

3D Building Detection and Reconstruction from Aerial Drone Imagery

*Marc WuDunn
Avideh Zakhor, Ed.*



Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-230

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-230.html>

December 1, 2021

Copyright © 2021, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

3D Building Detection and Reconstruction from Aerial Drone Imagery

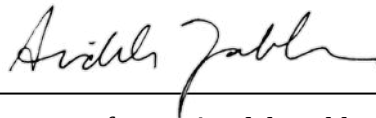
by Marc WuDunn

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences,
University of California at Berkeley, in partial satisfaction of the requirements for the
degree of **Master of Science, Plan II.**

Approval for the Report and Comprehensive Examination:

Committee:

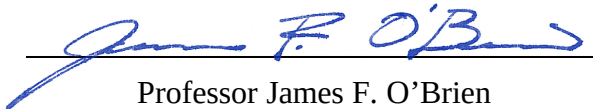


Professor Avidah Zakhor
Research Advisor

8/15/2020

(Date)

* * * * *



Professor James F. O'Brien
Second Reader

August 14, 2020

(Date)

3D Building Detection and Reconstruction from Aerial Drone Imagery

Marc WuDunn

August, 2020

Abstract

In recent years, the ubiquity of drones equipped with RGB cameras has made aerial 3D point cloud and model generation significantly more cost effective than with traditional aerial LiDAR-based methods. Most existing aerial 3D point cloud building reconstruction and segmentation approaches use geometric methods that are tailored to 3D LiDAR data. However, point clouds generated from drone imagery generally have a much different structure than the ones constructed through LiDAR imaging systems, for which these methods are not as suitable. In this thesis we present two methods: (a) an approach for segmenting building and vegetation points in a 3D point cloud; (b) a pipeline for extracting a building footprint with height information from aerial images. For both approaches, we leverage the commercial software Pix4D to construct a 3D point cloud from RGB drone imagery.

To segment the point cloud, our basic approach is to directly apply deep learning segmentation methods to the very RGB images used to create the point cloud itself, followed by back-projecting the pixel class in segmented images onto the 3D points. This is a particularly attractive solution, since deep learning methods for image segmentation are more mature and advanced as compared to 3D point cloud segmentation. Furthermore, GPU engines for 2D image convolutions are likely to result in higher processing speeds than could be achieved using 3D point cloud data. We compute F1 and Jaccard similarity coefficient scores for the building and vegetation point classifications to show that our methodology outperforms existing methods such as PointNet++ and commercially available packages such as Pix4D.

For building footprint extraction, the 3D point cloud is used in conjunction with image processing and geometric methods to extract a building footprint. The footprint is then extruded vertically based on the heights of the extracted rooftops. The footprint extraction involves two main steps, line segment detection and polygonization of the lines. To detect line segments, we project the point cloud onto a regular grid, detect preliminary lines using the Hough transform, refine them via RANSAC, and convert them into line segments by checking the density of the points surrounding the line. In the polygonization step, we convert detected line segments into polygons by constructing and completing partial polygons, and then filter them by checking for support in the point cloud. The polygons are then merged based on their respective height profiles to create a full building footprint, which can be used to construct a 3D model of the building. Notably, an application of the extracted 3D building model is the computation of the window-to-wall ratio of the building. Given a set of detected windows on a 2D image, we can project them onto the extracted 3D model and compute their area to obtain the window-to-wall ratio. We have tested our system on two buildings of several thousand square feet in Alameda, CA, and obtained an F1 score of 0.93 and 0.95 respectively as compared to the ground truth.

Contents

1	Introduction	5
1.1	Related Work on Point Cloud Segmentation	6
1.2	Related Work on Building Footprint Extraction	6
1.3	Contribution of this Thesis	7
2	Vegetation and Building Segmentation in Point Clouds	8
2.1	Methodology	8
2.2	Segmentation of Undistorted RGB Images	9
2.3	Projection of 3D Points to Classified 2D Pixels	10
2.4	Occlusion Detection	10
2.5	Efficient Occlusion Detection	13
3	Building Footprint Extraction	14
3.1	Line Detection	14
3.2	Line Segmentation	16
3.3	Line Processing	16
3.4	Polygon Extraction	18
3.5	Partial-Polygon Construction	19
3.6	Polygon Completion	19
3.7	Polygon Merging	21
3.8	3D Building Reconstruction	21
4	Experimental Results	23
4.1	Data Collection and Photogrammetry	23
4.2	Building and Vegetation Segmentation Results	24
4.3	Building Footprint Evaluation	26
4.4	Window-to-Wall Ratio Estimation	26
4.5	Thermal Image Overlay	29
5	Conclusions and Future Work	31
A	Appendix	35
A.1	RGB Point Cloud Generation	35
A.2	Merged RGB and Thermal Image Projects	35
A.3	3D Model with Thermal Textures	38

List of Figures

2.1	Block diagram of proposed pipeline.	9
2.2	2D segmentation for a single image of Dataset 1.	10
2.3	(a) Segmented point cloud with occlusion artifacts present. (b) Segmented point cloud with occlusion artifacts removed.	11
2.4	Three points P_1 , P_2 , and P_3 that project to a pixel (i_P, j_P) . Point P_1 is the 'hit' point, point P_3 is 'occluded' while the other point P_2 is within the allowable range of P_1 and so is 'unoccluded'.	11
2.5	(a) Quantization example for $\hat{d}(1)$ and $\hat{d}(2)$. (b) Block-Reduce example with $r_e(1)$	13
3.1	(a) A circular flight path in DJI GS Pro. (b) The full flight plan.	15
3.2	RGB Drone imagery for (a) Building 1. (b) Building 2. (c) Grid with points accumulated by grid cell for building 1. (d) Lines detected in the first iteration.	15
3.3	(a) Example of the 'gridded' line. (b) The line segment in (a) thresholded by density. (c) One of the detected lines, trimmed to match the support in the grid. (d) The extracted line segments.	17
3.4	(a) Merging endpoints close together. (b) Splitting a nearby line segment. (c) Detected lines after post-processing.	18
3.5	(a) Example of a 'partial' polygon in Building 1. (b) Counter-clockwise traversal.	18
3.6	(a) Completion process for a partial polygon with 3 corners. (b) Propagation effect of completing one partial polygon to the completion of another in a subsequent iteration. (c) Completion process for a partial polygon with 1 corner.	20
3.7	Example of the completed (unmerged) polygons after all iterations have completed.	21
4.1	(a) The example point clouds. Classification results from: (b) our method; (c) Pix4D; (d) PointNet++; (e) The ground truth. Green: Vegetation, Red: Building, White: Other	25
4.2	(a) Histogram of number of unique votes per point. (b) Histogram of winning vote's confidence per point.	26
4.3	(a) Polygons for Building 1 after the merge process has completed. Heatmap of the heights of merged polygons for (b) Building 1. (c) Building 2. 3D Rendering of Alameda Buildings (d) 1 and (e) 2, from the extracted 2D polygon footprints, with polygons extruded by height.	27
4.4	Flowchart for the window-to-wall ratio extraction process.	28
4.5	(a) 3D building model projected onto a 2D image, with roof polygons overlaid in blue and facades in red; (b) The corresponding window mask; (c) Facade (red) with its corresponding windows (blue).	29
4.6	Triangulation of the projected facade points with the facade in red and window in blue. . . .	29

4.7	3D building model overlaid on top of several thermal (IR) images, with the roof facade in blue and wall facades in red.	30
A.1	(a) Settings for the 'Initial Processing' step. (b) Settings for the 'Point Cloud and Mesh' step.	36
A.2	Point cloud generated from RGB images in Pix4d.	36
A.3	(a) Manual Tie Points (MTPs) for the RGB project shown as green arrows. (b) The merged point cloud with RGB (outer circle) and Thermal (inner circle) drone flights overlaid.	37
A.4	Settings used to generate the textured 3D mesh.	37
A.5	3D Mesh textured with thermal imagery (a) Top view. (b) Side view with windows.	38

Chapter 1

Introduction

In recent years, the ubiquity of drones equipped with RGB cameras has made aerial 3D point cloud and model generation significantly more cost effective than with traditional aerial LiDAR-based methods. Most existing aerial 3D point cloud building reconstruction and segmentation approaches use geometric methods that are tailored to 3D LiDAR data. However, point clouds generated from drone imagery generally have a much different structure than the ones constructed through LiDAR imaging systems, for which these methods are not as suitable. This thesis explores two methods that aid in analysis of envelope thermal efficiency of buildings. In particular we develop methods for 3D building reconstruction and 3D point cloud segmentation, which are important steps for thermal anomaly detection and extracting the window-to-wall ratio of buildings. Both can be useful in a wide variety of application areas, including urban planning, energy modeling, and disaster planning. These methods are also interrelated: point cloud segmentation can be used to directly improve the results of the footprint generation and vice-versa. The goal of this thesis is twofold: to present a method to segment the building and vegetation points in a point cloud, and to present a pipeline for extracting building footprints from drone imagery, including facade information, the building components, and the height information for each component. For both methods we first construct a point cloud using aerial RGB drone images, processed in Pix4D to create a point cloud. For experimental verification, we collected 2 datasets, each with several hundred images each, of two different buildings in Alameda, California.

The semantic understanding of 3D point clouds provides valuable information to many disciplines that model outdoor environments. After generating the 3D point cloud generated from Pix4D, we use state-of-the-art deep learning frameworks pre-trained on large datasets to segment the images, which we then use to assign labels to each of the points. To do so, we back-project the segmented pixels onto the points in the point cloud. In doing so, we develop geometric techniques to deal with occlusion issues. This pipeline is applied to aerial imagery collected by a commercial UAV of an urban outdoor scene, segmenting objects based on the classes ‘building’ and ‘vegetation’.

Our approach for the building footprint extraction consists of two overarching steps, line segment extraction and polygonization of the lines. In the line segment detection step, the point cloud is projected onto a regular grid, preliminary lines are detected using the Hough transform, subsequently refined with RANSAC, and then converted into line segments by checking the density of the points surrounding the line. In the polygonization step, detected line segments are converted into polygons, by constructing and completing partial polygons, and filtering them by checking for support in the point cloud. The height of each polygon is estimated, and then the polygons are merged based on their respective height profiles. After merging these polygons, we can create a 3D model by extruding the footprint with the estimated heights of each

polygon. From this 3D model, and given the locations of windows on the 2D drone images, we can compute the window-to-wall ratio.

1.1 Related Work on Point Cloud Segmentation

A large body of research has investigated semantic segmentation of 3D point clouds. Early learning-based methods used principle component analysis to segment based on learned spatial distributions of points [1]. However, the associated eigen-decomposition is computationally expensive. Recent state-of-the-art methods utilize deep learning applied to different forms of input data. [2, 3, 4, 5] demonstrated the application of a deep net directly to point clouds for the task of segmentation, while [6, 7] applied such geometric methods to aerial point clouds. However, purely geometric methods struggle with the fine details of outdoor imagery unless extremely dense LiDAR data is available. Collection of such data is often prohibitively expensive and time-consuming. Networks that jointly learn from geometric and color data have been proposed [8, 9, 10], as well as methods that flatten 3D surfaces to apply 2D CNNs [11, 12]. These approaches also assume availability of dense 3D data. Our work most closely resembles [13, 14], in which 2D segmentation methods are combined with photogrammetry reconstruction. However, [13, 14] mesh, texture-map, and create a synthetic view of the scene, which is then segmented and back-projected, thus introducing additional computationally complex steps.

1.2 Related Work on Building Footprint Extraction

Existing 3D reconstruction methods typically involve the use of satellite imagery, terrestrial laser scanning, or top-down Light Detection and Ranging (LiDAR) data [15]. Terrestrial laser scanning and LiDAR require costly equipment [16]. LiDAR suffers from limited range data. Terrestrial laser scanning does not easily scale to large areas. Building footprint extraction from a single satellite image is scalable and could be applied to large areas, but typically does not result in building facade information or accurate height information [17].

Several geometric approaches have been proposed using satellite imagery, [18, 19, 20, 21] but they only detect building outlines, as extracting height information from a single satellite image is difficult. They also tend to perform well for relatively simple building structures, such as those composed of few facades. Methods utilizing convolutional neural networks (CNNs) on satellite images have also been proposed [22]. Additionally, Microsoft [23] released a dataset containing over 100 million building footprints in the United States generated using neural networks. Although CNNs have been reported to work well for satellite data, they are difficult to apply to drone imagery or point cloud data, since they suffer from a lack of labeled training datasets. As a result, generating building footprints which also contain height information is difficult using deep learning methods.

Many 3D reconstruction methods utilize 3D data such as LiDAR or Digital Surface Models (DSMs) in order to construct footprints. Even though LiDAR can produce reasonable results [24, 25], LiDAR sensor sets for UAVs can be prohibitively expensive. Methods that utilize DSMs [26, 27, 28] allow building height information to be extracted, as height is encoded into the data model.

