3D Modeling of Interior Building Environments and Objects from Noisy Sensor Suites

By

Eric Lee Turner

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in
Engineering – Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Spring 2015

3D Modeling of Interior Building Environments and Objects from Noisy Sensor Suites

Abstract

3D Modeling of Interior Building Environments and
Objects from Noisy Sensor Suites

by

Eric Lee Turner

Doctor of Philosophy in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Avideh Zakhor, Chair

In this dissertation, we present several techniques used to automatically generate virtual models of indoor building environments. The interior environment of a building is scanned by a custom hardware system, which provides raw laser and camera sensor readings used to develop these models. Our modeling techniques can be separated into three categories: 2D floor plan models, simplified 2.5D extruded models, and fully complex and detailed 3D models. All models are produced automatically from the output data of a backpack-mounted ambulatory scanning system, which can scan multiple floors of a building efficiently. We can capture the entirety of a large building in only a few hours, including staircases and other hard to reach locations, which improves upon the state-of-the-art static scanning by several orders of magnitude. Our algorithms are capable of producing many kinds of 3D virtual building models while traversing through a GPS-denied environment, including point clouds, 2D floor plans, and 3D building models.

The novel contributions of this dissertation fall into three groups. First, we present multiple methods for producing 2D floor plans in a scalable fashion. We automatically partition these floor plans into separate building levels and separate rooms within each level. These floor plans can also be extruded into simplified 2.5D models of the building environment. Second, we present multiple techniques to produce complex 3D models of the scanned environment, capturing all observed detail. Third, we present several techniques that combine these two types of building models – simple and complex – to perform additional analysis of the building environment. Such analysis includes segmenting objects and furniture in the environment as separate models, reducing noise and artifacts within the models, and demonstrating novel visualization techniques. Additionally, we show several example datasets generated by the system in real-world environments, including buildings of over 40,000 square feet. We demonstrate how such building models are applicable in many fields of study, including architecture, building energy efficiency, virtual walk-throughs of buildings, indoor navigation, and augmented and virtual reality.

To Mom.

Contents

List of Figures

Acknowledgements

As soon as I saw University of California, Berkeley, I knew I would love it here. A campus cut with forested paths and narrow streams seemed straight out of a Calvin & Hobbes strip. As soon as I met the people here, I immediately found that the intellectual adventure matched the aesthetics.

I first want to thank my advisor, Professor Avideh Zakhor. The potential she sees in her students becomes manifest with each challenge she places before them. My understanding of the field and of academics in general has become so much richer thanks to her guidance. I also would like to thank Jonathan Shewchuk, Kyle Steinfeld, and Carlo Séquin, who served on my qualifying exam and dissertation committees. These are some of the best professors I've ever met, and have provided me with excellent guidance both in the classroom and in my committee.

This research was conducted with government support under and awarded by Department of Defense, Air Force Office of Scientific Research, National Defense Science and Engineering Graduate (NDSEG) Fellowship, 32 CFR 168a. Support was also given by the Army Research Office (ARO), contract W911NF-11-1-0088, and by ARPA-E under contract DE-AR0000331.

I also want to extend my deep gratitude to all my colleagues in the Video and Image Processing Lab. John Kua, Michael Krishnan, Rick Garcia, Shangliang Jiang, Nicholas Corso, Shicong Yang, Richard Zhang, and Joe Menke have all lent me guidance and helped me through. I especially want to mention Nicholas, who ensured that I took a daily constitutional around campus rather than hiding away in our lab.

Additionally, I want to thank my family. Even though I moved to the other side of the country, they have always been supportive. My Dad, David Turner, has not only been my role model of an engineer, but truly comforting at every step of my way through school. Lastly, I want to thank Samantha Shropshire, my girlfriend. For putting up with my working at odd hours, for supporting me, and for bouncing ideas off of, her patience knows no bounds. Sam, without you, I would never have survived graduate school.

Chapter 1

Introduction

Laser scanning technology is becoming a vital component of building construction and maintenance. During initial construction, laser scanning can be used to record the as-built locations of HVAC and plumbing systems before drywall is installed. In existing finished buildings, blueprints are often outdated or missing, especially after several remodelings. Laser scans can be used to generate building models describing the current architecture. Triangular meshes allow for the efficient representation of the scanned geometry. In addition to being useful in the fields of architecture, civil engineering, and construction, these models can be directly applied to virtual walk-throughs of environments, gaming entertainment, augmented reality, indoor navigation, and energy simulation analysis. These applications rely on the accuracy of a model as well as its compact representation.

Generating an accurate model of indoor environments is an emerging challenge in the fields of architecture and construction for the purposes of verifying as-built compliance to engineering designs [1, 2]. This task is made more challenging by the GPS-deprived nature of these environments [3]. Another application that requires an exported mesh to retain as much detail as possible is historical preservation via a virtual reality model [4, 5]. Building energy efficiency simulations can use watertight meshes of the environment to estimate airflow and heat distribution [6]. These simulations require simplified meshes as input, since finite element models are difficult to scale. It is also important to be able to generate an immersive visualization and walk-through of the environment for these applications, so experts can remotely inspect the scanned environment via tele-presence, a task that currently requires expensive travel and on-site visits. Different applications require models of different complexities, both with and without furniture geometry. The modeling approaches detailed in this report are useful for both types of applications, as shown in Figure 1.1. Figure 1.1a is a photograph of the scanned area: the hallways of Cory Hall on UC Berkeley campus, encompassing over 10,000 square feet of scanned area. Figure 1.1d represents the captured 3D point cloud of this area, which is discussed further in Chapter 2. Figures 1.1b and 1.1c show a high-detail 3D mesh of 2.7 million triangles generated using the algorithm in Chapter 4, with and without texturing, respectively. Figures 1.1e and 1.1f show a low-detail model of 2,644 triangles generated using the approach in Chapter 3, with and without texturing [7].

One of the primary challenges of indoor modeling is the sheer size of the input point clouds. Scans of single floors of buildings produce point clouds that contain hundreds of millions of points, often larger than the physical memory in a personal computer. Man-made

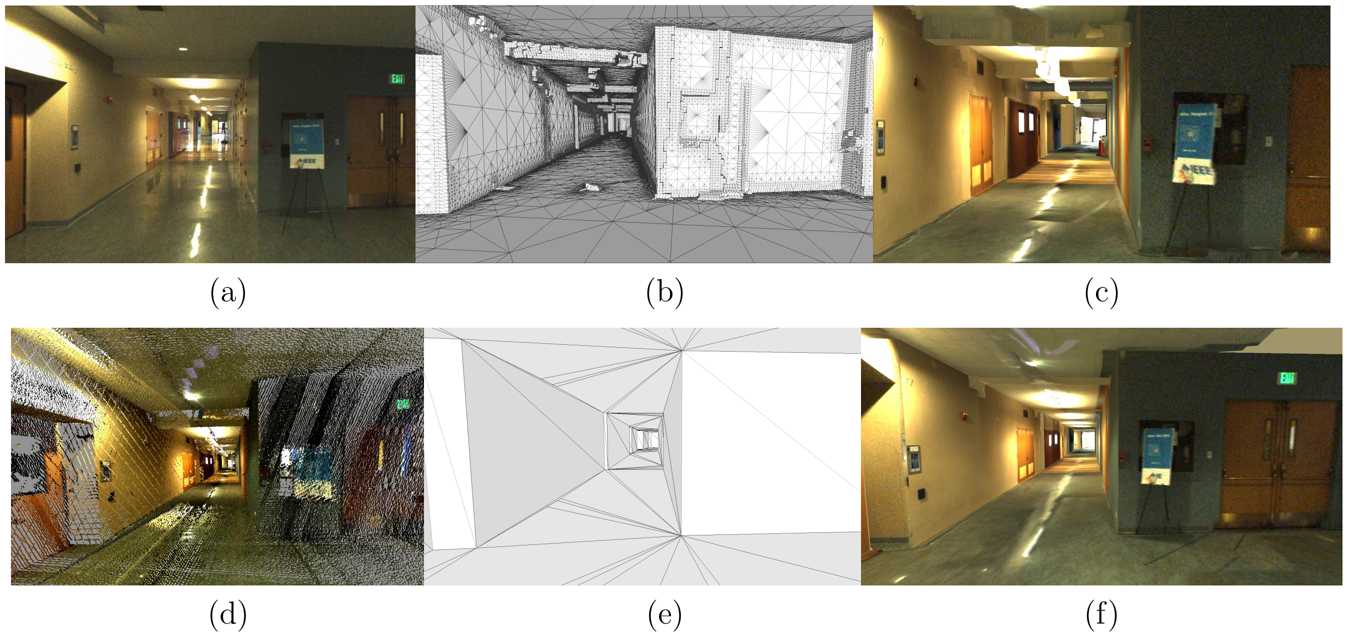


Figure 1.1: Models generated with the techniques described in this dissertation: (a) photograph of scanned area, academic building; (b) surface carving model of this area; (c) surface carving model with texturing; (d) point cloud of captured scans; (e) extruded floor plan model of area; (f) extruded floor plan with texturing.

geometry is typically composed of planar regions and sharp corners, but many conventional surface reconstruction schemes assume a certain degree of smoothness and produce rounded or blobby output if applied to these models [5,8–12]. In addition to large flat regions, building interiors also contain many small details, such as furniture. A surface reconstruction scheme must be able to represent the large surfaces in a building with an efficient number of elements and preserve their sharp features. The fine details of furniture are useful for some applications whereas others require furniture to be removed. In this dissertation, we discuss existing modeling techniques that remove fine details and those that preserve these details. The output of both of these modeling techniques can be texture-mapped with captured camera imagery [13]. An example of texture-mapping applied to these two modeling processes is shown in Figures 1.1c and 1.1f.

1.1 Surface Reconstruction

This section describes background techniques for surface reconstruction from input point cloud data. These techniques are useful for generating meshes of objects scanned with static table-top scanning systems.

Powercrust is a volumetric surface reconstruction approach that yields watertight models from point cloud data [8]. It computes the interior medial axis of a shape using the Voronoi diagram of the input scan points. By taking the union of the inner polar balls, the elements of the Delaunay triangulation that form the interior of the shape can be found, and are used to approximate the modeled volume. While this approach is popular, it is sensitive to noise in the input scans. Fortunately, other techniques have improved on this methodology to account for noisy input.

An approach to account for the noise in the Powercrust algorithm is to remove sufficiently small polar balls from consideration, which are likely to be due to input noise [14]. Another approach is to perform spectral clustering on the tetrahedralization of the points, as demonstrated by Kolluri et al. [12]. This approach, dubbed Eigencrust, provides substantially improved robustness to noise. An important consideration, though, is that the noise models tested by Kolluri et al. are randomly positioned outlier points and randomly perturbed input points. Both of these approaches assume that the randomness is independent and identically distributed from point-to-point. The model of *i.i.d.* noise does not fit the errors caused by mis-registration of scans, which is the most common source of noise from mobile scanning systems. These methods also assume that the point clouds are modeling smooth and continuous surfaces, which is not the case in building modeling. These algorithms may also require a global optimization step [12]. While advances have been made to perform these computations in an efficient and out-of-core manner [15, 16], the resulting models are too large to be practical for graphical or simulation applications.

