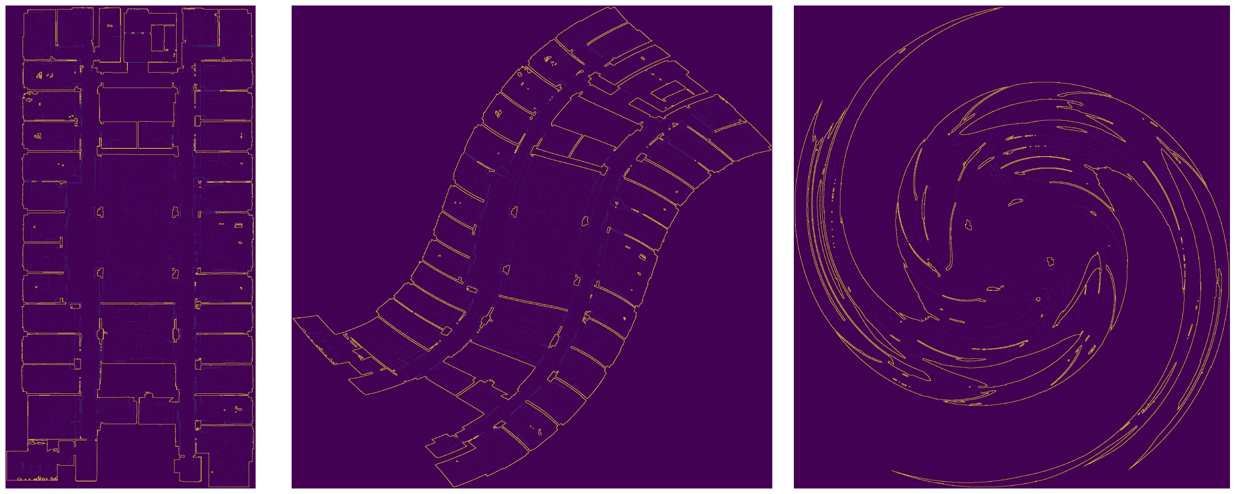

Figure 2. Detected walls overlayed onto density map of transformed point clouds at $s = 0$ (a), $s = \frac{\pi}{2}$ (b), $s = 2\pi$ (c). This gives an intuitive sense for the twisting transformation and its results in wall detection.

4) Results

To test quality, performance, and robustness, we test against the S3DIS dataset from Armeni et al.[1] on an area-wise and room-wise basis, with twisting transformations at $s = 0$, $s = \frac{\pi}{2}$, and $s = 2\pi$, using mean IOU (mIOU) as a metric. Note that a twisting transformation of $s = 0$ corresponds to the unmodified input point cloud. The results of these analyses are shown in Table 1 and Table 2. We test over the 6 areas included in S3DIS, with 44 rooms, 40 rooms, 23 rooms, 49 rooms, 68 rooms, 48 rooms respectively.

s	mIOU	Mean computation time
0	71.67	**1.28**
$\frac{\pi}{2}$	**74.25**	1.37
2π	66.94	1.36

Table 1. Mean IOU and computation time computed over all individual rooms from all areas in the S3DIS[1] dataset ($s = 0$). We repeat our analysis with our twisting transformation applied to each individual room with $s = \frac{\pi}{2}$ and $s = 2\pi$. Rooms are defined by the room segmentation annotations included in the dataset.

s	mIOU	Mean computation time
0	69.79	**112.47**
$\frac{\pi}{2}$	**70.62**	131.20
2π	56.93	132.45

Table 2. Mean IOU and computation time computed over individual areas in the S3DIS[1] dataset ($s = 0$). We repeat our analysis with our twisting transformation applied to each individual area with $s = \frac{\pi}{2}$ and $s = 2\pi$.

These results show promise in regards to manhattan invariance. Specifically, we can note improvement in the mIOU between $s = 0$ and $s = \frac{\pi}{2}$ in both the room-wise and area-wise calculations. This improvement, while surprising, is likely due to the increased exterior distance adjacent to void space in the density mapping. We can also observe that the extreme twisting created with $s = 2\pi$ shows reasonable performance under those conditions, decreasing mIOU by less than 5% in the room-wise analysis.

The mean computation time of a room-scale point cloud is approximately 1⅓ seconds. This is reasonable performance, but we can see that the Python implementation suffers from the additional $\Theta(log(n))$ complexity of the SQL-style merging operation. Each area is approximately 45 rooms, but we see a $\Omega(n)$ increase in computation time when processing the entire area at once over the individual rooms, higher than the linear increase we would expect. See Section 5 for additional discussion of computation time.