In general, the use of geometric features of buildings, such as their shape and their shadows, are useful for the extraction of building footprints. For instance, Lin and Nevatia [29] propose a method to detect simple, rectangular building roofs from a single oblique image, using edge detectors and 3D evidence present in the image. Their approach involves generating a number of hypotheses for building footprints which are then pruned to create the final one. They additionally estimate building height from the shadows. Shadow information is used in several other approaches [30, 31] for the extraction of footprints or their heights. The pipeline proposed by San et. al. [32] utilizes the Hough transform for the extraction of building edges, but

is limited to simple rectangular structures or circular structures. Likewise, Hammoudi et al. [33] employ the Hough transform for the extraction of edges corresponding to building facades, though they fall short of extracting an actual building footprint. Our method also employs the Hough Transform to generate 2D line segment candidates, which we augment to construct a full building footprint.

1.3 Contribution of this Thesis

This thesis explores two pipelines: (a) an approach for segmenting building and vegetation points in a 3D point cloud; (b) a pipeline for extracting a building footprint with height information from aerial images. For both pipelines, we opted use the Pix4D software suite [34] to generate a dense 3D point cloud and calculate the internal and external parameters for camera calibration from our collection of input images. In order to segment the point cloud we use a multiview-based approach, in which we perform 2D semantic segmentation on the drone images we captured, and then projecting the points onto the images. Pix4D provides a set of camera pose matrices as output based on the internal and external camera parameters calculated for each input image, which we use to project the points onto the camera images. This matrix is used to convert the 3D coordinates of a point in the generated dense cloud to the 2D coordinate of a pixel in an undistorted input image. In order to facilitate the segmentation, we develop a novel approach for efficiently detecting points in the point cloud that are visible from a camera image, and ones that are occluded by other points.

The pipeline for extracting polygonal building footprints with height information from 3D point clouds consists of two overarching steps: line segment extraction and a polygonization step where the extracted line segments are converted into polygons. This is used to construct a 3D model of the building, and can also be used to extract the window-to-wall ratio of a building, given an algorithm that detects the windows on the 2D images.

The outline of this thesis is as follows: In Chapter 2 we detail our approach for the segmentation of 3D point clouds, in Chapter 3 we detail our pipeline for extracting a building footprint from a 3D point cloud, in Chapter 4 we show our results for both the 3D point cloud segmentation and the building footprint extraction, and in Chapter 5 we summarize our main conclusions and areas of future work.

Chapter 2

Vegetation and Building Segmentation in Point Clouds

In this chapter, we will discuss the semantic segmentation of 3D point clouds to segment building and vegetation points, by utilizing state-of-the-art 2D semantic segmentation methods on images. This has applications in areas such as fire prevention, for which the distance between buildings and surrounding vegetation is very important. Section 2.1 provides an overview for our pipeline. Section 2.2 discusses our approach for segmenting the 2D drone images. Section 2.3 describes our approach for generating labels for the 3D points from the 2D drone images. Since at any given view, some points may not be visible due to occlusions, Section 2.4 explains our approach for detecting points that are occluded by other points, from the perspective of the camera image. Finally, Section 2.5 describes an approximation method that significantly improves the performance for detecting point occlusions.

2.1 Methodology

The semantic understanding of 3D point clouds provides valuable information to many disciplines that model outdoor environments. Urban planning, wireless communications engineering, and wildfire abatement are applications aided immensely by information on the structure and classification of environmental objects. The advent of increasingly accessible and accurate methods of surveying environments, combined with powerful computational methods for representing and processing surveyed data, have led to a large body of research into the task of semantic segmentation, producing a variety of approaches to this problem [35]. These approaches differ in complexity, accuracy, and data collection requirements. Two increasingly powerful tools that aid in this task are a) photogrammetry, which creates structure from 2D image collections [36], and b) deep networks, capable of accurately segmenting images that form the basis for 3D structure [37]. Combined, these tools provide a simple yet powerful method of generating and semantically segmenting 3D point clouds.

In this chapter, we present a pipeline for the generation of semantically-segmented 3D point clouds from aerial RGB imagery of outdoor environments. The outline for this pipeline is presented in Figure 2.1. By combining commercially-available photogrammetry software with open-source state-of-the-art deep net frameworks pre-trained on large datasets, our pipeline is simple, accurate, and accessible. The basic idea behind our approach is to apply deep learning methods to segment the very RGB images that are used to create the 3D point cloud, and to back-project the segmented pixels onto the points in the point cloud. In

doing so, we develop geometric techniques to deal with occlusion issues. This pipeline is applied to aerial imagery collected by a commercial UAV of an urban outdoor scene, segmenting objects based on the classes ‘building’ and ‘vegetation’. For applications such as fire prevention, the distance between vegetation and building is an important quantity to estimate.

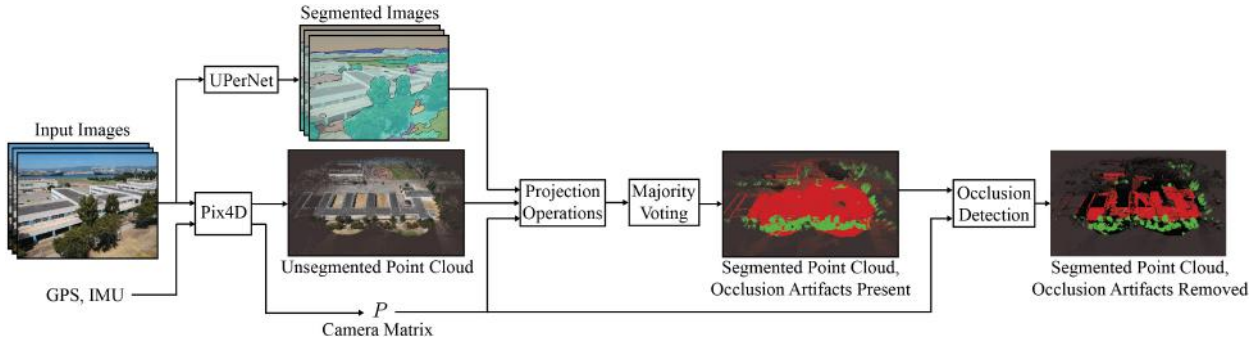


Figure 2.1: Block diagram of proposed pipeline.

A large body of research has investigated semantic segmentation of 3D point clouds. Early learning-based methods used principle component analysis to segment based on learned spatial distributions of points [1]. However, the associated eigendecomposition is computationally expensive. Recent state-of-the-art methods utilize deep learning applied to different forms of input data. [2, 3, 4, 5] demonstrated the application of a deep net directly to point clouds for the task of segmentation, while [6, 7] applied such geometric methods to aerial point clouds. However, purely geometric methods struggle with the fine details of outdoor imagery unless extremely dense LiDAR data is available. Collection of such data is often prohibitively expensive and time-consuming. Networks that jointly learn from geometric and color data have been proposed [8, 9, 10], as well as methods that flatten 3D surfaces to apply 2D CNNs [11, 12]. These approaches also assume availability of dense 3D data. Our work most closely resembles [13, 14], in which 2D segmentation methods are combined with photogrammetry reconstruction. However, [13, 14] mesh, texture-map, and create a synthetic view of the scene, which is then segmented and back-projected, thus introducing additional computationally complex steps.

2.2 Segmentation of Undistorted RGB Images

We segment RGB images with UPerNet [38], a framework based on a 50-layer ResNet [39] convolutional neural net architecture. The UPerNet framework was chosen for its flexibility in semantically segmenting scenes based on object, material, or texture. We use existing models pre-trained by the UPerNet authors on the ADE20K dataset [39]. We pass the undistorted images produced by Pix4D to the segmentation network after downsampling the resolution by a factor of 4. The output of the segmentation network is a 2D array corresponding to the pixel dimensions of the input image, with each element containing a class label for the pixel. The ADE20K output classes fall into multiple categories, which we condense into ‘building’, ‘vegetation’, and ‘other’. An output from the 2D image segmentation network is shown in Figure 2.2.



Figure 2.2: 2D segmentation for a single image of Dataset 1.

2.3 Projection of 3D Points to Classified 2D Pixels

We next project each 3D coordinate in the generated point cloud to the 2D coordinate of all segmented images using the camera pose matrices provided by Pix4D, and retrieve the pixel classes corresponding to the 2D coordinates.

Given the set of segmented 2D images $\mathbf{I} = \{I_1, I_2, \dots, I_m\}$, where each image I_m consists of a list of pixel coordinates $\{(u_1, v_1), (u_2, v_2), \dots, (u_n, v_n)\}$, a list of corresponding pixel classes $\{c_1, c_2, \dots, c_n\}$, and a corresponding camera pose matrix P_m ; and, given a point cloud

$C = \{(x_1, y_1, z_1), (x_2, y_2, z_2), \dots, (x_o, y_o, z_o)\}$, we determine $(u_{m,o}, v_{m,o})$ for each point (x_o, y_o, z_o) in C and each image I_m via that image's camera matrix P_m :

$$(u_{m,o}, v_{m,o}) = P_m \cdot (x_o, y_o, z_o) \quad (2.1)$$

If $(u_{m,o}, v_{m,o})$ is within the pixel coordinate range of the image, it is a valid projection. We then index to find the corresponding pixel class $c_{m,o}$ for that coordinate, and add it to a stored dictionary of pixel classes per 3D point. We take the majority vote of all c_m for each point, and assign this value as the final point class. A fundamental limitation of this projection method is as follows: 3D points which are occluded from the perspective of a 2D image map to the same pixel on that image if the points lie along the same ray of projection. This leads to highly inaccurate point class assignment, as any two points along the ray of projection are assigned to the same class. For example, with aerial imagery, the large amount of ground occluded by a building could erroneously be classified as building.

2.4 Occlusion Detection

When projecting pixel classes in captured images onto the point cloud, multiple 3D points with different distances might be affiliated with a given pixel. The purpose of occlusion detection is to choose the 3D point that is closest to the camera image under consideration. This way, more distant 3D points corresponding to the same pixel will not be assigned a class from that pixel, since they are occluded by closer 3D points. An example of a point cloud segmentation in which occlusion is not taken into account is shown in Figure 2.3(a), where a large number of points are mis-classified.

A standard approach for detecting occlusions is to trace a ray through the scene and check for a 'hit', i.e. an intersection within a 3D bounding shape. However, tracing rays can be quite costly from a computational

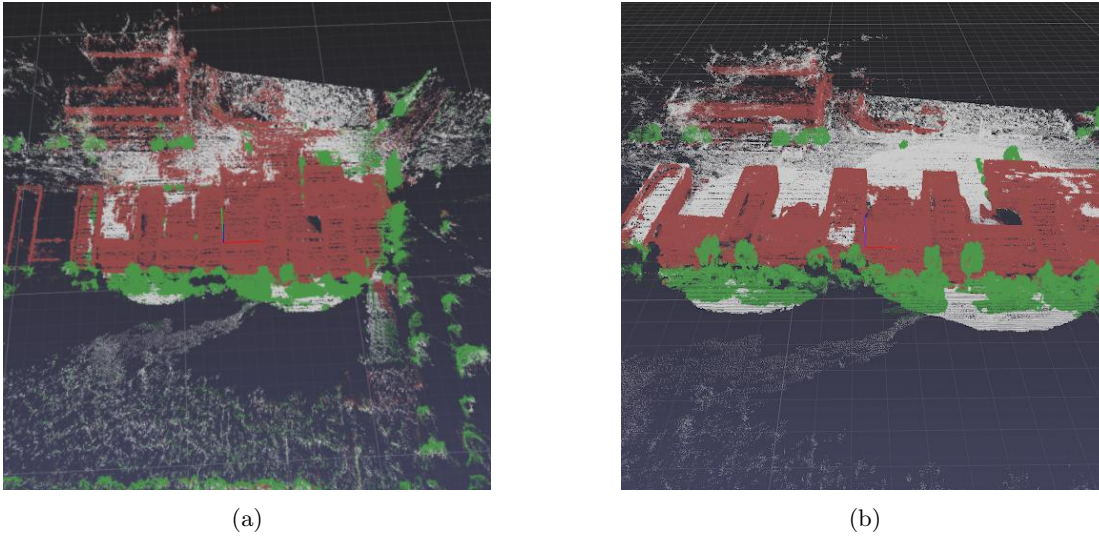


Figure 2.3: (a) Segmented point cloud with occlusion artifacts present. (b) Segmented point cloud with occlusion artifacts removed.

point of view, especially for several hundred images. Rather, we iterate over each captured image, and in each iteration we project all 3D points in the point cloud onto the image and compare projective distances.