Another technique used to produce watertight models is to define implicit functions on the scanned volume, and use an isosurface of this volume as the exported mesh. Once an implicit function is defined, techniques such as Marching Cubes or Dual Contouring can be used to mesh the surface [17, 18]. A popular method for defining this implicit function is to use signed distance fields [19]. Other popular approaches such as Poisson Surface Reconstruction allow the user to specify a resolution parameter for the generation of more compact models [11]. These schemes guarantee watertightness by using an implicit surface to model the point cloud [20]. While these approaches can be applied to large models using distributed computing techniques [21, 22], they are not suitable for modeling man-made architecture. The output models of these methods lack sharp features because they generate implicit surfaces using Gaussian basis functions.

Additionally, many common triangulation schemes for implicit surfaces generate uniform elements [17, 18], which are undesirable for large, flat surfaces that can be modeled just as accurately with fewer elements. If these approaches are used on a discretized voxel grid, undesirable artifacts of the discretization are preserved, requiring the final mesh to be smoothed, thus reducing accuracy [5]. Algorithms that adaptively mesh an isosurface or simplify an existing mesh rely on the local feature size of a model [10, 23–25]. Models with flat regions or sharp corners where the curvature approaches zero or infinity can become degenerate or have poor quality. Since building models are composed almost entirely of such areas, these techniques are not appropriate.

Models of building interiors are rich with flat surfaces and right angles. This prior knowledge supports the use of primitives that have these same aspects. Examples include voxel and octree structures, which are used in many carving techniques [5, 9, 26–28]. The advantage to such approaches is that they are robust to noise and registration errors in the input point cloud. One of the challenges with voxel representations is memory and computational intensity, becoming tractable only when performed in a distributed or parallel fashion [29]. Some voxel carving approaches can also inadvertently remove small details, if the voxel size is uniform and set to be larger than the feature length of the desired

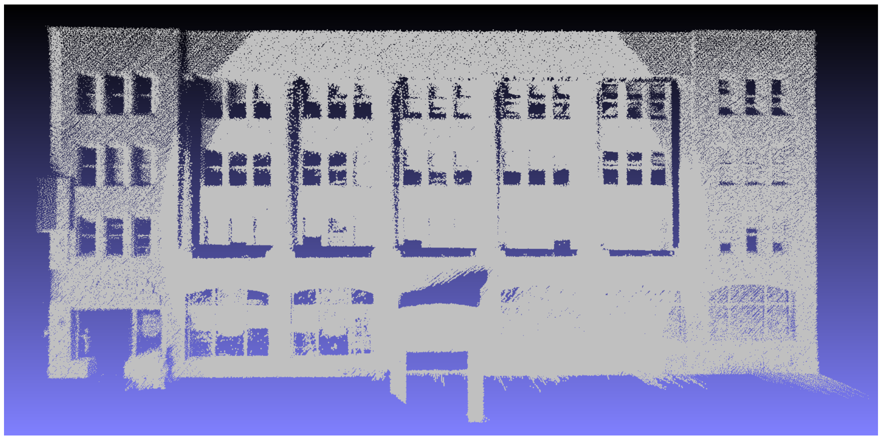


Figure 1.2: An example point cloud of a building façade. Note gaps near windows, ledges, and columns due to line-of-sight occlusions of the scanning system.

geometry [5]. In Chapter 4, we modify voxel carving to address these issues and introduce memory-efficient data structures that produce models that preserve fine details with an efficient number of elements.

Traditional surface reconstruction techniques, as described above, are written for and tested with common point cloud test sets such as the Stanford Bunny or Dragon. These models are scanned with precision 3D scanners that yield relatively low noise and minimal mis-registration when compared to scans of larger areas such as buildings. These approaches are also designed to model isolated objects, with the scanner moving around the object in a controlled environment. In contrast, the goal of this thesis is to model buildings, which are at a much larger scale and produce much higher rates of noise and mis-registration, with the scanners moved through the interior of the environment in a less controlled setting. While watertight volumetric processing is still useful for our applications, new techniques must be developed compared to isolated objects.

1.2 Outdoor Building Scanning

In the previous section, we discussed general surface reconstruction techniques. In 3D reconstruction of building environments, separate techniques are required for the subfields of indoor and outdoor building modeling. Most of this dissertation is specific to indoor building modeling, however many related techniques are used for outdoor modeling as well.

The current method of outdoor building scanning requires an acquisition vehicle to drive down a street while taking Light Detection And Ranging (LiDAR) scans and panoramic photographs of the surround area [30–33]. In many applications, we wish to generate a triangulated geometry for a collected point cloud. This geometry needs to be accurate to the original architecture.

For buildings of an appreciable height, the street-level LiDAR scans are taken at a very acute angle with respect to the surface of the building. Any protrusions from the building cause shadows in the LiDAR and therefore the building is not collected in its entirety, as shown in Figure 1.3. A typical solution would be to gather scans from multiple angles

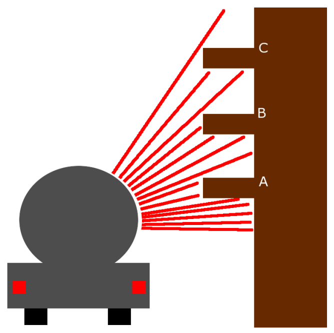


Figure 1.3: Locations on the building A, B, C are occluded from the LiDAR collector on the vehicle.

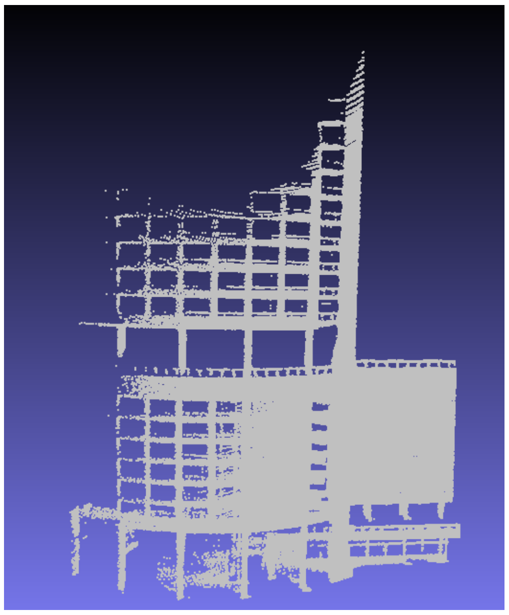
[19, 34]. For the urban reconstruction problem, this tactic cannot be used since the scanner is restricted to the street.

Previous attempts to model buildings have assumed that architecture takes a simple polyhedron geometry. This approach ignores minor details of a building façade, such as windows and ledges when applied on a large scale [31, 35]. Our goal is to preserve as much detail as possible in the reconstructed model geometry. Any interpolation must preserve the sharp edges of corners that occur within these holes. This requirement diverges from the typical assumptions of most surface completion methods, which usually generate smoothed surfaces [11, 36]. While developing 3D models with sharp features is a well-explored topic, most current approaches require a sufficient density of points near regions of high curvature, which would not be applicable here [37, 38].

Our goal is to generate sharp, axis-aligned, planar features in all locations, including those where the point cloud is sparse or not sampled at all. Our algorithm generates geometry for any holes in the point cloud while preserving flat planes and sharp corners, which are common in modern architecture [39]. First we determine the dominant plane of this façade to reconstruct a mesh of a building face. Then by treating the original points as a height map on this plane, we uniformly resample these heights over the entire surface using Moving Least-Squares (MLS) smoothing to mitigate noise in areas of high sampling, interpolate the areas corresponding to gaps in the point cloud, and guarantee the uniformity of the resulting geometry [40].

This method generates flat strips of extrapolated façade. If the building curves outward or has other non-planar characteristics, these strips conform to the shape of the building, as shown in Figure 1.4. Once a geometry has been processed, a texture is applied using panoramic photographs collected from the same location as the LiDAR. Since these images are projected onto the geometry, any irregularities in the geometry produce dramatic disturbances in texturing. Examples of this texturing process are shown in Figure 1.5.

While the techniques used for surface reconstruction of exterior building models are different from those used for interior models, both methods require the efficient processing of large amounts of scan data in the form of point clouds. Such point clouds may represent noisy or inconsistent geometry, due to being acquired via a mobile scanning system.


(a)

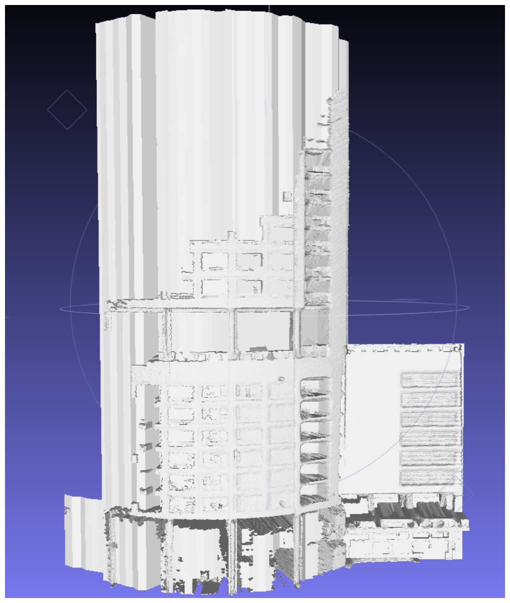

(b)

Figure 1.4: The curvature of the building is preserved when extrapolated; (a) building point cloud; (b) MLS triangulation.

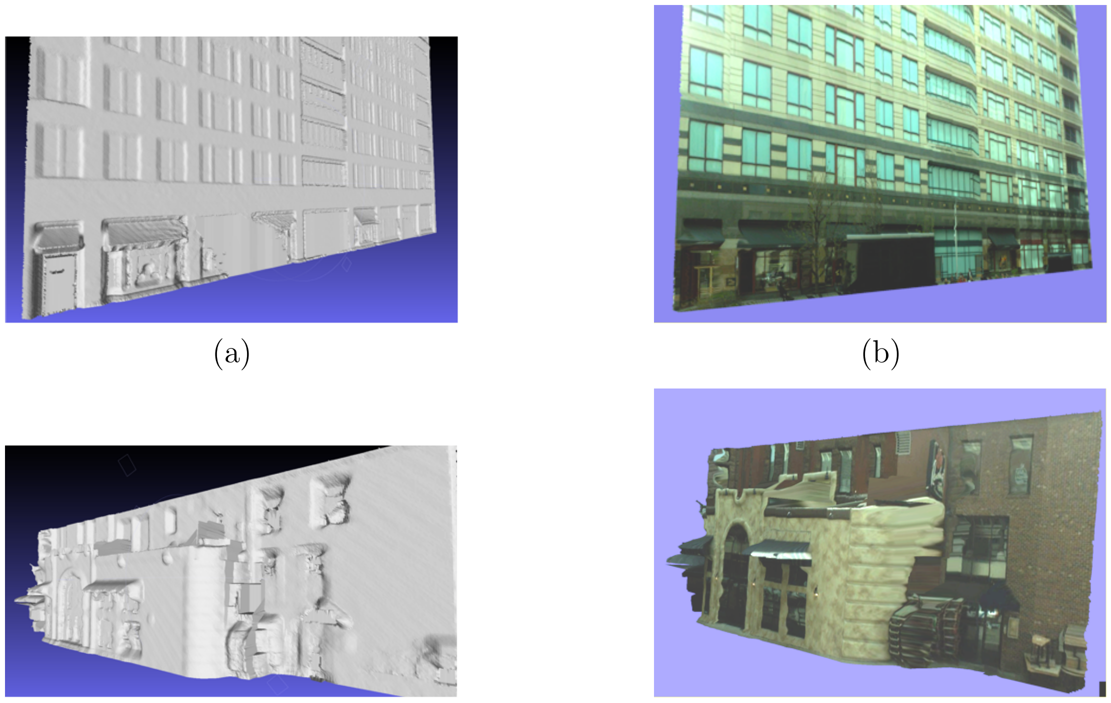

(a) (b) (c) (d)

Figure 1.5: Results of texturing geometry after sharp hole-filling; (a) sampling of flat façade; (b) with texture; (c) sampling of uneven façade; (d) with texture.