We can see that these results are somewhat comparable to the 1D histogram method of Armeni et al.[1], the detection in the wall category within 1.5% mIOU of our results. However, it is difficult to draw a rigorous conclusion from this because our class distinctions are different. We detect walls, windows, columns, and boards as walls since they all are primarily parallel to the gravity direction. A more rigorous comparison would require a ground-up implementation of their approach.

Most of the room results with an mIOU less than 50% were hallways. We can see that this subsequently brings down the area-wise mIOU. We recommend further investigations on more varied datasets.

5) Limitations and Future Work

A C++ implementation would primarily avoid the costly merging step in section 2.3). Although we chose to implement in Python for conceptual clarity, we discovered it was most efficient to make use of third party packages like Pandas and NumPy that are built off of the efficiency of C. In some ways, this did make the methodology somewhat more convoluted, as it necessitates the final remapping step. However, in a C++ implementation, we could avoid the re-merging step by creating a hashmap that stores the original rows of the point cloud as an additional attribute of the density map, so that our desired output already lies in the attributes of the wall image of Section 2.4. This should reduce the overall complexity of this methodology to be proportional to the spatial extent of the input point cloud, avoiding the additional $\Theta log(n)$ computational cost introduced by the SQL-style merge. Since all of these operations are computationally quite straightforward, a fresh C++/Halide implementation would lend itself to further optimization through multi-threading, SIMD operations, and memory-spatial localization.

I also propose that this methodology could easily extend to detect floors and ceilings in point clouds. To do so, we could project into the xz-plane or yz-plane and replace the square dilation kernel with one that only extends in the upwards direction (to detect floors) or downward direction (to detect ceilings). However, this would require additional fine tuning and well-defined assumptions about our input point cloud. Note that while a similar kernelized dilation approach is possible in three dimensional space, we can approximate it much more efficiently with analyses of 2-D cross sections of the point cloud in the xy, yz, and xz-planes.

Since our wall image is a predicted image of the floorplan, this methodology lends itself directly to more robust room segmentation methods such as those surveyed in *Room segmentation: Survey, implementation, and analysis* by Bormann et al.[2] To detect walls in separate floors of an input building's point cloud representation, a room segmentation approach could be applied to the floor plan generated by using our approach, but projecting into the xz-plane or yz-plane instead of the xy-plane. One could then detect walls on a floor by floor basis using our proposed approach in the xy-plane as studied in this paper.

Because the twisting transformation might have an effect on the overall point-density distribution, we believe these results should be verified on a wider variety of datasets, preferable those collected from buildings that already have non-manhattan geometry. Both Armeni et al.'s more recent 2D3DS dataset and the Matterport3D[3] dataset may have enough non-manhattan geometry to reach more rigorous conclusions. With a more expansive dataset, we could also perform a more rigorous ablation study of some of the parameters discussed in Section 2 such as pixel size and dilation kernels, and determine if our parameters need to be better tuned for hallways.

With this in mind, it might still be worthwhile to compare with Armeni et al.'s original end-to-end framework[1], to see if this 2D histogram approach further improves their results and conclusions.

6) Conclusion

In this paper, we provide a promising approach to wall detection in point cloud representations of large scale 3D spaces. We demonstrate an improvement in mIOU with modified, non-manhattan versions of the S3DIS[1] point clouds over their unmodified counterparts on both a room-scale and area-scale basis. These results show promising possibilities for future work that may aid room segmentation and the detection of other room bounding structures such as floors in ceilings in a fashion that avoids the $\Theta(n^3)$ complexity of kernelizing in three dimensions.

References

[1] Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. *3D Semantic Parsing of Large-Scale Indoor Spaces (2016).*
- http://www-inst.eecs.berkeley.edu/~ee290t/fa18/readings/Armeni_3D_Semantic_Parsing_CVPR_2016_paper.pdf

[2] Richard Bormann, Florian Jordan, Wenzhe Li, Joshua Hampp, and Martin Hägele. *Room segmentation: Survey, implementation, and analysis (2016).*
- https://ieeexplore.ieee.org/document/7487234/

[3] OpenCV 3.0.0 Documentation. *Morphological Transformations.*
- https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html

[4] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Nießner, Manolis Savva, Shuran Song, Andy Zeng, Yinda Zhang*Matterport3D: Learning from RGB-D Data in Indoor Environments (2017).*
- http://inst.eecs.berkeley.edu/~ee290t/fa18/readings/indoor-scene-matterport-funkhauser-2017.pdf