In what follows, we describe our approach for determining whether a given 3D point P is visible in a given image. Let the closest point along the camera ray be denoted as P_{hit} . Let d_P denote the projective distance of P with respect to the camera optical center, and (i_P, j_P) denote the pixel upon which P projects

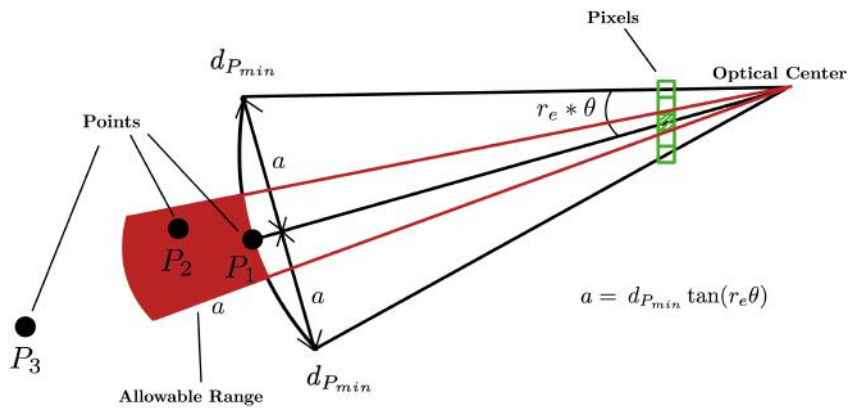


Figure 2.4: Three points P_1 , P_2 , and P_3 that project to a pixel (i_P, j_P) . Point P_1 is the 'hit' point, point P_3 is 'occluded' while the other point P_2 is within the allowable range of P_1 and so is 'unoccluded'.

onto. We project each 3D point in the point cloud using the camera pose matrix, filtering out the points that lie outside of the camera image. We then round the resulting coordinates of the projection to the nearest integer (i, j) values; this corresponds to assigning each point projection to the nearest pixel in the camera image. Next, we compute the minimum projective distance $d_{min}(i, j)$ for all 3D points projected onto the pixel coordinates (i, j) , which will correspond to the projective distance of P_{hit} . This results in a matrix \mathbf{A}_{min} of the minimum projective distances, whose size is the same as the size of the camera image.

The most obvious way to determine whether P is occluded is to directly compare d_P with the associated entry in A_{min} for pixel (i_P, j_P) . However, the extremely high resolution of the camera imagery coupled with the sparsity of the point cloud can result in many pixels having only a small number of 3D points projected onto them. This would render the comparison between d_P and the associated minimum value in A_{min} meaningless, and could result in an inaccurate occlusion detection. To circumvent this problem, we instead compare d_P with the distances of a set of 3D points $S(i_P, j_P)$ that project to the ‘vicinity’ of (i_P, j_P) , where vicinity is defined as a circle of radius r_e . We declare P as ‘unoccluded’ if its projective distance d_P is ‘close’ to the minimum projective distance of all 3D points in set $S(i_P, j_P)$. We denote the 3D point in the set $S(i_P, j_P)$ with such minimum distance as P_{min} , and its associated distance with $d_{P_{min}}$. Roughly speaking, we need to determine whether d_P is close enough to $d_{P_{min}}$, and if it is, we can declare P to be unoccluded. We perform this check for each 3D point in the point cloud whose projection lies in the bounds of the given camera image.

The radius r_e should ideally vary with the projective distance d_P of the point, since at a point further away from the camera image, the physical distance between two adjacent rays passing between neighboring pixels increases. Thus, we propose the following relationship for r_e :

$$r_e \propto \frac{1}{d_p} \quad (2.2)$$

Figure 2.4 provides a visualization of how to determine whether d_p is close enough to $d_{P_{min}}$, where P_1 corresponds to both P_{hit} and P_{min} . The figure shows a side view of the camera pixels and the optical center. The red pixels depict the pixels within radius r_e of pixel (i_P, j_P) . Point P_2 is labeled as ‘unoccluded’ due to its proximity to P_1 , while point P_3 is labeled as ‘occluded’ as it is far away from P_1 . The black spherical cone (SC) in the figure depicts the extrusion of the circle of radius r_e in the image plane outward by the distance $d_{P_{min}}$, with the apex of the SC corresponding to the optical center of the camera. Similarly, the red SC in the figure corresponds to the extrusion of the SC associated with pixel (i_P, j_P) , again with the apex of the SC at the optical center of the camera. If we denote the angle between the rays that pass through successive pixels as θ , then the angles for the red and black SC are θ and $2r_e\theta$ respectively. The quantity a in the figure is the base radius of the black SC, and is given by $a = \tan(r_e\theta)d_{P_{min}}$. We construct the red zone in Figure 2.4 by extruding the red SC beyond the black SC by an amount a , and declare all points that fall into the red zone to be unoccluded. Intuitively, this makes sense since the lateral dimension of the red zone should be approximately proportional to the base radius of the black SC. In a way, as the distance $d_{P_{min}}$ increases, we need to be more ‘lenient’ in declaring P as unoccluded. To summarize, we declare P to be unoccluded if:

$$d_p - d_{P_{min}} > d_{P_{min}} \tan(r_e\theta) \quad (2.3)$$

Then, for all unoccluded points, we take the class label for the corresponding pixel in the segmented image, and add it to the stored dictionary of pixel classes for that 3D point, much like in Section 2.3.

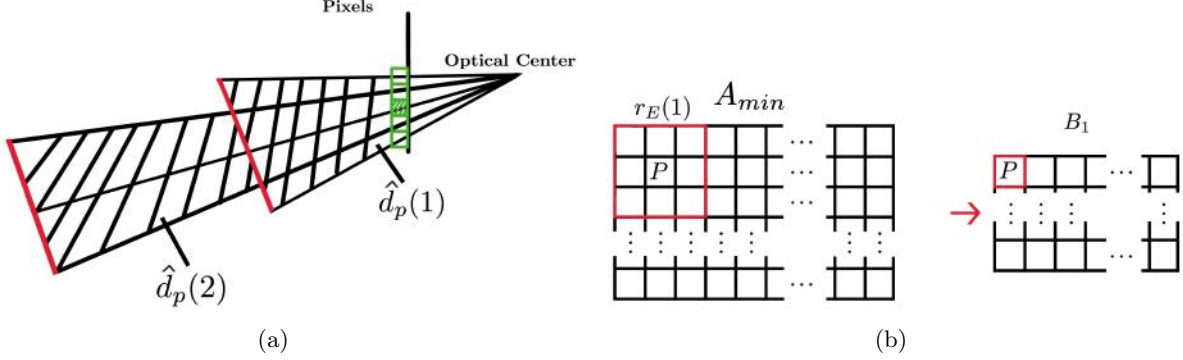


Figure 2.5: (a) Quantization example for $\hat{d}(1)$ and $\hat{d}(2)$. (b) Block-Reduce example with $r_e(1)$.

2.5 Efficient Occlusion Detection

Though the qualitative results are acceptable for the aforementioned method, the running time is fairly high. Specifically, on an Intel Core i5-8600K CPU and for a point cloud with 6 million points, it requires several minutes per camera image. Though this method can be done in parallel, we opt to improve efficiency by implementing an approximation to the above method instead.

Rather than computing r_e as a function of d_P for each point P , we first quantize the projective distances d into N discrete sections $\hat{d}(1), \dots, \hat{d}(N)$ throughout the range of the projective distances in \mathbf{A}_{min} . Then, each of the N corresponding $r_e(i)$ values can be computed as well, using the relationship of Equation 2.2 for each of the $\hat{d}(i)$ values. This is shown in Figure 2.5(a) for distances $\hat{d}(1)$ and $\hat{d}(2)$, with corresponding r_e of 2 and 1 pixel(s) respectively. Using these $r_e(i)$ values, we can perform a block-reduce operation on \mathbf{A}_{min} , for each such $r_e(i)$, to obtain N matrices $\mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_N$. By applying the block-reduce operation, we are assigning each pixel a particular neighborhood for which to compare projective distances, for each $r_e(i)$.

The block-reduce operation, or equivalently *min*-pooling in this case, downsamples the \mathbf{A}_{min} matrix by applying a *min* function to each 'block' of \mathbf{A}_{min} , where the block size b_{size} is a square $1 + (2r_e(i))$ pixels wide. This creates a new matrix B_i with the width and height dimensions reduced by a factor of b_{size} . We opted to use the min function described here in a per-block fashion rather than sliding window similar to the method in Section 2.4. Using the block-reduce operation is an additional approximation that is more computationally efficient, for which we did not notice a significant difference in the correctness of the results as compared to using the sliding window.

An example of block reduce operation from matrix \mathbf{A}_{min} to B_i is shown in Figure 2.5(b) for $r_e(1)$. Next, for each 3D point P , we compute its projective distance d_P , and compute its closest quantized distance $\hat{d}(i)$ and the corresponding $r_r(i)$. We then simply index into the matrix \mathbf{B} , and check whether Equation 2.3 holds for the projective distance at the corresponding coordinate in \mathbf{B}_i . While this may lead to some artifacts as a result of the spatial discretization, in practice we do not find this to be a significant problem. However, by using this approximation we achieve a significant speed-up, with running time on the order of seconds per image rather than several minutes. The resulting segmentation with the occlusion errors removed is shown in 2.3(b).

Chapter 3

Building Footprint Extraction

In this chapter, we describe our pipeline for building footprint extraction. This process involves several steps. In Section 3.1 we perform a line detection step, where we project points along the Z-axis onto a 2D grid and count the number of points that fall into each grid cell. We then use the Hough Transform to extract candidate lines, which mainly correspond to facades in the image, as these will have a high number of points along the Z-axis. In Section 3.2 we convert these lines into line segments, by using a 1D median filter along the line and checking for support from the underlying points. In Section 3.3 we post-process the line segments. This is done by making the line segments intersect if they are nearly intersecting and moving endpoints to the same location if they are near each other. We then construct the edges and assign neighbors. Section 3.4 provides an overview for the polygonization, and explains our approach for converting the line segments into edges. Sections 3.5 and 3.6 then proceed in an iterative fashion. We extract 'partial' polygons in Section 3.5, by traversing along the edges. Then, in Section 3.6 we create full polygons by 'completing' the partial polygons, which creates new edges and hence new partial polygons. In Section 3.7 we merge the resulting polygonal footprint based on the respective heights of the polygons. Finally, in Section 3.8 we estimate the heights of the merged polygons and construct a 3D building model.

Each of these circular flights shown in Figure 3.1 takes on the order of 40 minutes, including 10 minutes to recover the drone and to replace the batteries. These images are then input to Pix4D in order to produce a 3D point cloud. Figures 3.2(a) and 3.2(b) show the point clouds of two buildings we will be processing throughout this chapter.

3.1 Line Detection

After constructing the point cloud, we detect lines in the point cloud, using a process similar to Hammoudi et al.[33] of accumulating points in grid cells and applying the Hough Transform to detect lines. The points are first projected onto a 2D grid, with a resolution of $0.1m$. The number of points in each cell is counted and then the value is normalized to be between 0 and 1. Projection along the z or height axis results in cells containing portions of a facade to have a much higher point count than those without them. We apply a threshold τ to the cells of this grid, so that cells not corresponding to a wall, i.e. those with a low point count are set to 0. For the first iteration, τ is set to a high, fixed value - for example, 5% of the normalized maximum number of points in any cell. After the first line segment is extracted, the average point density of the extracted line segment or facade can be used to update τ . This ensures that only cells with similar

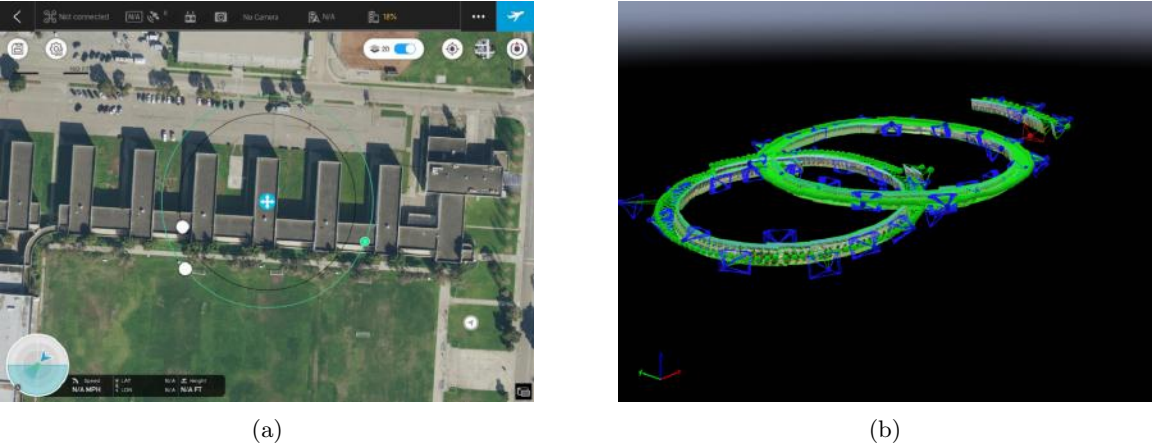


Figure 3.1: (a) A circular flight path in DJI GS Pro. (b) The full flight plan.