1.3 Indoor Building Scanning Systems

For the remainder of this dissertation, we discuss prior work on reconstruction of models of the interiors of buildings. In the architecture community, there is a continuing push for the use of a consolidated Building Information Model file format [41]. Such models need to be semantically rich for all phases of building development, including architectural, structural, mechanical, electrical, plumbing, etc. Traditionally, each separate aspect needed to be remodeled from scratch, by hand, which introduces errors into the modeling process.

It is important to distinguish between models that are as-designed compared to as-built. Each phase of building development produces a design, but it is still important to compare this design to the as-is nature of the building. Scanning of building environments is an important technology to be able to compare the result of construction to what was desired. With current technology, there is still a fair amount of manual intervention that is required to convert a 3D scan of a building into a suitable BIM file that can be used for such a comparison.

Traditional industry-standard building scanning uses static scanners mounted on tripods that are moved from area to area in the building [42–44]. This scanning process is labor intensive and slow, but produces highly accurate point clouds after stitching. Many mobile systems have been introduced to automate indoor scanning. Due to the cost of full 3D laser range finders, the majority of indoor modeling systems use 2D LiDAR scanners. Wheeled platforms that carry scanning equipment and are manually pushed through the environment are popular [5,45]. The mobility of such systems is limited, since they are unable to traverse rough terrain or stairs easily. Other researchers have investigated mounting laser range finders on unmanned aerial vehicles [46–48]. Such platforms are agile in that they can scan difficult-to-reach areas, but are limited in scanning duration by short battery life, preventing scalable scanning.

In this dissertation, we focus on ambulatory scanning platforms, where the sensor suite is carried by a human operator as the operator moves through the building environment [4,49,50]. These systems allow for rapid data acquisition and can actively scan for several hours at a time. They use 2D LiDAR scanners due to the cost and weight of full 3D laser range finders. The captured scans are used both to reconstruct the geometry of the environment and to localize the system in the environment over time, in Simultaneous Localization and Mapping (SLAM). The majority of datasets shown in this paper have been generated by a backpack-mounted system that uses 2D LiDAR scanners to estimate the 3D path of the system over time as well as multiple scanners to generate geometry for the environment [51–54]. This system also has multiple cameras collecting imagery during the data acquisition process, which allows for scanned points to be colored or for generated meshes to be textured with realistic imagery [13].

1.4 Modeling of Indoor Building Environments

The primary topic of this dissertation regards surface reconstruction of scan data of indoor building environments, regardless of what hardware systems are used to collect building scan data. Techniques used to model different aspects of building geometry from captured scans can be classified into three main categories: Floor-Plan Generation, Simplified 3D Modeling, and Detailed 3D Modeling. Floor-plan generation focuses on estimating 2D positions of walls in the building. Simplified 3D modeling similarly focuses on 3D modeling only the permanent features of a building: floors, walls, and ceilings. Detailed 3D modeling focuses on modeling all aspects of the scanned geometry, including fine details such as furniture or objects observed in the building. Note that most existing approaches are developed for static scans of buildings, which have very low noise and capture high detail. The focus of this dissertation is to develop modeling techniques for mobile scanning systems, which are much more likely to suffer from mis-registration noise or missing geometry.

In this section, we discuss existing techniques to generate each of these types of building models. This dissertation also describes novel work we contributed in each of these areas, in Chapters 3, 4, and 5.

1.4.1 Floor-Plan Modeling

Interior scanning is traditionally used for robotic navigation in indoor environments. The rough locations of walls are necessary to avoid collisions. Generating floor plans for modeling, however, requires a much higher degree of precision [55]. Given these antecedent studies, techniques have been developed using a single horizontal scanner collecting a 2D cross-section of the environment [56]. Constructing full 3D scans allows for more sophisticated means of identifying walls from other obstacles in a building. In this scenario, one can compute a top-down 2D histogram of point densities across the xy-plane [55]. Areas with high density are considered likely to be wall locations. While clutter is mitigated by being less represented in the histogram than walls, no direct measures are taken to remove outlier samples before line-fitting. Further, each story of a building must be processed separately.

Previous approaches to floor plan modeling typically assume walls are well-fitted by straight line segments and whether these fitted models are watertight is not guaranteed [55–57]. Many architectural designs incorporate curved features, which would not be modeled accurately by these approaches [58, 59]. The absence of a watertightness guarantee requires extensive post-processing to remove disconnected and outlier segments that are interpreted as noise.

Floor-plan modeling techniques are based on sampling positional information of walls within the environment captured by the scans, then using these wall samples to generate a plan composed of line segments or polygons. Weiss et al. use a cart-based system with a horizontal laser scanner [56]. The output scan points are exactly the sample positions of the walls. They find lines in this scan map with a Hough Transform, indicating walls in the environment. Okorn et al. employ a similar method, but their input scans represent a full 3D point cloud [55]. The wall sample positions are found by computing a top-down histogram of the input points, and areas of high density are classified as vertical surfaces. The approaches we discuss in Chapter 3 employ a similar top-down histogram.

Another important feature of analysis in building floor plans is automatic room partitioning. The first publication on this topic is our approach, which is expanded in Section 3.3.2 [60]. A subsequent method that also performs room partition is Mura et al.'s paper. This method takes a different approach to estimating wall positions [61]. They perform region growing in the 3D point cloud to find planar regions, which are projected into

2D and treated as potential wall candidates. A cell complex is then built to volumetrically identify separate rooms in the model. There are also methods that take a floor plan as input, and use this information to generate a 3D model of the environment by extruding the defined wall information [62, 63]. This type of modeling yields aesthetically pleasing results with well-defined floors, walls, and ceilings.

1.4.2 Simplified 3D Modeling

Since building features are almost entirely planar, a popular approach is to explicitly fit planar elements to the input point clouds. This assumption allows for plane-fitting to be performed on the input point cloud, either by a histogram approach or random consensus [43, 44]. Such approaches do not guarantee watertightness of the resulting mesh and can require substantial post-processing. Sanchez and Zakhor use PCA plane-fitting to find floors, walls, and ceilings explicitly in the point cloud, as well as explicitly fitting staircases [64]. Since this method is applied to ambulatory data, it produces missing components, holes, and double-surfacing due to mis-registration. Xiong et al. also perform plane-fitting in a similar fashion, but also analyze the computed planes for the locations of windows and doors, which are represented as holes in the surface [2]. Adan et al. find planes by first generating a floor-plan, then extrude the floor-plan into a 3D model [43]. One limitation with these methods is that they are not necessarily watertight, though floor plan extrusion can be done in a watertight fashion [60, 65, 66]. These approaches allow for accurate wall geometry and reduce the complexity of the output model. Extruded floor plans also allow for models to explicitly define floors, walls, and ceiling surfaces. Being able to make assumptions about the planarity of an environment has been used successfully to model only the major features of scanned objects even in the presence of high noise [67].

Other approaches have focused on volumetric processing to ensure watertightness when computing simplified 3D models. Xiao et al. find horizontal cross-sections of the building, forming a sequence of 2D CSG models that are then stacked together and simplified [68]. While this approach does lead to aesthetically pleasing models, it assumes Manhattan-world models, which leads to topological errors if an insufficient number of cross-sections are recovered. Oseau et al. use a voxelization approach, with a follow-up graph-cut step to remove small details in the environment, leaving only floors, walls, and ceilings [69]. This optimization step, however, can also cause significant deformations in the final geometry depending on the input parameters.

1.4.3 Detailed 3D Modeling

One of the methods used to preserve the detail of furniture in a building scan is to explicitly search for and classify furniture models in the scan. These techniques attempt to find locations in the input scans that match best with a stored database of known furniture. The pre-existing model of the recognized piece of furniture is then oriented in the output model. Nan et al. employ this technique, with explicit classification of chairs and tables [70]. Kim et al. also use this method, using a larger library of objects and operating on noisier scans [71]. They also discuss how search and classification can allow for change detection across scans of the same area taken at different times. A major downside of this method is that the classification is only as good as the database. Objects that are in unexpected orientations or are not in the database are misclassified. For instance, a sideways chair is misclassified as a table. One major benefit of this approach is the ability to model the objects independently of the room itself, as discussed later in Section 5.6.

Recently, object detection methods from indoor scans have been proposed without the use of a training dataset [72, 73]. The work of Mattausch et al. segments point clouds using a bottom-up approach to fit rectangular patches on to the scans, then finds clusters of patches that are repeated often. Even though this approach can be applied to large datasets, it assumes very basic building geometry to segment objects. Additionally, it only detects objects of certain complexity, is unable to detect very small objects, and requires objects to be repeated often in the environment. We expand on this work by segmenting objects volumetrically, rather than in the point cloud domain.

There are also methods that capture fine detail of buildings by attempting to be as accurate to the input point cloud as possible. Holenstein et al. generate a space-carving model that voxelizes the scanned environment, labeling any voxels intersected by a scan-line to be interior [5]. The boundary of the interior voxels is meshed with Marching Cubes. The advantage of this approach is an increased robustness to mis-registration errors, but the downside is that over-carving can cause the loss of fine detail. Lastly, a popular approach to detailed modeling of environments is Kinect Fusion [74, 75]. It allows for both meshing and tracking, but is limited in that it cannot handle large areas, and the actual meshing approach is a minor extension of signed distance fields [19].

1.5 Contributions and Organization of Dissertation

In this dissertation, we present several techniques to automatically generate virtual models of indoor building environments. The interior environment of a building is scanned using a custom hardware system, which provides a set of data products used to develop these models. Our modeling techniques can be separated into three categories: 2D floor plan models, 2.5D extruded simplified models, and dense 3D models. All approaches are produced automatically from the output data of our scanning system. These methods are intended to be agnostic to the specific hardware used for scanning and in many cases have been applied to scans taken with other hardware systems. Unless stated otherwise, however, any examples shown in this dissertation were generated from scans acquired using an ambulatory backpack hardware system.

An overview of how data products are used by our techniques is shown in Figure 1.6. All methods we discuss in this dissertation use laser range scans as input, either in the form of registered point clouds or as raw scan files. The output models of our techniques are exported in open-source file formats. Figures in this dissertation showing our output models are generated by the rendering program MeshLab [76].