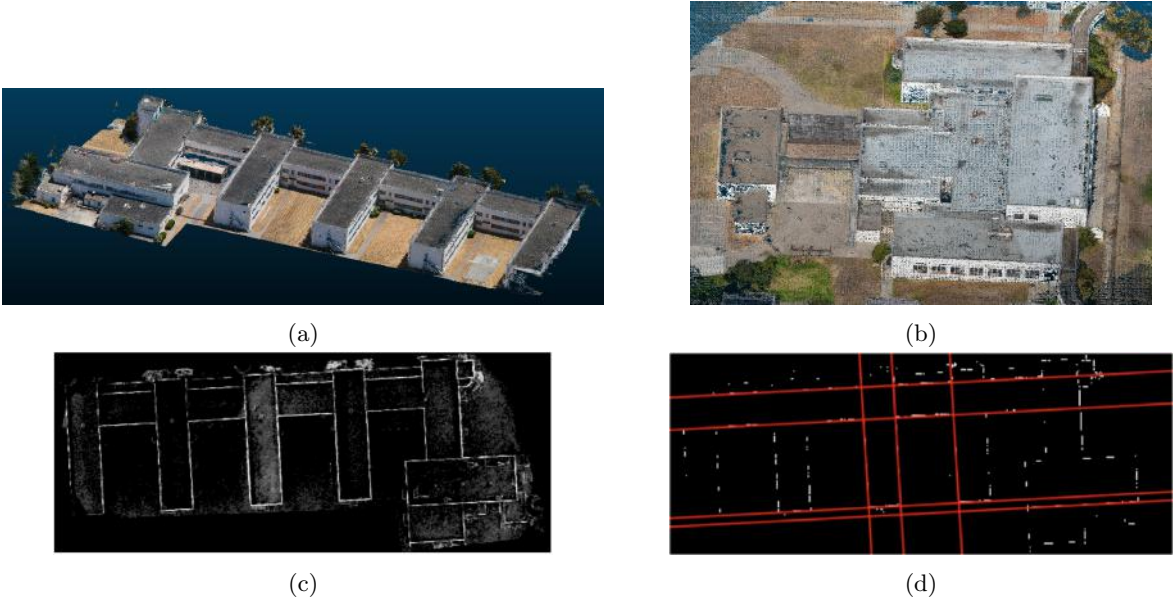


Figure 3.2: RGB Drone imagery for (a) Building 1. (b) Building 2. (c) Grid with points accumulated by grid cell for building 1. (d) Lines detected in the first iteration.

densities to those corresponding to walls have a value of 1. We opt to use,

$$\tau = \frac{1}{2}\rho_{avg}h_{min}$$

where ρ_{avg} is the running average density and h_{min} is the minimum wall height given as input. The factor of 2 is added because a wall may be split between two adjacent grid cells. Figure 3.2(c) shows an example of the accumulated grid.

Next, we apply the Hough transform to the accumulated grid. We found the approach by Hammoudi et al. [33] of clustering in Hough space to be fairly unreliable with the relatively high number of walls in our dataset, so we instead opt to find local peaks in Hough space and filter out spurious and duplicate lines later in the pipeline. For each line, we extract all points within a small perpendicular distance around the line, say $0.2m$, which mostly corresponds to the points on the wall. We further refine the line orientation by applying RANSAC to these extracted points to find the best-fit line corresponding to the wall points. Figure 3.2(d) shows an example of the detected lines.

3.2 Line Segmentation

The walls lie along the detected lines, and so we wish to determine the exact portion of each line overlapping a wall. To do so, we accumulate the points along the detected line into a 1D grid, with cells of length $0.3m$. Then, we once again threshold the 1D grid cells using the running average point density of all previously extracted walls, or 5% of the cells' maximum point count for the first iteration. A diagram showing a gridded line and the thresholded result is shown in Figures 3.3(a) and 3.3(b) respectively. Figure 3.3(c) shows one of the detected lines in Figure 3.2(d), trimmed so that its endpoints lie on the first and last grid cell with a value greater than the threshold.

After thresholding the grid, we apply a 1D median filter to the grid, whose size is twice the minimum wall length l_{min} given as input. This has the dual purpose of removing noise along the line, as well as filtering out wall fragments that lie perpendicular to the line, whose corresponding cells would still have high point counts in the image, but only for a few successive cells.

Next, we extract all contiguous sections of the grid with non-zero values to form line segments. The corresponding line segments for the trimmed line in Figure 3.3(c) is shown in 3.3(d). In each iteration, we repeatedly apply the line detection and line segmentation process, while removing the points associated with each detected line from consideration. This ensures that the Hough transform detects different lines at each iteration. This process continues until no more line segments are extracted.

3.3 Line Processing

We next post-process the extracted walls, in order to ensure they meet at endpoints. We first determine whether any endpoints of two different line segments are close together, and if so we close the gap between them by connecting them together, as shown in Figure 3.4(a). We also check whether by extending a line segment L_1 a fixed fraction, say 15%, of the length of the line, would cause it to intersect another line, L_2 . If so, we extend L_1 and split L_2 at the point of intersection into two segments, as shown in Figure 3.4(b). Finally, we delete duplicate segments by checking whether the endpoints of any pair of lines are close together. An example of the post-processed output for the data in Figure 3.3 is shown in Figure 3.4(c).

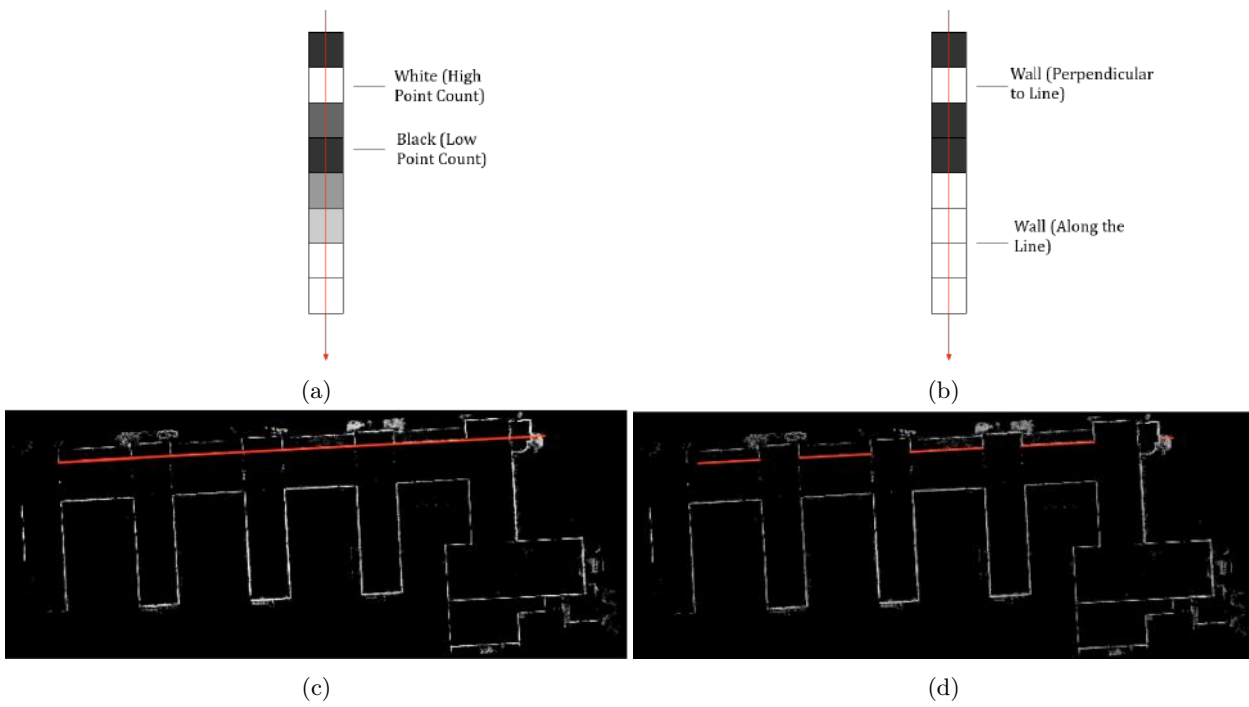


Figure 3.3: (a) Example of the 'gridded' line. (b) The line segment in (a) thresholded by density. (c) One of the detected lines, trimmed to match the support in the grid. (d) The extracted line segments.

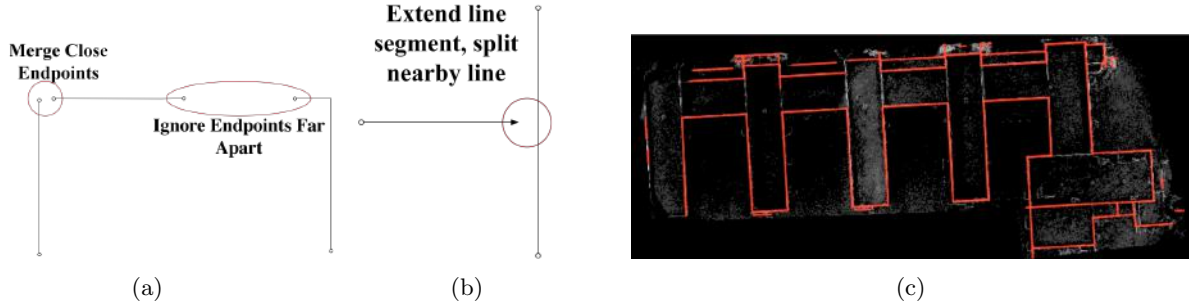


Figure 3.4: (a) Merging endpoints close together. (b) Splitting a nearby line segment. (c) Detected lines after post-processing.

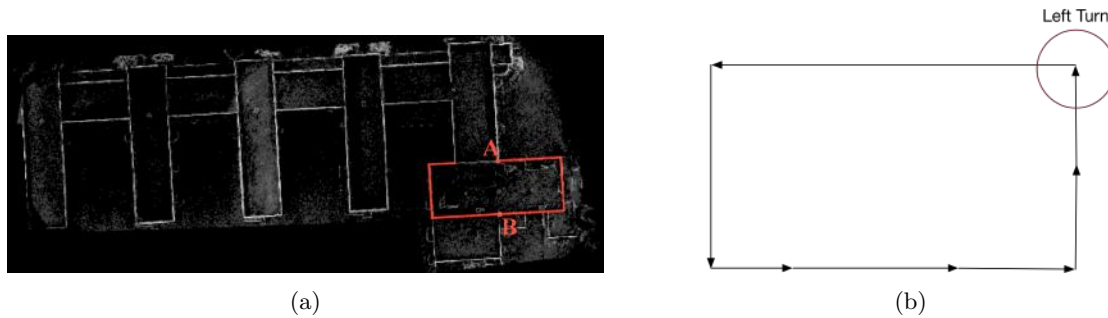


Figure 3.5: (a) Example of a 'partial' polygon in Building 1. (b) Counter-clockwise traversal.

3.4 Polygon Extraction

Once line segments have been extracted, they are converted into a polygonal footprint. Some line segments may have been missed in the previous steps, so in addition to converting existing line segments into full polygons, polygons are inferred from the arrangement of line segments, and then checked for underlying support in the point cloud. Our approach to polygonization can be summarized as follows. We first iteratively construct polygons from the line segments. To do so, we extract all closed polygons that are already present. Next, we determine polygon candidates, called 'partial polygons', which are sets of consecutive line segments not forming a closed polygon. An example of a partial polygon is shown in Figure 3.5(a). Then, we add new line segments following specific rules in order to 'complete' the partial polygon to form a closed polygon, and repeat the process. Once no new closed polygons are created, the polygon construction process terminates, and we estimate the height of all the closed polygons using the points lying inside them. Finally, we merge the closed polygons based on their height.

To extract the polygons, each line segment is treated as an edge in a planar graph. For two such edges, if their endpoints overlap, i.e. their endpoints are within an ϵ -distance from one another, we consider the two edges to be neighbors. Clearly, it is possible to traverse along the edges of any closed polygon until one returns to the starting node.

We consider the sequence of edges in a cycle to be an ‘ordering’ of the edges. By ordering edges in this manner, the edges are either in a clockwise or counter-clockwise ordering around the polygon. If we were to traverse along any convex polygon in clockwise order, we would only ever make right turns, and vice-versa for counter-clockwise order. An example of a counter-clockwise traversal of edges is shown in Figure 3.5(b).

Another important property is that an edge can be a part of at most two polygons, one on each side of the edge. By observing these two properties, we can extract all polygons from the edges. We start at an edge of a polygon, adding the neighbors connected to one of the ends of the edge. Then, we choose to make only right or left turns, walking along the edges continuously and adding the rightmost or leftmost edge until we return to the starting edge. Thus, if we start with an edge and choose to turn only left, we obtain a polygon on one side, namely the left side of the edge. If we instead choose to turn only right obtain the polygon on the right side. For a right (left) turn, the angle between two successive edges is defined to be negative (positive).