First, we discuss the specifics of the hardware system used throughout this dissertation in Chapter 2. This system is a collection of laser scanners and other sensors, worn as a backpack by a human operator. The operator traverses the environment at normal walking speed, allowing the system to collect data about the interior area. We discuss specifics on the mechanics of the backpack system and special considerations needed to ensure accuracy

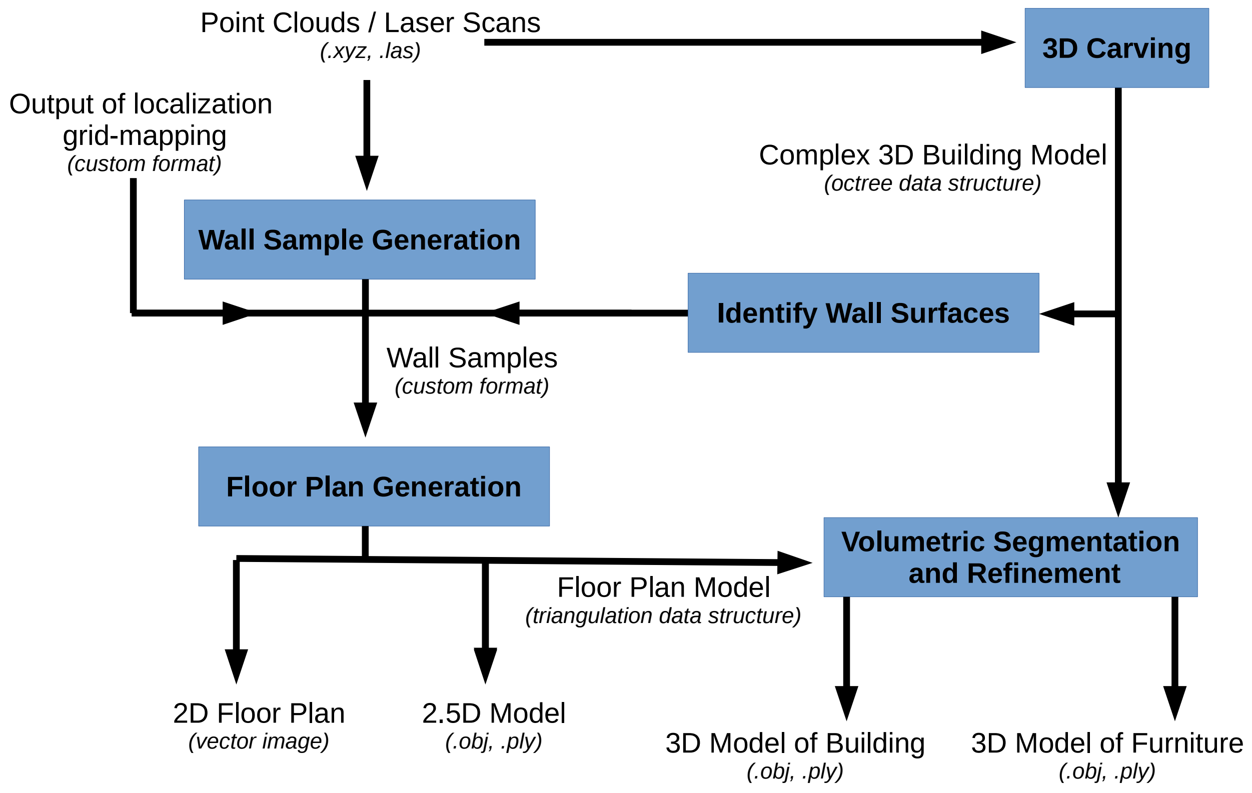


Figure 1.6: This flow chart shows file formats and data structures used to pass data products between algorithms described in this dissertation. These techniques are used to generate 2D floor plans, 2.5D simplified models, and fully 3D complex models.

of the resulting scans. We also discuss preprocessing steps required once a data acquisition is performed. Specifically, these steps detail how the system is localized while traversing through a GPS-denied environment and how we ensure a consistent global coordinate system. This step needs to be performed before any of our method can be applied to a dataset. Once the localization is performed, we generate a point cloud of the environment. This point cloud can be colored based on the camera imagery. Additionally, these scans are used to determine the number of building levels covered during the acquisition, and the partitioning of those levels.

In Chapter 3, we detail the techniques developed to generate 2D floor plans of the interior building environment. These techniques use the 3D scan information to generate a consolidated 2D estimate of wall positions. These wall position estimates are used to develop a volumetric floor plan of each level in the building environment. Each wall position estimate, or "wall sample", contains an average (x,y) position, height information, and indices of sensor poses that can observe it. We show multiple floor plan generation techniques, including methods to automatically detect and classify separate rooms in the environment. Once a

2D floor plan is produced, its geometry can be extruded into a simplified 2.5D model, which includes height information. This process is depicted in Figure 1.6 by the “Wall Sample Generation” and “Floor Plan Generation” boxes.

In Chapter 4, we describe several methods for generating complex, fully-3D models of building environments. Such models include not only building elements, but also furniture and other objects within the building environment. These models can be processed to explicitly fit planar elements to surfaces in the environment or to preserve the curvature detail observed. Planar surfaces are useful since buildings are typically dominated by such features, and because they allow for simplified output geometry. This process is shown in Figure 1.6 by the “3D Carving” box.

In Chapter 5, we combine these different techniques to improve the richness of the output models. We first compare the complexity of the different model types. We then discuss how these two types of methods – 2D floor plan generation and 3D dense modeling – can be combined to provide additional analysis on building modeling, such as segmenting furniture geometry or removing scanning artifacts. This combination of techniques can be used to not only augment each individual model output to improve accuracy, but also provide new information about the environment. This first half of this chapter discusses how the 2D floor plan generation methods can be improved by the 3D dense modeling results, and the second half of the chapter iterates how the 3D dense modeling is improved by incorporating the 2D floor plan output. This process is depicted in Figure 1.6 by the “Identify Wall Surfaces” and “Volumetric Segmentation and Refinement” boxes.

In Chapter 6, we show the applications of these techniques, and how building models at different granularities are required based on how the model is to be used. We show several examples of use-cases for these modeling techniques.

Lastly, in Chapter 7, we offer concluding remarks and directions for future work.

Chapter 2

Preliminaries

The surface reconstruction algorithms described in this dissertation can be applied to any indoor scanning data. For convenience, unless otherwise stated, we use the ambulatory backpack system for all examples shown in this document [51, 52]. As the human operator walks through the building environment, at normal speed, this system collects data from a variety of sensors on-board. These data products are logged and then processed off-line. Localization software can accurately estimate the trajectory of the system over time and localize the system [54]. This procedure allows one to rapidly move through a large environment, spending only a few seconds in each room yet capturing full geometry information.

Once the prior art of localization is applied to the scan data from the hardware system, some data products can be generated with minimal processing, such as 3D point cloud data. Unlike most geometry products discussed, a point cloud contains no topology information. Instead, a point cloud is a concatenation of the raw laser range data, transformed into the global coordinate frame. In this chapter, we present methods for colorization of these point clouds from camera imagery and partitioning of point cloud data via automatically segmented building levels.

2.1 Hardware System

The Video and Image Processing Lab at UC Berkeley has developed a custom ambulatory backpack-worn scanning system. This system contains several 2D laser scanners, cameras, and an Internal Measurement Unit (IMU). It has also been augmented with Wi-Fi antennas and infrared cameras. The Wi-Fi antennas are used to record signal-strength of nearby access points, which is useful for indoor localization techniques [77]. The infrared cameras are useful for energy-efficiency analysis of building environments. These data products are discussed further in Chapter 6.

Our geometry reconstruction algorithms use the retrieved laser range data as input. These scans are taken with time-of-flight laser scanners that capture 3D geometry of the environment. Multiple 2D laser scanners are mounted at different orientations to capture all observed geometry in the environment. Figure 2.1a shows an annotation of the sensors on the backpack system.

The system uses Hokuyo UTM-30LX sensors, whose intrinsic noise characterization is

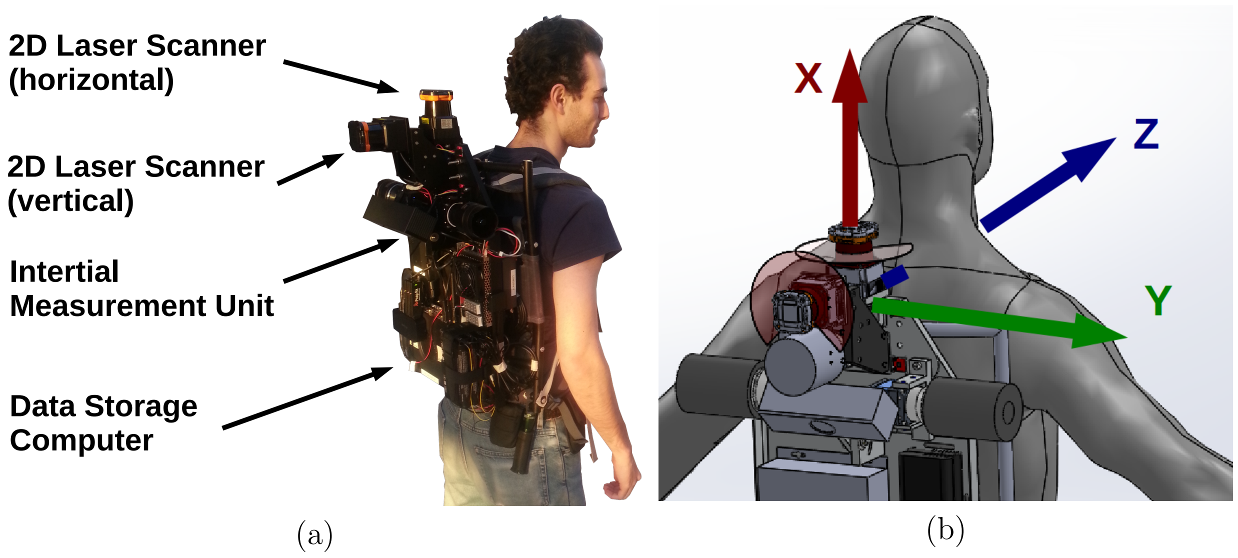

2D Laser Scanner (horizontal)
2D Laser Scanner (vertical)
Intertial Measurement Unit
Data Storage Computer
(a)
X
Z
Y
(b)

Figure 2.1: Annotated scanning hardware system, worn as a backpack. The operator walks through the indoor building area as the system scans the surroundings. Our method combines the LiDAR and inertial measurements to form a virtual 3D model for the observed space: (a) the system worn by operator; (b) the common coordinate frame of backpack system.

given in [78, 79]. This noise contributes on the order of 1 to 2 centimeters to the standard deviation of the positional estimate of scan points. This uncertainty value increases as the range of the point increases, with a maximum range of 30 meters. Since this system is mobile and ambulatory, the acquired point clouds have more noise than traditional static scanners due to natural variability in human gait. As such, our geometry reconstruction approaches need to be robust to motion induced scan-level noise and its associated artifacts.

The primary advantage of a backpack system is speed of acquisition. The backpack uses 2D Time-Of-Flight (TOF) laser scanners, which means we have sparser scan information per frame and traverse each area of the environment much faster than when scanning the equivalent area with a Kinect or Google Tango. Since the capture system produces much sparser point clouds than Kinect-based scanning, it is not feasible to employ the same averaging techniques that allow for high-quality Kinect Fusion [74]. The advantage of these 2D TOF sensors is that they are less noisy, have a longer range, and a wider field of view. The result is that much larger environments can be covered in significantly less time when using these sensors in a sweeping motion [49]. A backpack-mounted system allows an unskilled operator to rapidly walk through an environment to acquire data.