3.5 Partial-Polygon Construction

As can be seen in the output of the line processing step in Figure 3.4(c), there are initially only a few closed polygons present. As a result, in order to construct a polygonal footprint, closed polygons must be created from the incomplete ‘partial’ polygons that are present. We construct the partial polygon such as the one shown in Figure 3.5(a) by following a similar polygon extraction process described in Section 4.1. These partial polygons are sets of edges that would be grouped together without actually forming a closed polygon, i.e. missing a contiguous section. Since a traversal through a partial polygon does not return to the starting edge, constructing these partial polygons requires some care. Traversing edges in a single direction will not give us all the edges of a partial polygon unless we start at an edge which contains one of the two endpoints of the partial polygon. However, we can find such an edge by performing an initial traversal of the polygon in one direction. Once we reach an edge with no valid neighbors, i.e. one of the edges which contains an endpoint of the partial polygon, we reverse our traversal direction and the direction of the turns, adding any edge we visit. In this reversed traversal, we visit all edges from one endpoint to the other, obtaining all the edges in the partial polygon. An example of a partial polygon $ACDEFGHI$ is shown in Figure 3.6(a).

3.6 Polygon Completion

Once we have constructed the partial polygons, we need to construct closed polygons from them. Given the partial polygon $ACDEFGHI$ in Figure 3.6(a), there are many potential approaches for constructing a closed polygon, some leading to higher polygon counts and some to fewer. In the example shown in the figure, a closed polygon $BCDEF$ is constructed, and a new partial polygon $ABFGHI$ is created. We could have simply closed the entire partial polygon $ACDEFGHI$ by extending the edges CA and HI so that they intersect, but this approach would force us to assign only a single height to the entire polygon. By creating smaller polygons instead, we can achieve finer granularity in height assignment which is useful in the height based merging step to be described later.

We opt to create an additional edge that closes a portion of the partial polygon. This has the effect of creating a small closed polygon, and potentially another partial polygon to be completed in future iterations. The implementation of the polygon completion process proceeds as follows. We consider two distinct classes of partial polygons: those with 2 or more corners, and those with only one corner. For each partial polygon with 2 or more corners, where a corner is connected to two edges with a non-zero angle between them, we first take the two adjacent corners of the polygon with the smallest distance between them. In Figure 3.6(a), this corresponds to corners C and E . Next, we take the smallest edge connected to either of these

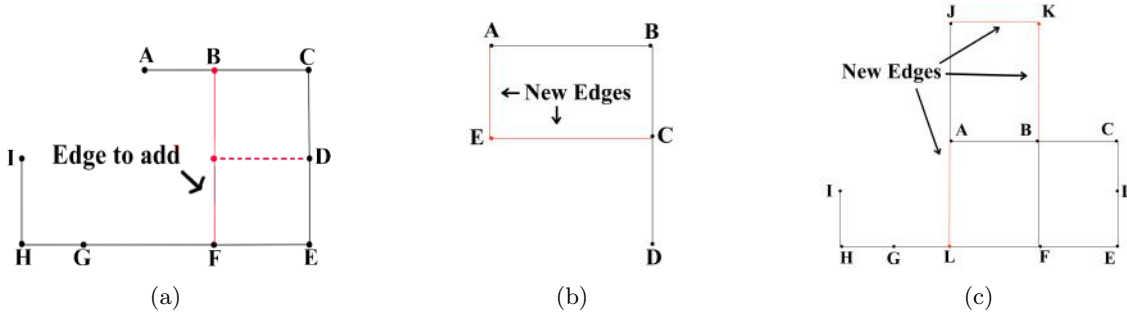


Figure 3.6: (a) Completion process for a partial polygon with 3 corners. (b) Propagation effect of completing one partial polygon to the completion of another in a subsequent iteration. (c) Completion process for a partial polygon with 1 corner.

corners, such that it does not lie *between* the two corners. In Figure 3.6(b), this corresponds to edge FE . From these three resulting endpoints, namely CEF , we then complete the rectangle by adding edge BF and splitting edge AC into two pieces, namely AB and BC . This process completes a portion of the partial polygon. In this process, choosing the corners with the smallest distance between them tends to lead to more proportional polygons. Although we could have conceivably chosen the largest distance instead, this would have led to long, skinny, small polygons in future iterations, resulting in error prone height estimation due to a low point count.

For the case where the partial polygon only has one corner, we instead complete the polygon using the two edges extending from the corner. An example of this is shown in Figure 3.6(b), where the only corner is endpoint B . As a result, edges AB and BC are selected, and edges CE and AE are created. The reason we require at least one corner is because any polygon with no corners corresponds to a single line, for which we have no information how far a polygon might extend.

Figure 3.6(a) shows one iteration of the completion process, where the partial polygon closes at endpoint F , and the edge BF is added as a result. Note that the addition of an edge in one iteration can lead to new partial polygons in subsequent iterations, creating a 'domino' effect, which enables our method to complete the entire partial polygon $ACDEFGHI$, while creating several smaller polygons in the process. This allows us to segment the polygons in a more granular fashion based on their heights, since we can estimate the height for each polygon individually. We also perform the same process on the already closed polygons, in order to subdivide them further. For example, in Figure 3.6(a), we add an additional edge starting at endpoint D and ending on the line segment BF , represented by the dashed line.

The newly added edges, such as BF , create new endpoints such as B , which may cause other partial polygons to close at B in future iterations if they share the edge BC . This is shown in Figure 3.6(c), where B causes a neighboring partial polygon, namely $JABC$, to create the edges BK and KJ in the next iteration. This is useful because facades are often aligned with one another even if they are not directly in contact. For example, for the partial polygon of Figure 3.5(a) the two facades, whose points of intersection with the partial polygon are labeled A and B respectively, are aligned vertically but do not come in contact with one another. If either facade is missing from the line detection step, the other facade is used to recover it.

For each edge we add as a result of the polygon completion process, we check that the height profile of the points along the edge are consistent and above h_{min} , and remove it otherwise. This is to ensure that the edge is not part of the ground and does not lie across rooftops of different heights. To obtain the heights along each of these newly added edges, we extract the 3D points from the point cloud around each edge

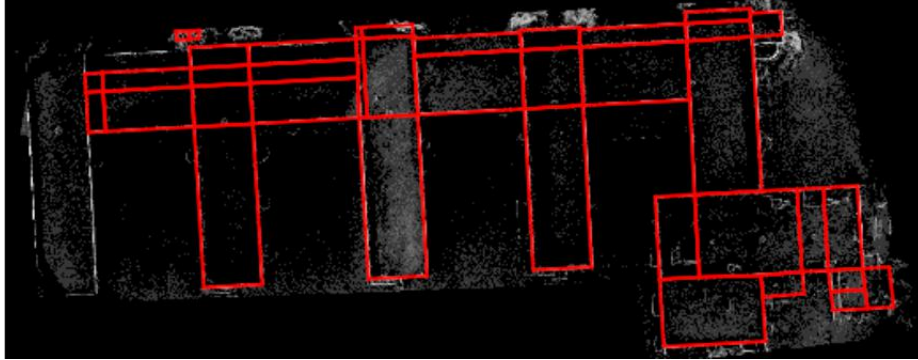


Figure 3.7: Example of the completed (unmerged) polygons after all iterations have completed.

similar to Section 3.3, segment the points into a 1D grid along the line, and assign to each grid cell the 90th percentile for the height of the points that lie inside them, as these correspond to the rooftop heights with high probability. Then, we propose to take the first and third quartiles of the values over all the grid cells corresponding to the edge under consideration, and verify that they differ no more than a small amount, e.g. 0.5 meters.

Once all the discovered partial polygons in the iteration have been completed, the iteration is complete. Since new partial polygons may have been formed as a result of this process, the steps in Sections 4.1, 4.2, and 4.3 are repeated. This creates a 'domino' effect where the additional edges at each new iteration allow previously undiscovered partial polygons to be constructed, until no more edges or closed polygons are added. We then remove all edges that are not part of any closed polygon. An example of the completed polygons for the point cloud shown in Figure 3.2(a) is shown in Figure 3.7.

3.7 Polygon Merging

Once all the polygons have been completed, we recursively merge adjacent polygons if they are close together in average height, e.g. within 0.5 meters of each other. To do so, we iterate through each edge that is shared between two polygons, and check their heights. After merging two polygons, we update the height of the merged polygon by simply performing a weighted average of the average heights of the two polygons, where the weighting factor is based on the area of the polygon. Using the average height has only been tested to work for flat rooftops, though it could conceivably work with non-flat rooftops as well. The merged polygons for Building 1 is shown in Figure 4.3(a). Figures 4.3(b) and 4.3(c) show the resulting heatmap of the heights of the merged polygons for Buildings 1 and 2 respectively. Additionally, 3D renderings for the footprint of the two buildings are shown in Figures 4.3(d) and 4.3(e).

3.8 3D Building Reconstruction

After obtaining a set of closed polygons, we estimate their height by averaging the height of the points that lie inside it, within a fixed distance of the boundary. We use our 2D grid to compute the average height of the points in each cell, and also average over the height of the cells. By taking the average over grid cells rather than directly using point height, the height computation is spread out across the entire area of the

polygon, minimizing the effects of objects, such as air conditioning units, that may lie on top of the roof, while also speeding up computation time. Since we are only interested in the height of the roof, we ignore cells near the boundary of the polygon as these contain the facade points as well. Using the average height inherently limits the footprint to flat rooftops, since height for pyramid-shaped rooftops is computed at their apex. If we were to use the 90th percentile rather than average height, our approach could potentially also be used for pyramid-shaped rooftops.

Chapter 4

Experimental Results

We collected 2 datasets, each consisting of several hundred images of two buildings in Alameda, California. Images are captured using the DJI Spark drone with a 12MP RGB camera, 25mm lens, and f/2.6 aperture. The buildings are captured in a circular pattern, with a 70% overlap between successive images. Images are then processed using Pix4D as described in Section 4.1 to generate a dense cloud with 6 million points.

This chapter is organized as follows. Section 4.1 details our data collection and processing for the generation of the point cloud and camera poses. Section 4.2 shows our results for the building and vegetation segmentation methods of Section 2. Section 4.3 shows our results for the building footprint extraction methods of Section 3. Section 4.4 explains how we used an extracted building footprint to obtain the window-to-wall ratio of a facade, given input RGB images with windows already detected. Finally, Section 4.5 shows a set of thermal images that are overlaid with the extracted 3D building model.

4.1 Data Collection and Photogrammetry

After a building is selected, we plan the flight so that the drone captures images at most 30 meters from the building, following many circular paths with a gimbal angle of -35° off the horizon. This ensures that captured images are sufficiently close to the building to allow for a high quality reconstruction. Successive images are captured to have a 85% front overlap with one another and are geo-tagged. Examples of the flight plan for a portion of a building can be seen in Figures 3.1(a) and 3.1(b). In order to comply with federal guidelines, the area is scouted so that there are no persons beneath the flight path of the drone. Flights during rain and heavy wind are avoided, for both safety reasons and to ensure the images are of high quality.

There exist several commercial applications for the construction of 3D models and 3D point clouds from a set of images. In order for these applications to work effectively, images generally require high amounts of both frontal overlap, i.e. successive images in the flight direction, and side overlap, i.e. between flight tracks. We opted use the Pix4D software suite [34] to generate a dense point cloud and calculate the internal and external parameters for camera calibration from our collection of input images. Pix4D additionally provides a set of camera pose matrices as output based on the internal and external camera parameters calculated for each input image. This matrix is used to convert the 3D coordinates of a point in the generated dense cloud to the 2D coordinate of a pixel in an undistorted input image.

	Building			Vegetation		
	Ours	Pix4D	PointNet++	Ours	Pix4D	PointNet++
Precision	0.90	0.98	0.76	0.72	0.42	0.47
Recall	0.90	0.48	0.90	0.91	0.87	0.47
Jaccard	0.82	0.48	0.69	0.64	0.41	0.31
F1	0.90	0.64	0.82	0.79	0.58	0.47

Table 4.1: Precision, Recall, F1 scores and Jaccard Similarity Coefficient scores for our methodology on Dataset 1.

	Building			Vegetation		
	Ours	Pix4D	PointNet++	Ours	Pix4D	PointNet++
Precision	0.90	0.99	0.75	0.83	0.25	0.22
Recall	0.85	0.62	0.73	0.55	0.90	0.63
Jaccard	0.77	0.58	0.60	0.49	0.24	0.19
F1	0.87	0.76	0.74	0.66	0.39	0.33

Table 4.2: Precision, Recall, F1 scores and Jaccard Similarity Coefficient scores for our methodology on Dataset 2.

4.2 Building and Vegetation Segmentation Results

Pix4D is used to generate a point classification for each of these points, which we use as a comparison to the results of our method. For the occlusion detection, we find that using a single block-reduce size of (10,10) for a 12MP camera image works well for our purposes. We use the full image dataset to generate the dense point cloud, but only use 1 in 5 of the dataset images for segmentation, to improve runtime. Because we use a subset of photogrammetry images for segmentation, a percentage of 3D points may not be mapped to any 2D pixels in the segmented images. We find this to be ~ 7000 points (0.12%) for Dataset 1 and ~ 34100 points for Dataset 2 (0.31%). These are small enough to have a negligible impact on the results. An example point cloud is shown in Figure 4.1(a), with the resulting predictions from our method, Pix4D, and PointNet++ in Figures 4.1(b), 4.1(c), and 4.1(d) respectively, and the ground truth in Figure 4.1(e). Qualitatively, for both datasets Pix4D and PointNet++ leave a large number of vegetation points unclassified and misclassify a large number of building points as vegetation points.

To evaluate the performance of our method, we compute the F1 score and Jaccard Similarity Coefficient score for the two classes: ‘building’ and ‘vegetation’. For comparison, we also compute the scores from the output of Pix4D’s classifier and for PointNet++ trained on the Semantic3D dataset. Note that for these classifiers, we combine the LAS classifications of ‘high’, ‘medium’, and ‘low vegetation’ into a singular ‘vegetation’ class, as our work is not focused on distinguishing between the different types of vegetation. The results are shown in Tables 4.2 and 4.1 for Datasets 1 and 2, respectively. Comparing the accuracy scores, our method significantly outperforms both Pix4D and PointNet++ in detecting vegetation points for both Jaccard and F1 scores on both datasets, significantly outperforms both classifiers in detecting building points for Dataset 2, and modestly outperforms both classifiers in detecting building points for Dataset 1.

Additionally, we evaluate confidence metrics for points in our classified dataset. Figure 4.2a shows a distribution of the number of unique classes (‘vegetation’, ‘building’, ‘other’) that a point has been assigned by segmented pixels from separate images. This represents how likely the classified pixels among all images

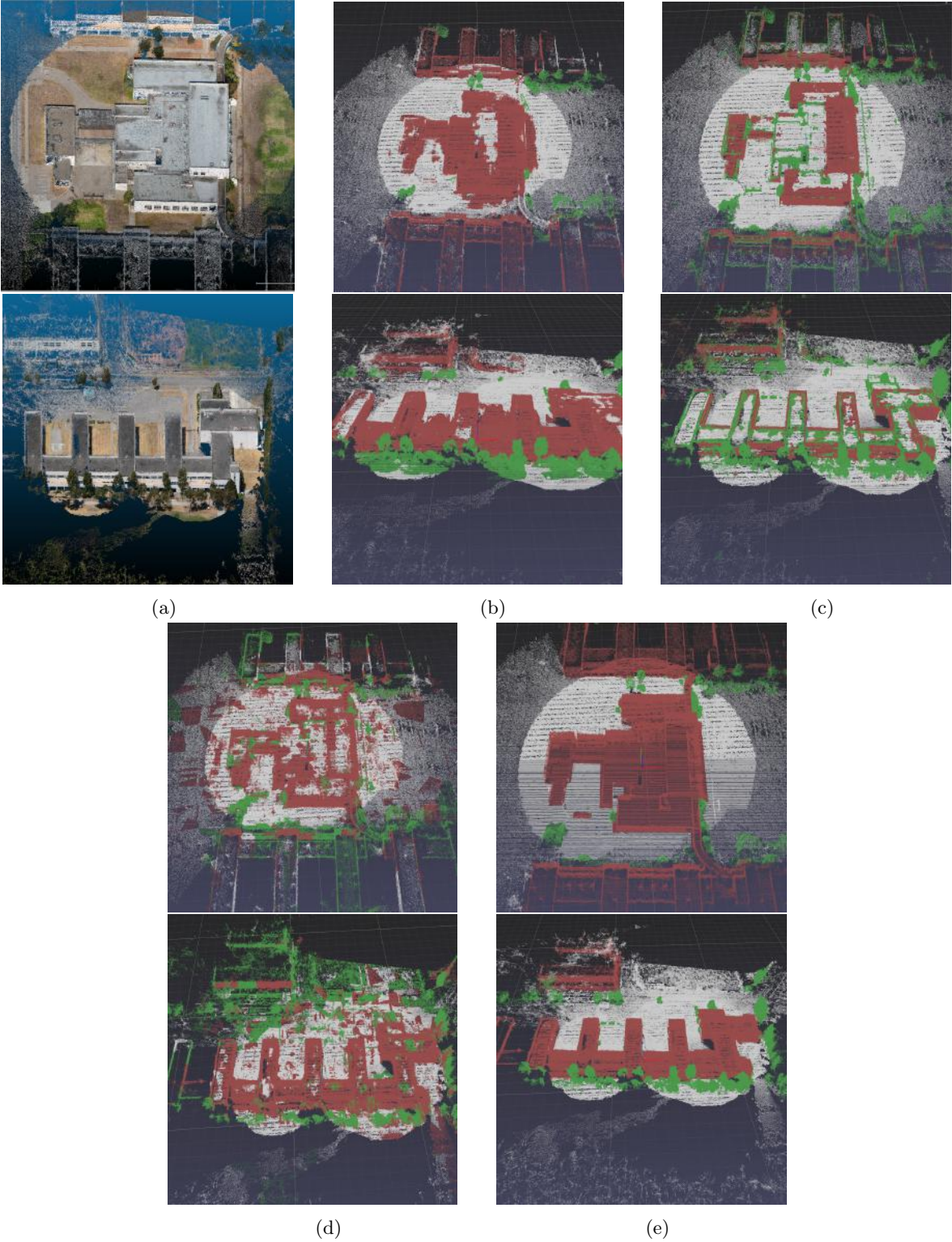


Figure 4.1: (a) The example point clouds. Classification results from: (b) our method; (c) Pix4D; (d) PointNet++; (e) The ground truth. Green: Vegetation, Red: Building, White: Other

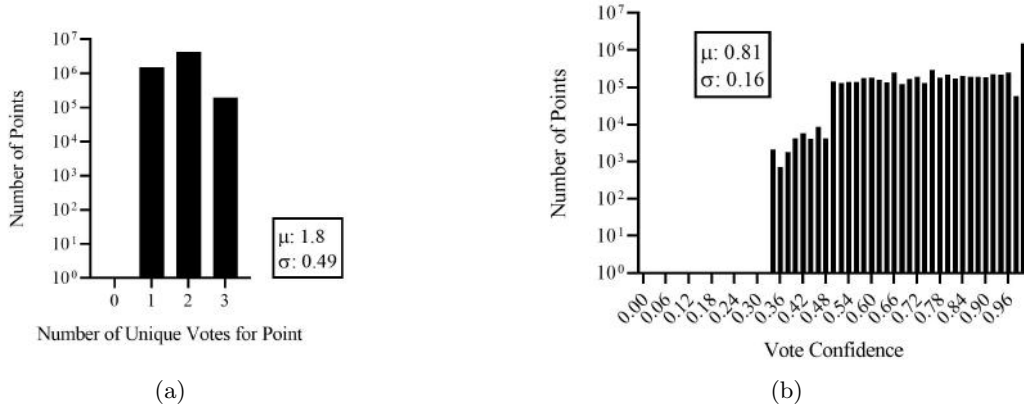


Figure 4.2: (a) Histogram of number of unique votes per point. (b) Histogram of winning vote’s confidence per point.

mapping to a given point agree with one another. The mean of this value is 1.8. We also show a histogram of ‘vote confidence’ in Figure 4.2b. We define the confidence of a point as the ratio of the number of majority classifications to the number of total classifications. This represents how ‘strong’ the majority class is relative to other assigned classes. This distribution has a mean of 0.81, showing that there is often a strong majority among the classes assigned to a point.

4.3 Building Footprint Evaluation

The proposed footprint extraction has been performed on two buildings in Alameda, with point clouds shown in Figures 3.2(a) and 3.2(b) using the full pipeline as described in Chapter 3. To evaluate our results we hand-labeled the two buildings using satellite imagery of the building from Google Earth. We manually placed markers around the building outline to obtain their coordinates, which we compiled into polygons. To account for alignment discrepancies, we performed a grid-search to find the best alignment between the generated footprint and the ground truth. The grid-search was performed by iteratively shifting the generated footprint at different orders of magnitude, and choosing the one that maximized the intersection area between the generated building footprint and the ground truth. Next, we computed the F1 and IOU scores for each building as shown in Table 4.3. Our method achieves F1 scores and IOU scores above 0.9 for both buildings. Alameda Building 1 actually corresponds to a larger building complex, for which only a section was imaged. For reasonable comparison, the ground truth for Alameda building 1 was cut-off at the far-left side, since most of its facades were not present in the images, and so the number of points corresponding to it in the point cloud was low.

4.4 Window-to-Wall Ratio Estimation

The building footprint was additionally used to extract the window-to-wall ratio of the windows and building facades of an image. A block diagram for the entire process is shown in Figure 4.4. Window-to-wall ratio is crucial for energy modeling of a building, because it has significant effects on heating, cooling, lighting,

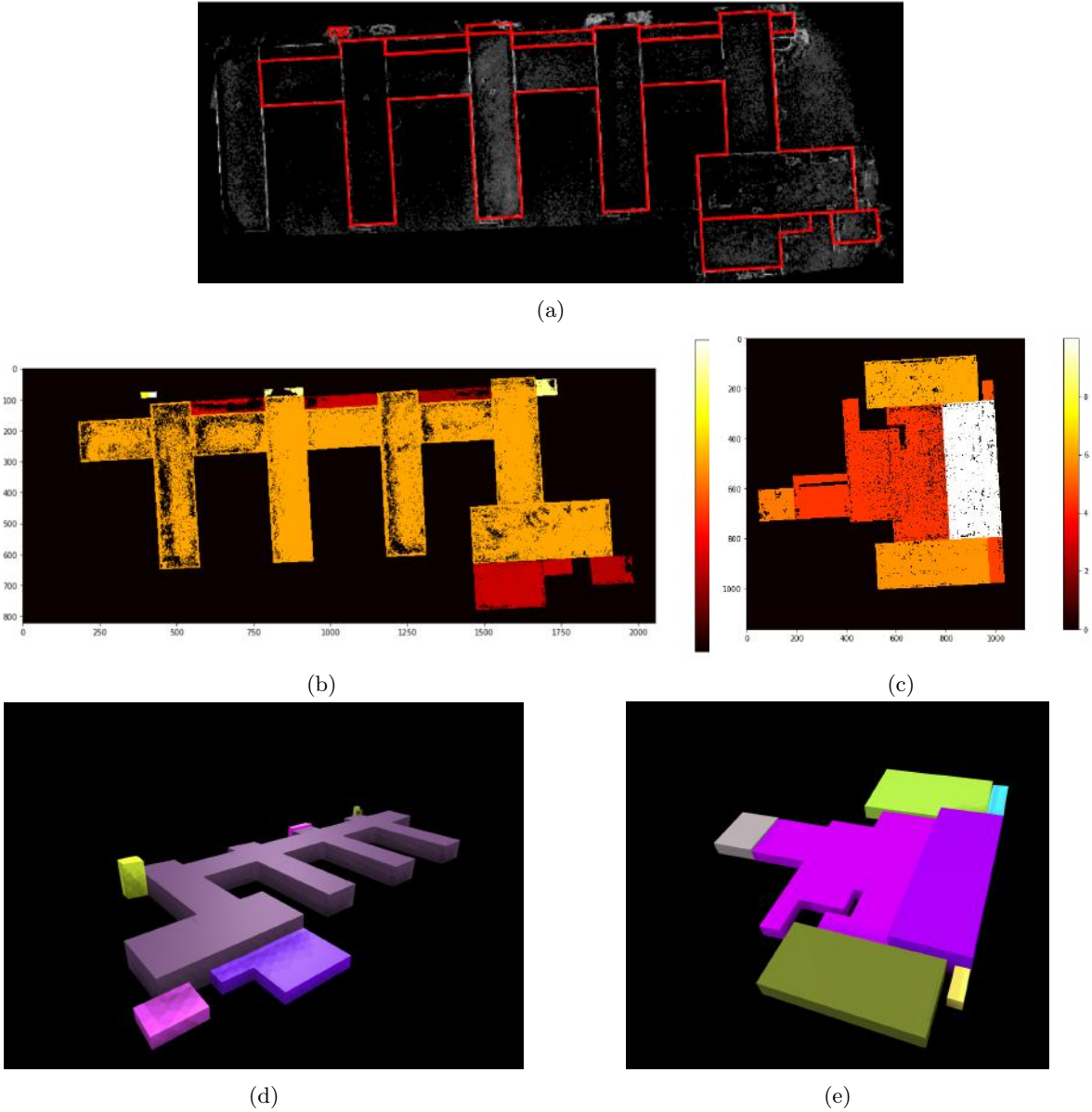


Figure 4.3: (a) Polygons for Building 1 after the merge process has completed. Heatmap of the heights of merged polygons for (b) Building 1. (c) Building 2. 3D Rendering of Alameda Buildings (d) 1 and (e) 2, from the extracted 2D polygon footprints, with polygons extruded by height.