Since the hardware system contains may separate sensors, it is necessary to create a common system coordinate frame, as shown in Figure 2.1b. As described by Chen et al., each sensor has an independent coordinate frame, but they are extrinsically calibrated against one another to establish the rotation and translation between any given sensor and the common coordinate frame [52]. Once this extrinsic calibration process is performed, the orientation of any given sensor can be determined with respect to the common system coordinate frame. As discussed in Section 2.2, after a data collection the orientation of the system is determined at each timestep within a model coordinate frame. The combination of the transform generated

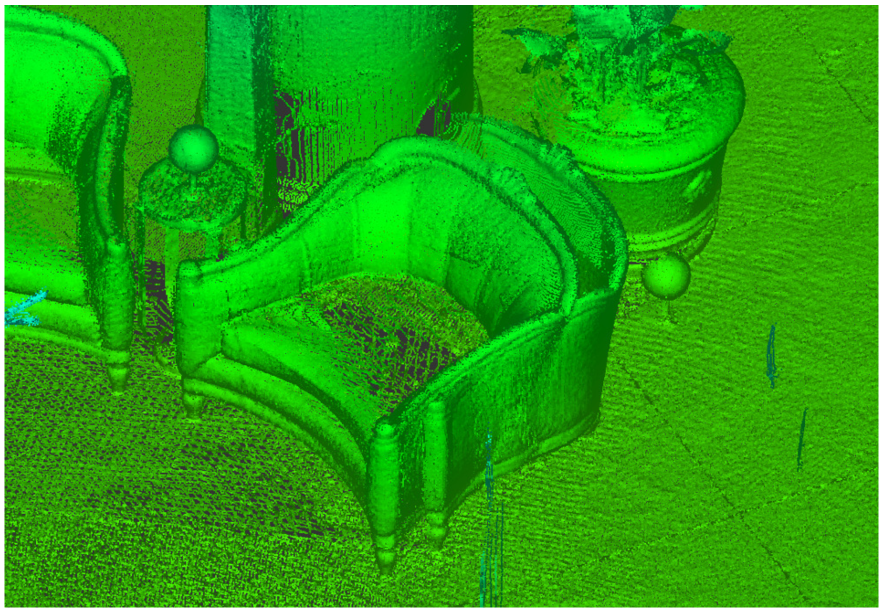


Figure 2.2: An example of mis-registration or "double-surfacing" in a point cloud. The chair shown in the figure was scanned at two different times, causing a duplicate chair to be shown at a slight offset.

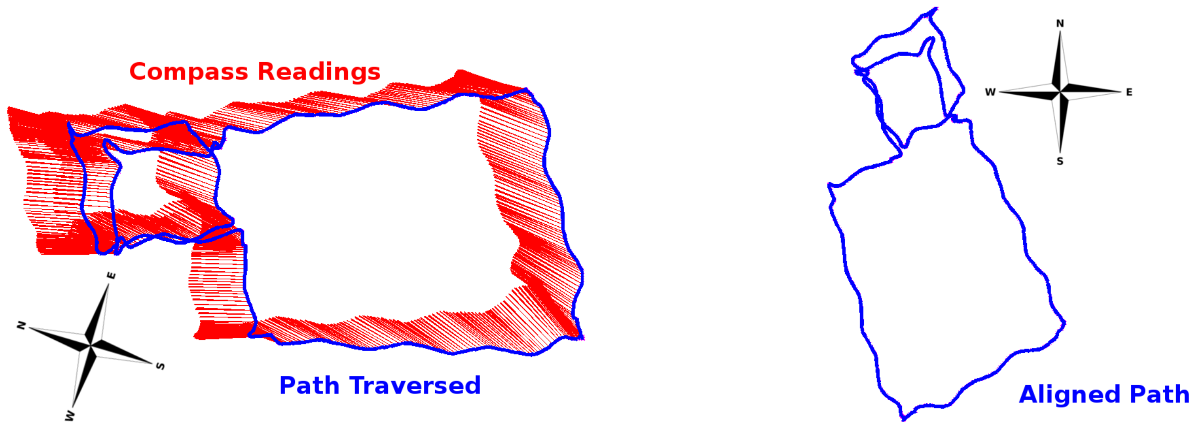
by extrinsic sensor calibration and the localization process allows for sensor readings to be placed in model coordinates.

2.2 Localization

The backpack hardware system includes localization procedures developed to track the moment of the operator through the indoor environments [51–54]. Once the localization procedure is complete, the transformation between the backpack's coordinate frame and the model coordinate frame is computed for each timestamp of operation. In order to perform geometry reconstruction, however, the positions of each sensor must also be tracked. The internal coordinate frame of the sensor, which is specific to the make and model of the sensor hardware, can be converted to the system common coordinate frame based on the extrinsic calibration of each sensor's rigid position and orientation on the backpack system, which is given by the coordinate frame shown in Figure 2.1b. With these transformations, any sensor reading can be placed in global coordinates.

For the rest of this dissertation, we refer to these transformations using the following symbols. The rigid affine transformation for a given sensor s is represented by the 3×3 rotation matrix $R_{s \to c}$ and the 3×1 translation vector $T_{s \to c}$. At a given time t, the affine transformation from the system common coordinates to the model coordinates is given by the 3×3 rotation matrix $R_{c \to m}(t)$ and the 3×1 translation vector $T_{c \to m}(t)$.

Since the localization procedure is performed by discrete integration, the expected error in the reported position and orientation of the system at any pose is higher than in competing technologies, such as static scan systems [51–53]. Static systems often utilize markers or



(a) (b)

Figure 2.3: An example of aligning model coordinates to magnetic north: (a) individual compass estimates of North (red) are compiled from each pose of the path (blue) and represented in world coordinates; (b) the model coordinates are transformed so that the mean estimate of magnetic North across all readings is oriented in the $+Y$-direction.

control points within the environment to ensure manual alignment of scans, allowing for positional accuracy of within 1 mm [32,33]. By contrast, the mean positional accuracy of the output poses in the backpack localization procedure is typically 10 cm [54]. Additionally, the errors in successive poses are highly correlated. Typically a local set of scan points captured within a few seconds of one another are highly accurate, yet if the same features in the environment are scanned twice with a long interval between scans, then the scans are likely to be mis-matched. This behavior leads to the presence of "double-surfacing" in the output scans, which can be observed in the generated point cloud of the environment. Figure 2.2 shows an example of such mis-registration. The chair in the depicted point cloud is represented by two copies, slightly offset from one another. Note that the point cloud shown is generated using a static scanning system, which can also suffer from mis-registration issues.

To provide consistent output geometry, the modeling techniques described in this dissertation are designed to prevent such "double-surfacing" from occurring in the output meshes. This trait is accomplished by performing volumetric analysis of the input scans, as discussed in Chapter 4.

2.3 Aligning Model Coordinates

The system path generated by the localization procedure is oriented according to the heading vector of the first pose [53, 54]. For geometry reconstruction, we often are interested in creating models of the environment that are aligned to world coordinates, so that the automatically-generated models described later in this dissertation can be compared to ground-truth architecture. Such alignment is also useful to keep the orientations of model coordinate frames consistent between separate scans of the same environment and to ensure that output geometry is easily recognizable. Therefore, before geometry processing, we perform an additional orientation step that aligns the path to the East-North-Up (ENU) coordinate frame.

One of the on-board sensors of the backpack hardware system is a 3D magnetometer, which is built into the Inertial Measurement Unit (IMU). Individual readings from a 3D compass are often noisy when indoors, due to nearby ferrous building elements that interfere with hard and soft iron calibration [80, 81]. Each individual reading is therefore considered unreliable during the localization process. Once the final, consistent path is generated we are able to generate a least-squares estimate of compass north and reorient the path so that North is aligned with the $+Y$-direction.

First, the 3D magnetometer reading is recorded at each pose during the data acquisition. As shown in the example in Figure 2.3a, these vector readings represent the magnetic field at that location. The direction of each reading is a noisy estimate of the direction of magnetic South. The magnetic field flows South, so the reverse direction, as shown in Figure 2.3a, is an estimate of North. Let $\vec{m}_i \in \mathbb{R}^3$ be the magnetometer reading at pose i, in the coordinate frame of the magnetometer sensor `mag`. The rotation required to orient the model coordinate frame to ENU coordinates is

$$\texttt{argmin}_{\theta\in[0,2\pi]} \sum_i \left| (R_z(\theta)\, R_{c\to m}(t_i)\, R_{\texttt{mag}\to c}\, \vec{m}_i) - \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} \right|^2 , \tag{2.1}$$

where t_i is the timestamp of pose i, and $R_z(\theta)$ is the rotation matrix

$$R_z(\theta) = \begin{bmatrix} \texttt{cos}\,\theta & -\texttt{sin}\,\theta & 0 \\ \texttt{sin}\,\theta & \texttt{cos}\,\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} . \tag{2.2}$$

This optimization rigidly rotates the model coordinate frame about the z-axis to best align all magnetometer readings with the direction we wish to denote south: the $-Y$-direction. An example of this procedure is shown in Figure 2.3b. This procedure ensures that all output models are aligned with magnetic North, in ENU coordinates. While not applied in the current configuration, it is possible to also align the model to true North if given the latitude and longitude coordinates of the scanning location, by performing a look-up of magnetic declination [82].

2.4 Point Cloud Generation

When a finalized path and coordinate frame is produced, one can immediately generate a point cloud of the raw scans taken in the environment, as discussed by Chen et al. [52]. This point cloud allows visualization of observed building geometry. Each point can be computed independently by transforming its position from its sensor frame to the model coordinate frame. In order to ensure accuracy of the exported points, we have adapted this point cloud reconstruction to use spherical linear interpolation (SLERP) to position the points in world coordinates [83]. This step is necessary since the scan frame timestamps may not align with the localization pose timestamps. An example point cloud of an indoor environment is shown

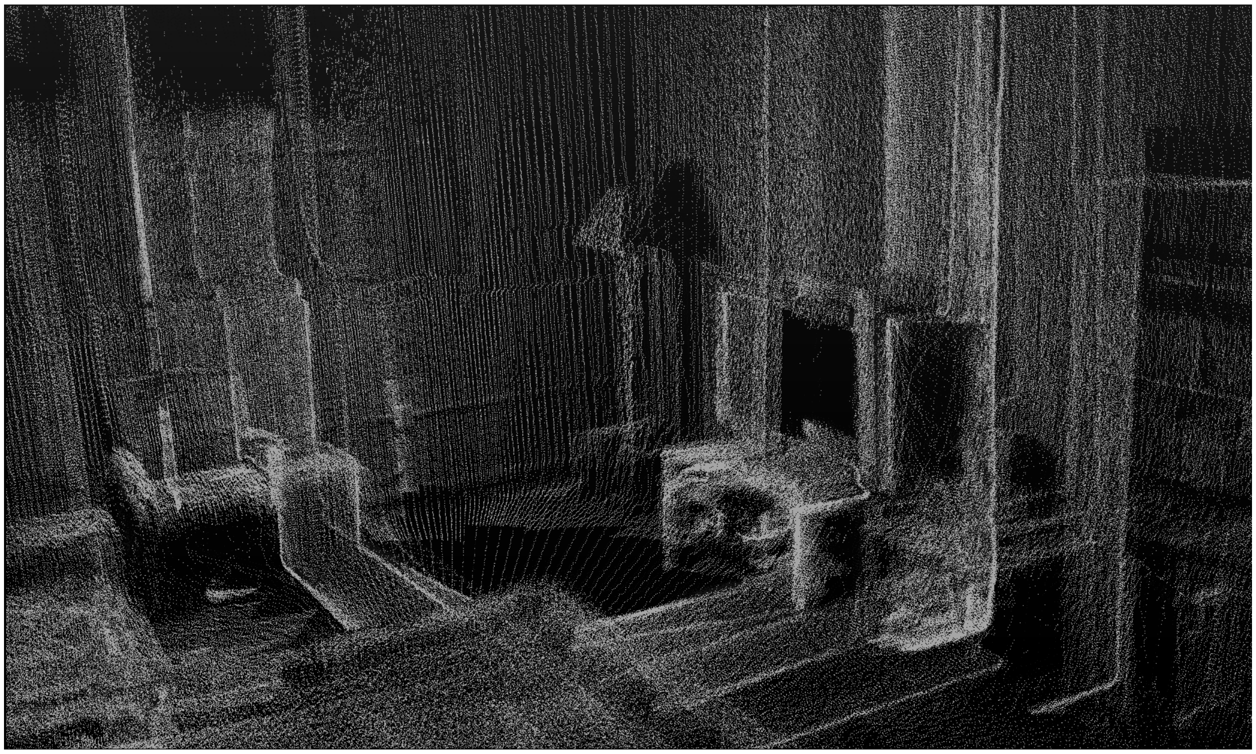


Figure 2.4: A point cloud of an indoor building environment.

in Figure 2.4. This scene shows a corner of a room, with a lamp to the left of a TV on a table.