	F1 Score	IOU Score
Alameda Building 1	0.934	0.901
Alameda Building 2	0.950	0.930

Table 4.3: F1 and IOU scores for our method evaluated on the Alameda datasets

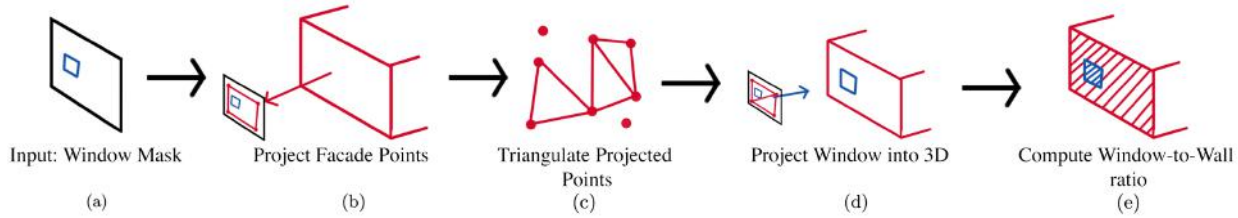


Figure 4.4: Flowchart for the window-to-wall ratio extraction process.

ventilation, and more. For this approach, we assume that windows have already been detected on a 2D image as depicted in Figure 4.4(a), and we simply wish to compute their areas in 3D along with the area of their corresponding facades. To do so, we take the facade corner points of the extracted 3D building model, and project them onto the RGB drone camera image that corresponds to the input image with the detected windows. This is shown in Figure 4.4(b). The projective distance for each of the corners of the facade is stored. This will be used to determine the ordering of the facades along the optical axis. An example for the projection on one RGB image is shown in Figure 4.5(a), with the projected building model overlaid on top of an RGB image.

We use a window mask for a 2D image as input, shown for the RGB image for Figure 4.5(a) in Figure 4.5(b). Then, for each projected facade onto its corresponding image, we determine if it contains each window by checking whether the window lies within the 2D facade corners in the image. Since we are not explicitly determining whether a facade is occluding another, a window may lie on many of the projected facades, so we need choose the closest such facade to the optical center to find the facade the windows actually lie on. This is done by choosing the facade with the minimum projective distance for the particular RGB image. Once we have determined the correspondence between the facades and their respective windows in 2D, we wish to use this information to back-project the windows into 3D and onto the corresponding 3D facade. Since the facades could have an arbitrary shape, especially the roof polygons, we perform a triangulation of all the projected facade corners via Delaunay Triangulation in 2D, as shown in Figure 4.4(c). We then determine the resulting triangles that each of the window vertices lie within. This allows us to obtain the position of each window point in barycentric coordinates, i.e. relative to the facade corners, as depicted in Figure 4.6 where the example window point W can be written in terms of the facade points A, B and C with the equation:

$$W = \alpha A + \beta B + \gamma C \quad (4.1)$$

For $\alpha + \beta + \gamma = 1$. Then, using the barycentric coordinates of the window points, we can compute a position for the window points in 3D space, as shown in Figure 4.4(d). This is done by plugging in the coordinates of the 3D facade points from the 3D building model into the same barycentric equation, i.e. replacing the 2D facade points A, B, C with the corresponding 3D facade points, and using the same α, β , and γ . Once we have the 3D coordinates for the window points we can compute the window to wall ratio by simply

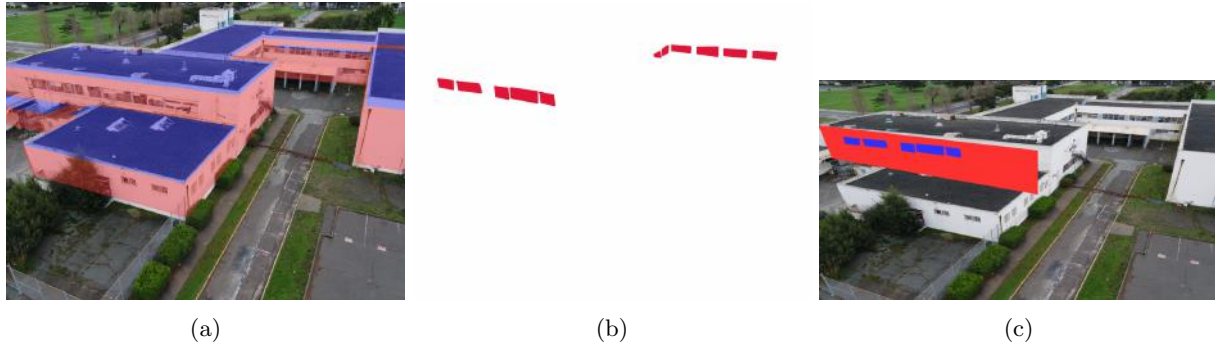


Figure 4.5: (a) 3D building model projected onto a 2D image, with roof polygons overlaid in blue and facades in red; (b) The corresponding window mask; (c) Facade (red) with its corresponding windows (blue).

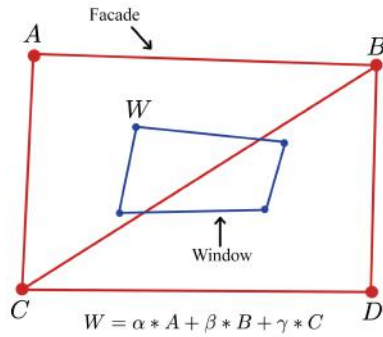


Figure 4.6: Triangulation of the projected facade points with the facade in red and window in blue.

calculating the surface area of the facade using its 3D coordinates, along with the surface area of each of its windows with their 3D coordinates. This is shown in 4.4(e). For the facade shown in Figure 4.5(c), the window-to-wall ratio is 11.01%.

4.5 Thermal Image Overlay

We additionally extracted a point cloud from a project that included both thermal images and RGB images. The process for creating the point cloud with both types of images is detailed in Appendix A.2. After generating a 3D model from our building footprint extraction, we additionally projected each of its facades onto a set of thermal images. A Zenmuse XT camera was used to capture thermal images of one the buildings in Alameda. Four such images with the 3D model overlaid on top can be seen in Figure 4.7.

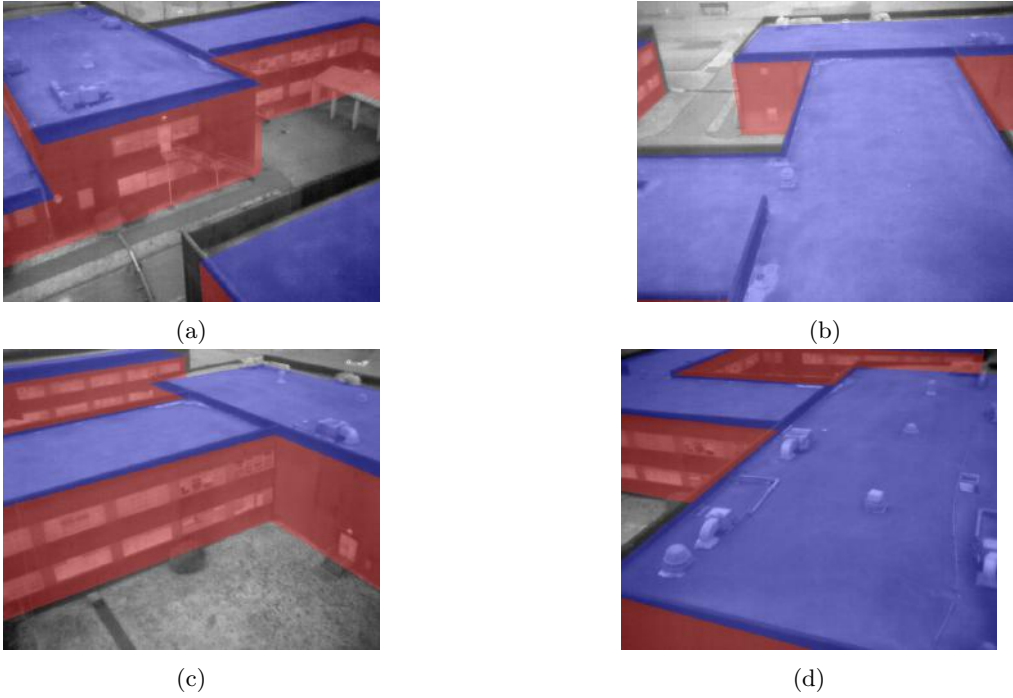


Figure 4.7: 3D building model overlaid on top of several thermal (IR) images, with the roof facade in blue and wall facades in red.

Chapter 5

Conclusions and Future Work

We have presented two pipelines, one for the semantic segmentation of 3D point clouds and the other for generating building footprints from 3D point clouds. For both pipelines, we utilized 3D point clouds constructed from two buildings in Alameda, California. To do so, we captured RGB images using a drone and then processed them with photogrammetry software to construct a 3D point cloud. By utilizing photogrammetry to accomplish this task, we avoid the need for LiDAR or other depth information from the environment.

We address a fundamental limitation of the segmentation approach, namely occlusion, with a novel occlusion detection method. Results are favorable compared to state-of-the-art semantic segmentation built-in to an existing photogrammetry software suite as well as compared to the results using PointNet++. Future improvements to this work include a confidence score for point classifications, calculated with a Bayesian method on the list of classes for each point. This would likely improve results compared to the current majority vote of pixel classes corresponding to the point. Note that the choice of 2D image segmentation approach is independent from our pipeline, so as 2D image segmenters improve, so too will the results of our pipeline.

Our methodology for generating building footprints achieves good results on both point clouds, demonstrating that the polygonization approach is a promising avenue for generating building footprints. One shortcoming of our footprint extraction approach is that sparse line segments can cause potential polygons to be missed. Future work could focus on either improving the line segment detection algorithm or further developing the polygonization logic. Moreover, tree detection logic for point clouds could be integrated with our method to improve results, as trees incorrectly result in polygons. Additionally, the method as presented only accounts for flat rooftops, as non-flat rooftops were not present in our test datasets. However, the only portion of our method that assumes a flat rooftop is merging polygons, and so our methods could potentially be extended to non-flat rooftops by adjusting how the height profiling is performed. Finally, our approach does not readily extend to buildings that are less regular, such as circular or pyramid-shaped buildings. This is because circular buildings are not easily polygonized. Meanwhile, the facades of a pyramid-shaped building do not lie along the vertical axis, making their extraction problematic for our method.

Bibliography

- [1] M. Carlberg, P. Gao, G. Chen, and A. Zakhor, “Classifying urban landscape in aerial lidar using 3d shape analysis,” in *2009 16th IEEE International Conference on Image Processing (ICIP)*, Nov 2009, pp. 1701–1704.
- [2] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas, “Pointnet: Deep learning on point sets for 3d classification and segmentation,” 2016.
- [3] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas, “Pointnet++: Deep hierarchical feature learning on point sets in a metric space,” 2017.
- [4] Loïc Landrieu and Martin Simonovsky, “Large-scale point cloud semantic segmentation with superpoint graphs,” *CoRR*, vol. abs/1711.09869, 2017.
- [5] Alexandre Boulch, “Convpoint: Continuous convolutions for point cloud processing,” *Computers Graphics*, vol. 88, pp. 24 – 34, 2020.
- [6] Lyne P. Tchapmi, Christopher B. Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese, “Segcloud: Semantic segmentation of 3d point clouds,” 2017.
- [7] E. Özdemir and F. Remondino, “Classification of aerial point clouds with deep learning,” *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLII-2/W13, pp. 103–110, 2019.
- [8] Vishakh Hegde and Reza Zadeh, “Fusionnet: 3d object classification using multiple data representations,” 2016.
- [9] Ji Hou, Angela Dai, and Matthias Nießner, “3d-sis: 3d semantic instance segmentation of rgb-d scans,” 2018.
- [10] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz, “SPLATNet: Sparse lattice networks for point cloud processing,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2530–2539.
- [11] Hao Pan, Shilin Liu, Yang Liu, and Xin Tong, “Convolutional neural networks on 3d surfaces using parallel frames,” *ArXiv*, vol. abs/1808.04952, 2018.
- [12] Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou, “Tangent convolutions for dense prediction in 3d,” 06 2018, pp. 3887–3896.

- [13] A. Boulch, B. Le Saux, and N. Audebert, “Unstructured point cloud semantic labeling using deep segmentation networks,” in *Proceedings of the Workshop on 3D Object Retrieval*, Goslar, DEU, 2017, 3Dor '17, p. 17–24, Eurographics Association.
- [14] Alexandre Boulch, Joris Guerry, Bertrand Saux, and Nicolas Audebert, “Snapnet: 3d point cloud semantic labeling with 2d deep segmentation networks,” *Computers Graphics*, vol. 71, 12 2017.
- [15] Norbert Haala and Martin Kada, “An update on automatic 3d building reconstruction,” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 65, no. 6, pp. 570–580, 2010.
- [16] Peter Crawford, “Effects of aerial lidar data density on the accuracy of building reconstruction,” 2018.
- [17] Taejung Kim, Ts Javzandulam, and Tae-Yoon Lee, “Semiautomatic reconstruction of building height and footprints from single satellite images,” in *2007 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, 2007, pp. 4737–4740.
- [18] Nitin L. Gavankar and Sanjay Kumar Ghosh, “Automatic building footprint extraction from high-resolution satellite image using mathematical morphology,” *European Journal of Remote Sensing*, vol. 51, no. 1, pp. 182–193, 2018.
- [19] Aaron K Shackelford, Curt H Davis, and Xiangyun Wang, “Automated 2-d building footprint extraction from high-resolution satellite multispectral imagery,” in *IGARSS 2004. 2004 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, 2004, vol. 3, pp. 1996–1999.
- [20] Salman Ahmadi, M.J. Valadan Zoej, Hamid Ebadi, Hamid Abrishami Moghaddam, and Ali Mohammadzadeh, “Automatic urban building boundary extraction from high resolution aerial images using an innovative model of active contours,” *International Journal of Applied Earth Observation and Geoinformation*, vol. 12, no. 3, pp. 150 – 157, 2010.
- [21] G Sohn and IJ Dowman, “Extraction of buildings from high resolution satellite data,” 2001.
- [22] Ksenia Bittner, Fathallah Adam, Shiyong Cui, Marco Körner, and Peter Reinartz, “Building footprint extraction from vhr remote sensing images combined with normalized dsms using fused fully convolutional networks,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 11, no. 8, pp. 2615–2629, 2018.
- [23] Microsoft, *US Building Footprints*, 2018.
- [24] Ajit Sampath and Jie Shan, “Building boundary tracing and regularization from airborne lidar point clouds,” *Photogrammetric Engineering and Remote Sensing*, vol. 73, 07 2007.
- [25] Keqi Zhang, Jianhua Yan, and S-C Chen, “Automatic construction of building footprints from airborne lidar data,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 44, no. 9, pp. 2523–2533, 2006.
- [26] Mathieu Brédif, Olivier Tournaire, Bruno Vallet, and Nicolas Champion, “Extracting polygonal building footprints from digital surface models: A fully-automatic global optimization framework,” *ISPRS journal of photogrammetry and remote sensing*, vol. 77, pp. 57–65, 2013.
- [27] Ksenia Davydova, Shiyong Cui, and Peter Reinartz, “Building footprint extraction from digital surface models using neural networks,” in *Image and Signal Processing for Remote Sensing XXII*. International Society for Optics and Photonics, 2016, vol. 10004, p. 100040J.

- [28] F Nex and Fabio Remondino, “Automatic roof outlines reconstruction from photogrammetric dsm,” *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 1, no. 3, pp. 257–262, 2012.
- [29] Chungan Lin and Ramakant Nevatia, “Building detection and description from a single intensity image,” *Computer Vision and Image Understanding*, vol. 72, no. 2, pp. 101 – 121, 1998.
- [30] Ali Ozgun Ok, “Automated detection of buildings from single vhr multispectral images using shadow information and graph cuts,” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 86, pp. 21 – 40, 2013.
- [31] R. B. Irvin and D. M. McKeown, “Methods for exploiting the relationship between buildings and their shadows in aerial imagery,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, no. 6, pp. 1564–1575, 1989.
- [32] D Koc San and M Turker, “Building extraction from high resolution satellite images using hough transform,” 2010.
- [33] Karim Hammoudi, Fadi Dornaika, and Nicolas Paparoditis, “Extracting building footprints from 3d point clouds using terrestrial laser scanning at street level,” 09 2009, vol. 38.
- [34] Pix4D SA, “Pix4d,” .
- [35] Yuxing Xie, Jiaojiao Tian, and Xiao Xiang Zhu, “A review of point cloud semantic segmentation,” 2019.
- [36] J. L. Schönberger and J. Frahm, “Structure-from-motion revisited,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 4104–4113.
- [37] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 3431–3440.
- [38] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun, “Unified perceptual parsing for scene understanding,” in *European Conference on Computer Vision*. Springer, 2018.
- [39] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba, “Scene parsing through ade20k dataset,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Appendix A

Appendix

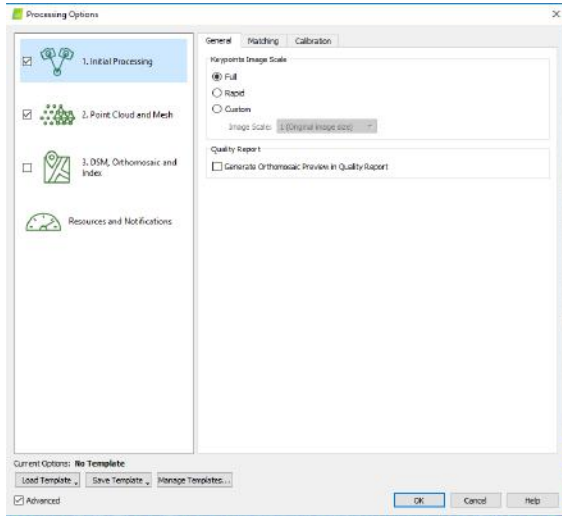
This appendix describes our process for generating point clouds and 3D meshes from Pix4D. Section A.1 outlines our procedure for generating point clouds from RGB images. Section A.2 additionally outlines the procedure for merging an RGB project with a thermal project, for the purpose of generating camera pose matrices for thermal images. Finally, Section A.3 describes the process for creating 3D models with Pix4D that are texture mapped with thermal images.

A.1 RGB Point Cloud Generation

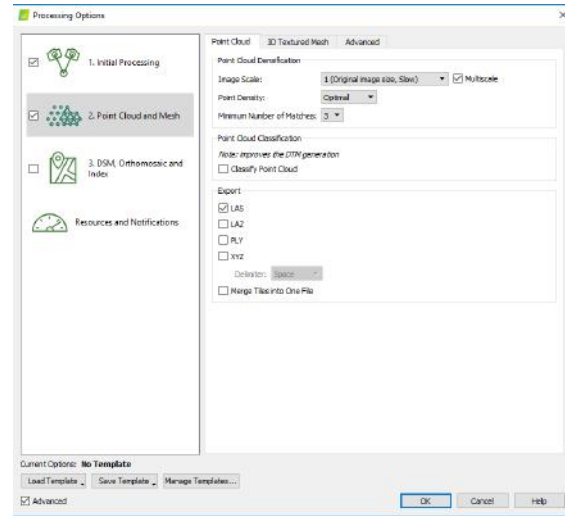
After capturing a sufficient number of geo-tagged RGB images, we process them in Pix4D to create a point cloud . In Pix4D, we opted to use the '3D Models' template. For the 'Initial Processing' step, we set the 'Keypoints Image Scale' to 'Full', leaving the other settings to their default value, as seen in Figure A.1(a). For the 'Point Cloud and Mesh' step, we adjusted the image scale to use the original image sizes, as shown in Figure A.1(b). Then, we simply run Pix4D to create the point cloud as seen in Figure A.2.

A.2 Merged RGB and Thermal Image Projects

We additionally extracted a point cloud from a project that included both thermal images and RGB images. We used the Zenmuse XT camera to capture the thermal images. In order to create a point cloud using both image types in Pix4D, we constructed two separate projects for the thermal images and the RGB images. After the 'Initial Processing' step has completed for both projects, we added several Manual Tie Points (MTPs) for corners of the building in each projects, so that the MTPs corresponded with one another across the two projects. Manual tie points were added by selecting one of the tie points from the result of the 'Initial Processing' step, clicking 'New Tie Point', and then selecting in several different images the location of the tie point. An example of the MTPs is shown for the RGB project in Figure A.3. Once the tie points were added, we merged the two projects into a singular one and ran the 'Point Cloud and Mesh' step for this merged project, obtaining a point cloud and the camera poses for both the RGB and thermal input images. The settings for the point cloud generation were the same as in A.1. The resulting point cloud with the drone flight path, i.e. a representation of the camera images, is shown in Figure A.3(b).



(a)



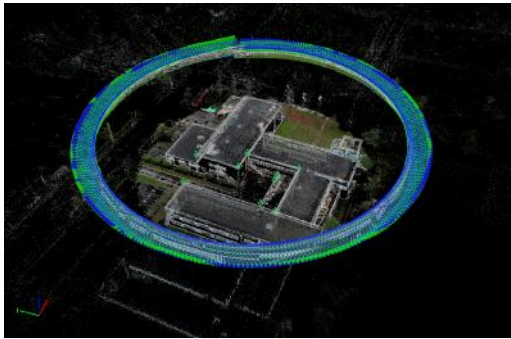
(b)

Figure A.1: (a) Settings for the 'Initial Processing' step. (b) Settings for the 'Point Cloud and Mesh' step.

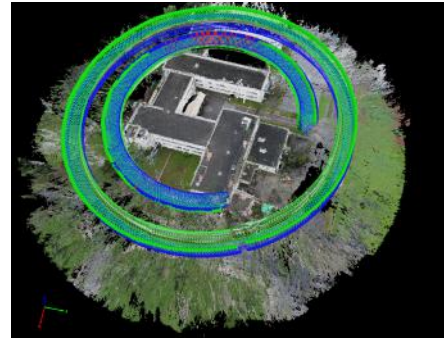


(a)

Figure A.2: Point cloud generated from RGB images in Pix4d.

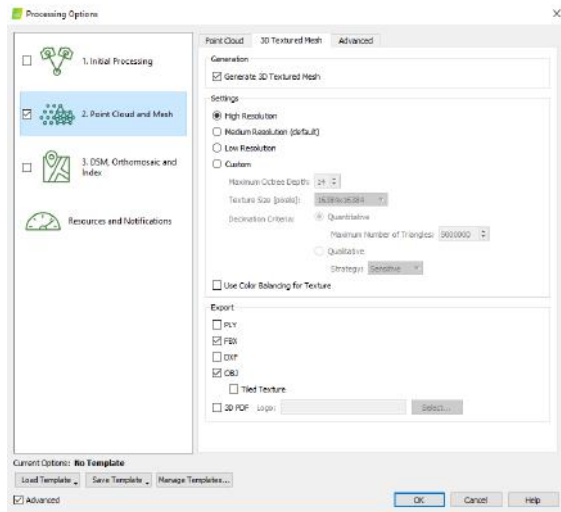


(a)

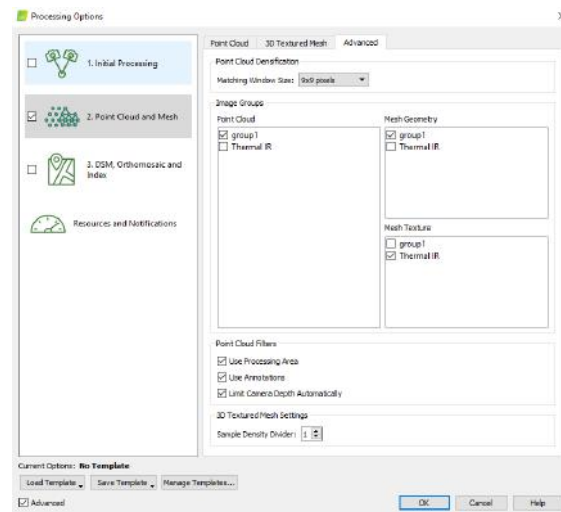


(b)

Figure A.3: (a) Manual Tie Points (MTPs) for the RGB project shown as green arrows. (b) The merged point cloud with RGB (outer circle) and Thermal (inner circle) drone flights overlaid.

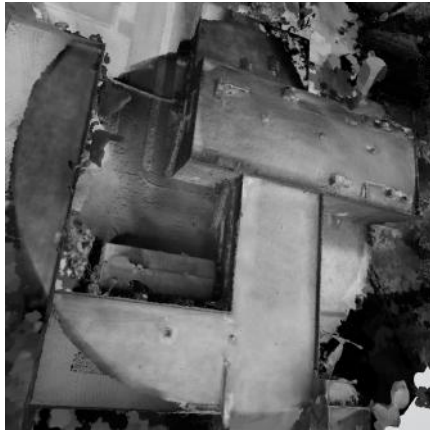


(a)



(b)

Figure A.4: Settings used to generate the textured 3D mesh.



(a)



(b)

Figure A.5: 3D Mesh textured with thermal imagery (a) Top view. (b) Side view with windows.

A.3 3D Model with Thermal Textures

Using the same RGB and thermal images, we also constructed a 3D mesh using Pix4D that was texture mapped with the thermal images. The geometry for the 3D mesh was created from the RGB images inside Pix4D, with the thermal texture simply projected on top. This was performed using a similar process to Section A.2 where two individual projects were created, one for the RGB images and one for the thermal, MTPs were added to both projects, and then the two projects were merged together. Once the two projects were merged, the 'Point Cloud and Mesh' step is run, with the settings shown in Figure A.4. A 3D mesh textured with thermal images generated as a result of this process is shown in Figure A.5.