There are also several existing techniques to apply color as texture to point clouds and other models [13, 84, 85]. The coloration is taken from camera imagery taken temporally close to when the range points were acquired. If no camera observes a point at the time it is captured, then it is left uncolored. In this dissertation, we have added an extrapolation procedure to allow points to be colored by a larger set of potential images. For a given point $\vec{x}$ taken at time t, the point color is determined by querying all available cameras for any images within Δt seconds of time t. Typically, we use a window size of $\Delta t = 2$ seconds. For each reported camera, we project the position $\vec{x}$ onto the camera image plane, and determine the pixel color at that position. This projection is demonstrated in Figure 2.5. In addition to color, we also compute a quality measure $q_x = (\vec{x} - \vec{c})^T \vec{n}$, where $\vec{c}$ is the camera position and $\vec{n}$ is the optical axis of the camera. The quality q_x is 1.0 if $\vec{x}$ projects into the center of the image, and decreases as the projected location moves away from the center of image. For all images within the time window $[t - \Delta t, t + \Delta t]$, the pixel color associated with the highest quality measure is chosen to color the point.

The point cloud is colored based on all available imagery in the dataset. Figure 2.6a shows a camera image of a room and Figure 2.6b shows the colored point cloud of the same environment.

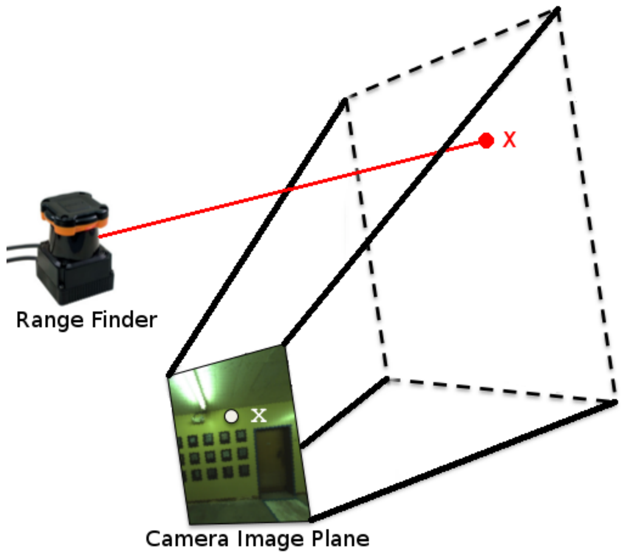


Figure 2.5: Determining the color of point x based on camera imagery.

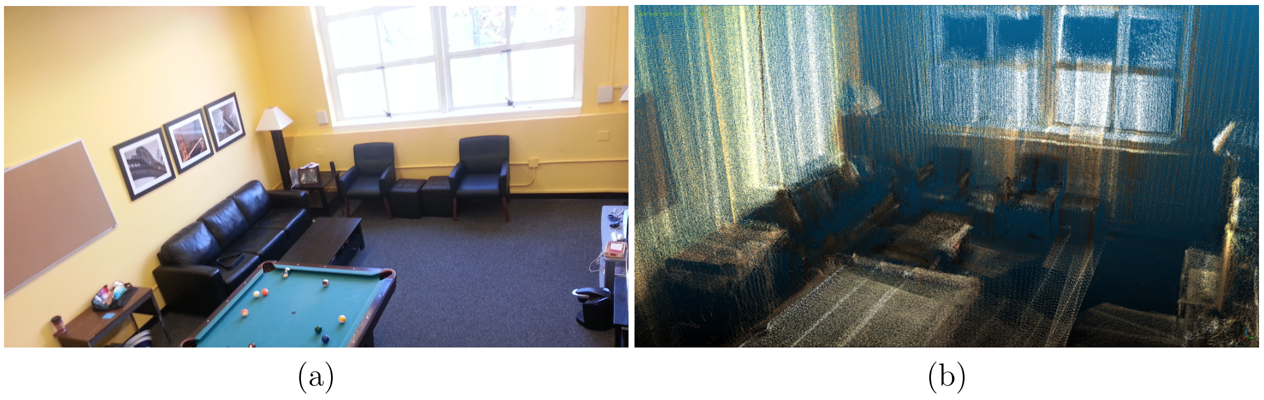


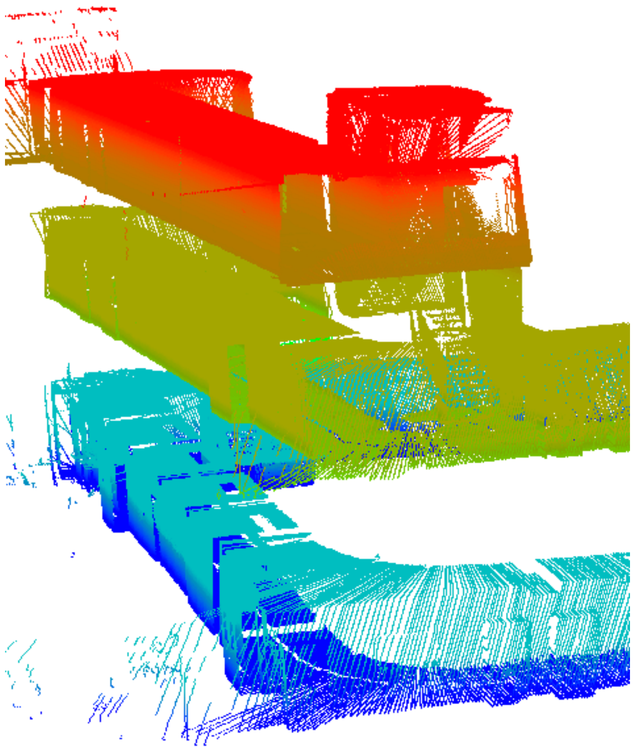
Figure 2.6: A point cloud colored by nearby camera imagery.

2.5 Partitioning a Point Cloud by Building Levels

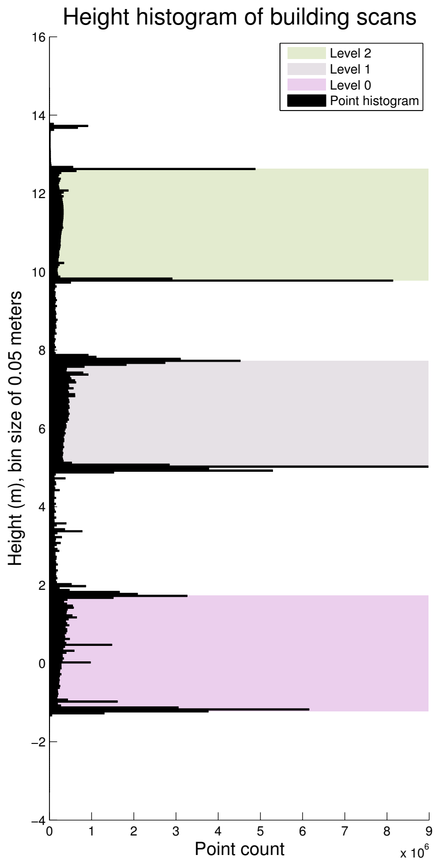
As a human-mounted scanning system, the backpack has the capability of collecting scans across several building levels in a single acquisition. The methods described in Chapter 3 deal with generating floor plans and require building levels to be partitioned so they may be processed independently. In this section, we discuss how the exported point cloud described in the previous section can be used to identify different levels observed in each dataset.

Some localization systems that rely on 2D grid-maps of wall samples are capable of detecting when the operator moves from one level to another and automatically partitioning their output grid-maps accordingly [50]. We need to determine both the elevation and the vertical extent of each level. To detect and separate building levels, we identify the primary floor and ceiling surfaces for each level. This identification can be done either in the point cloud domain or after a full model has been generated. The latter method is discussed in Section 5.2. Here, we discuss how level splitting can be accomplished via point clouds, which offer a faster if less accurate partitioning.

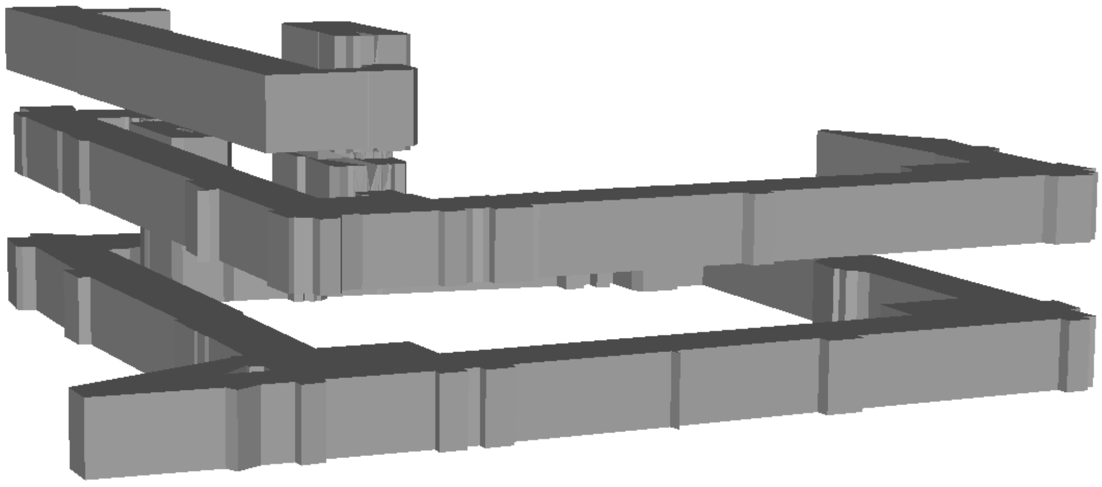
We have developed a histogram approach can be used to separate the point cloud by levels [7, 59]. Figure 2.7a shows an example point cloud, colored by height, which contains multiple levels. By computing a histogram along the vertical-axis of the point cloud, it is possible to find heights with high point density, which indicates the presence of a large horizontal surface. An example of this process is shown in Figure 2.7b, where peaks in the histogram correspond to the floors and ceilings of each scanned level in the building. Points scanned from above, in a downward direction, are used to populate a histogram to estimate the position of each floor, and the histogram used to estimate the position of each ceiling is populated by points scanned from below. The local maxima of these two histograms show locations of likely candidates for floor and ceiling positions, which are used to estimate the number of scanned levels and the vertical extent of each level. The candidate floor and ceiling heights are pruned by first taking the lowest floor maximum as the elevation of the first level's floor. The first level's ceiling elevation is determined by the most populated ceiling maxima position that resides below the next maximal floor elevation. This process is repeated for all levels, which allows for detection of both the number of levels and their range extents. Once levels are separated, they can be processed and analyzed separately. Figure 2.7c shows the final extruded mesh with all three scanned levels.



(a)



(b)



(c)

Figure 2.7: An example point cloud partitioning by height: (a) the input point cloud, showing geometry for three levels; (b) the vertical histogram showing estimates of each building level height; (c) the mesh of this building scan produced by method from Chapter 3.

Chapter 3

Floor Plan Generation

In this chapter, we discuss two methods to produce 2D floor plans of indoor building scans. Both methods start by generating a set of 2D wall samples. Wall samples are represented by a set of points that indicate areas likely to be large vertical surfaces. We list the numerous techniques we use to generate wall samples in Section 3.1.

We have developed two different floor plan generation techniques to operate on wall samples. In Section 3.2, we describe a global optimization method to generate watertight floor plan geometry based on the Eigencrust algorithm [12, 59]. This method explicitly fits both straight and curved features into the floor plan geometry to preserve curved building features. In Section 3.3, we discuss our second floor plan generation approach, which is adapted from our original algorithm, but does not require a global optimization step, allowing for faster processing [60]. Additionally, it explicitly partitions the floor plan geometry into individual rooms. This method also introduces simplification mechanisms that allow for minimal elements to represent the geometry while still preserving the watertight, volumetric topology.

Lastly, in Section 3.4, we compare these methods on several models, including multi-story building scans. We also compare our automatically generated results to the ground-truth blueprints of the scanned environments.

3.1 2D Wall Sampling

The input data used for floor plan generation consist of points in the horizontal (x,y) plane, which we call wall samples. These points depict locations of walls or vertical objects in the environment. We assume that interior environments have “2.5-dimensional” geometry: all walls are vertical whereas floors and ceilings are perfectly horizontal. We do not, however, assume that walls are perpendicular to each other. In many application scenarios only 2D scanners operating in one plane are used, so this assumption is needed to extract 3D information about the environment. The input to our algorithm is a set of 2D wall samples, where each sample is associated with the scanner pose that observed it, as well as estimates of the floor and ceiling heights at the wall sample location. Many mapping systems use a horizontal LiDAR scanner to estimate a map of the area as a set of wall sample positions, while refining estimates for scanner poses. These mobile mapping systems often have additional sensors capable of estimating floor and ceiling heights at each pose [46, 52].

Wall samples can be derived in three different ways. First, histogram analysis can be performed on the 3D point cloud generated for a building environment as discussed in Section 2.4, to generate a 2D sampling of wall positions [59]. This technique is described in Section 3.1.1. Second, wall samples can be exported directly from the grid-map generated alongside a particle filter during the localization step of scan processing [54, 60]. This method is described in Section 3.1.2. Third, wall samples can be generated from a processed 3D volumetric model of the environment. This last option is described in detail in Section 5.3. Each of these three methods provides its own advantages. Generating wall samples directly from a point cloud is the least processing intensive and the simplest method that still incorporates 3D information. Generating wall samples from a 2D grid map enables the sampling to occur in a streaming fashion, allowing for real-time processing of floor plans. Lastly, generating wall samples from a processed 3D model gives the highest accuracy and consistency, though it is the most processing intensive.

3.1.1 Generating Wall Samples from Point Clouds

Given a 3D input point set, P, we wish to generate a floor plan for each story represented. The coordinate system is defined so that the z-component represents height. First, we compute how many stories are present in P and partition P into separate point sets P_0, P_1, ..., P_f for each level by height, as discussed in Section 2.5.

For each story P_k, the method for computing wall samples is to subsample the full 3D point cloud to a set of representative 2D points [7, 55, 59, 60]. This process cannot be done in a streaming fashion, but can provide accurate estimates for wall positions. Such an approach is useful when representing dense, highly complex point clouds with simple geometry. Under the 2.5D assumption of the environment, wall samples can be detected by projecting 3D points onto the horizontal plane. Horizontal areas with a high density of projected points are likely to correspond to vertical surfaces. Our goal is to find these wall locations with a sampling of points $S_k \subseteq P_k$. By defining a grid along the x-y plane, we can find coordinate locations that are likely to represent walls. The width of each grid cell is denoted by d_s, which should be smaller than the minimum feature size expected in a floor plan. A typical value for this parameter is $d_s = 5$ cm. This value is chosen to ensure the geometry of major building elements is preserved, but is still larger than the expected noise value in the point cloud. This grid partitions P_k laterally. For a grid cell g, let the neighborhood set N_g be the subset of points of P_k that are within a horizontal distance of d_s from the center of g, as shown in Figure 3.1. Thus an input point can be in the neighborhood of multiple grid cells.

We perform the following checks on N_g to determine whether this neighborhood represents a portion of a wall. The first assertion is that a wall must be densely sampled, so if $|N_g|$ is less than a threshold, it is rejected. Our scanners create point clouds with thousands of points per square meter of surface, so a threshold of $|N_g| \geq 50$ points is sufficient. This value is dependent on the density of the point cloud and rate of scanning. The second check is to verify the height of a wall. The vertical support of N_g must be at least the minimum wall height threshold. The value of this height threshold may vary depending on application, but a length of $H = 2$ meters works well to capture permanent wall features while ignoring furniture and other interior clutter. We also wish to verify that the distribution of points

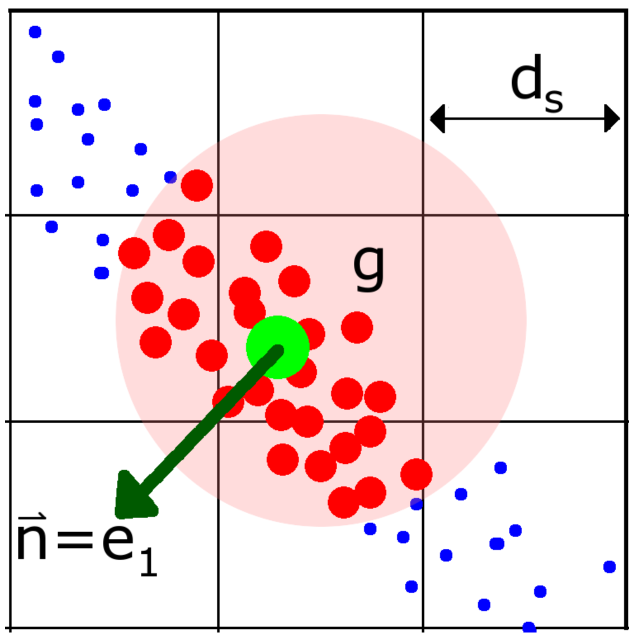
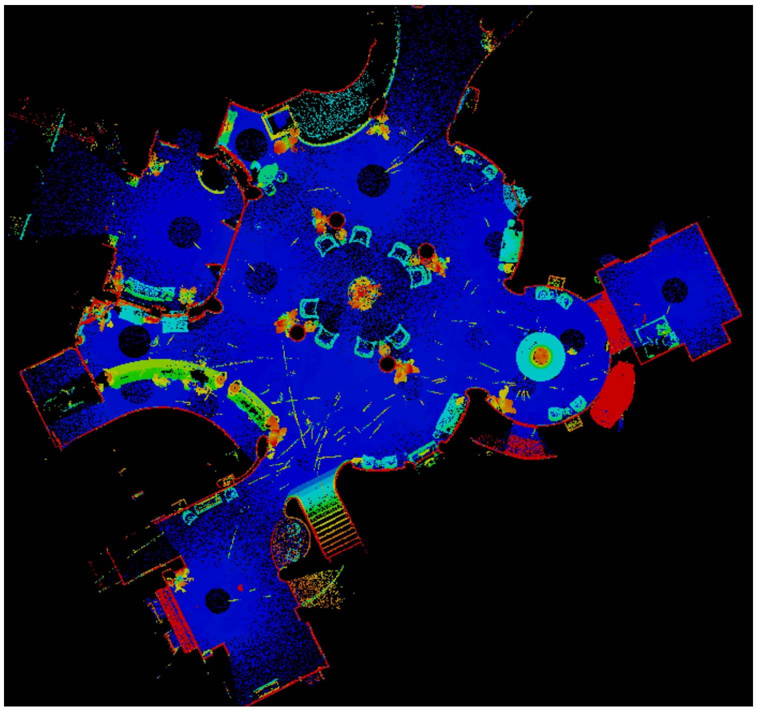


Figure 3.1: A point set P_k (small dots in blue) is partitioned by a grid in the x-y plane (lines in black). For each grid cell, a neighborhood of points (medium dots in red) is computed and the median position of the points (large dot in green) is taken as a wall sample. The normal of this neighborhood (vector in dark green) is computed by PCA.

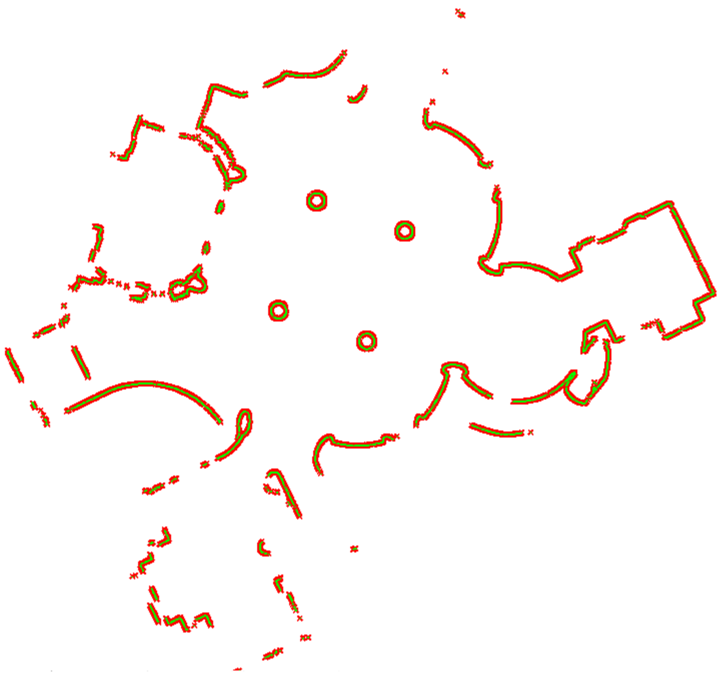
is uniform vertically, which signifies that a flat surface was captured. For this uniformity check, we compute the histogram of the heights of N_g with 16 bins and require that at least 12 of these must be non-empty. Additionally, at least three of the highest four bins must be non-empty, which ensures a neighborhood extends all the way to the ceiling. If these requirements are fulfilled, g is said to represent a wall. The median horizontal position for the elements of N_g is computed. If this median lies within the bounds of g, then it is taken as a "wall sample" for g.

The normal vector for this wall sample is calculated using Principal Components Analysis (PCA) [86]. Let C be the 2×2 covariance matrix of the xy-positions of the elements of N_g and $\vec{e}_1, \vec{e}_2$ be the eigenvectors of C with corresponding eigenvalues $\lambda_1 \leq \lambda_2$. The normal vector is chosen to be $\vec{n} = \vec{e}_1$, as shown in Figure 3.1. The confidence of this normal estimate is computed as $c = 1 - \frac{2\lambda_1}{\lambda_1+\lambda_2}$. If c is less than 0.15, then the neighborhood points are not well-aligned and are rejected as the location for a wall. If pose information for the scanner is known for each point, this normal vector's direction may be flipped to guarantee that it points towards the laser scanner. Performing the above operation for all grid cells produces a set of wall sample points $S_k \subseteq P_k$.

Each wall sample is represented by its 2D position, the minimum and maximum height values of the points that sample represents, and the poses of the scanners that observed the sample location. As we discuss later, these scanner poses provide crucial line-of-sight information that facilitate floor plan reconstruction. Wall samples can be generated from point clouds from any type of scanning system. Figure 3.2 shows an example of collecting wall samples collected from a point cloud of a hotel lobby. While most examples in this



(a)



(b)

Figure 3.2: (a) Point cloud of scanned area, viewed from above and colored by elevation; (b) wall sample locations generated from point cloud. Clutter such as furniture or plants do not affect position of wall samples.

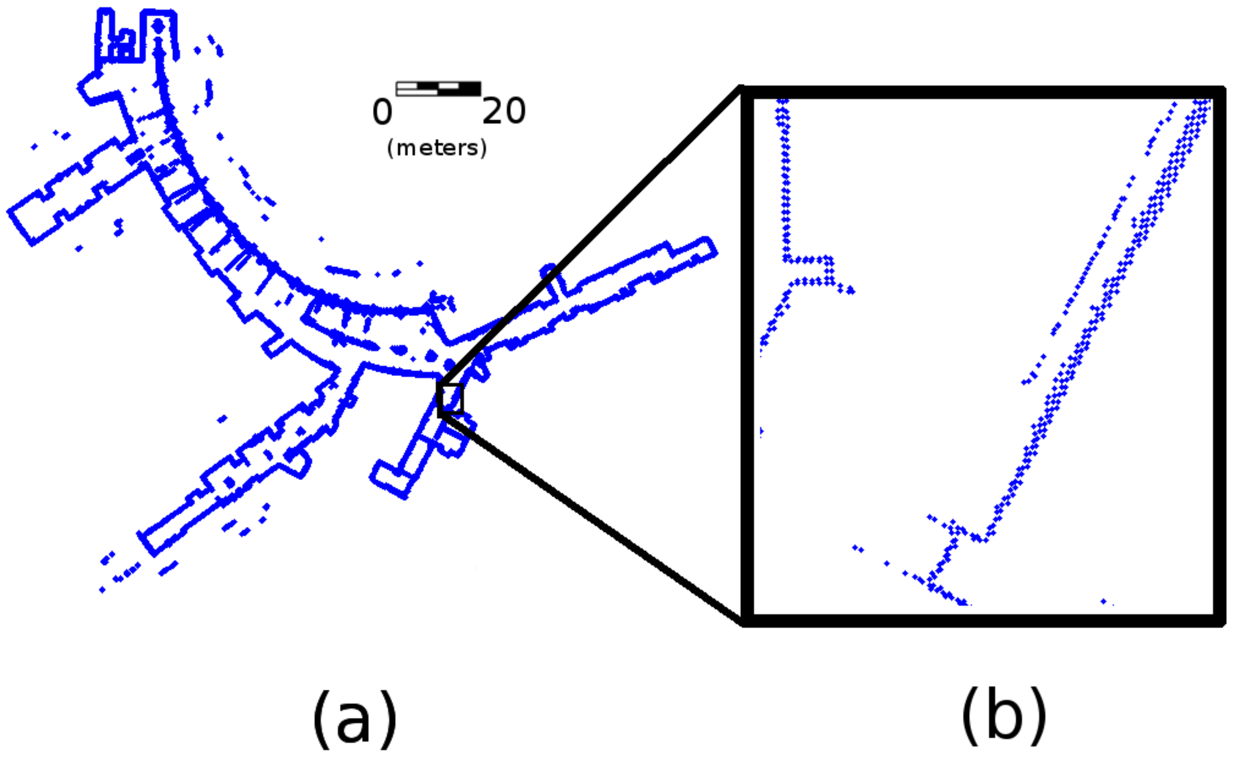


Figure 3.3: Example wall samples of hotel hallways and lobby generated from a particle filter system. (a) Wall samples of full model; (b) close-up of wall in model.

dissertation are generated by the backpack-mounted scanning system discussed in Chapter 2, this point cloud was created with static scanners. An example of wall samples generated from backpack scans for a hotel hallway is shown in Figure 3.3. As shown, even though the walls are well sampled, noise in the localization estimate causes noisy wall samples with outliers.

3.1.2 Generating Wall Samples from Particle Filter Grid-Maps

Many mapping systems use a horizontal LiDAR scanner to estimate a map of the area as a set of wall sample positions, while refining estimates for scanner poses. Such systems perform Simultaneous Localization and Mapping (SLAM) by defining a rough grid-map of the area while also estimating the pose of the system via a particle filter [46, 54]. This grid-map is often represented by a set of 2D position samples that indicate the presence of walls or other obstacles [45]. In such particle filtering approaches, the grid-map can be used directly as a set of wall samples for floor plan model generation [60]. As discussed in Section 3.3, we have developed a technique for floor plan generation that is fast enough to work in a streaming, real-time fashion on this type of input data. Particle filtering approaches to localization typically allow for real-time mapping in a streaming fashion [87, 88] and can therefore benefit from a real-time floor plan generation algorithm that delivers a live map of the environment.

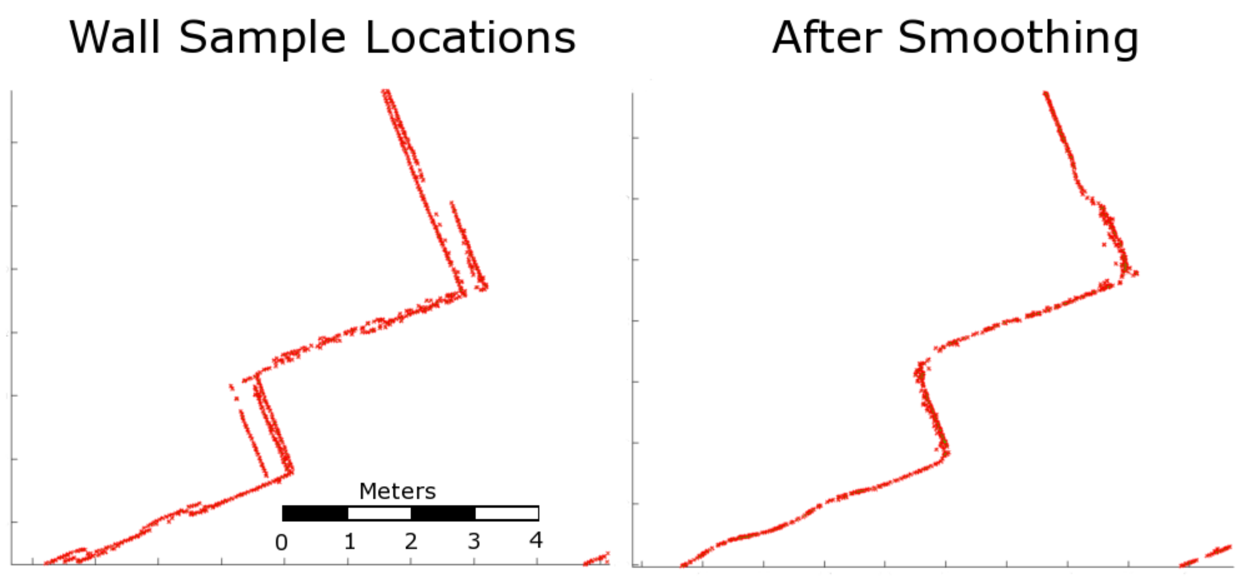


Figure 3.4: Average filtering reduces occurrences of registration errors exposed in wall sampling.

The input to our 2D floor plan algorithm is a set of 2D wall samples, as computed in Section 3.1, where each sample is associated with the scanner pose that observed it, as well as estimates of the floor and ceiling heights at the wall sample location. In Section 3.4, we show comparisons of floor plans generated with both types of wall samples: those generated from point cloud data and those generated from particle filter grid-maps.

3.2 Eigencrust Floor Plan Reconstruction

Given a point cloud representing a subset of the interior of a building that can cover multiple stories, we propose a three-step process to generate a watertight floor plan for each story in the point cloud by fitting line segments and curves to model features [59]. First, the output wall samples generated using the method described in Section 3.1 are smoothed to prevent double-surfacing, as discussed in Section 3.2.1. Second, the Delaunay triangulation of the generated wall samples is computed and each triangle is labeled as “inside” or “outside” by a 2D version of the Eigencrust algorithm [12]. This process is discussed in Section 3.2.2. The edges that separate these two labels are output as wall locations, as detailed in Section 3.2.3. Third, noise reduction is achieved by fitting these facets to line segments and curved components with Random Sample Consensus (RANSAC) [89], discussed in Section 3.2.4.

3.2.1 Smoothing Wall Samples

A crucial step in generating a point cloud is to register the poses of separate scan positions to a common coordinate system. Any error in this process may generate duplicate instances of a scanned wall with slight misalignments. A wall position may be scanned from multiple

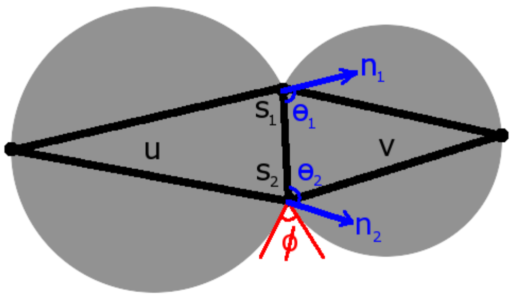


Figure 3.5: The angles of intersection between two triangles u and v, which share the edge $\overline{s_1 s_2}$. These values are used to compute weights between triangle pairs.

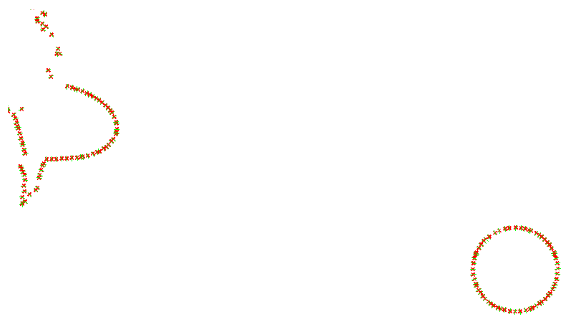
disjoint poses, causing it to appear multiple times in the set of generated wall samples. If the registration error is significant, smoothing may be required to force these wall instances to become aligned [59].

For a model with wall sample set S_k, a sample $s \in S_k$ with normal vector $\vec{n}$ has a smoothing set that is a subset of S_k within an anisotropic neighborhood of s. The position of s is translated to be the mean position of this smoothing set. The smoothing set contains all samples that are (a) within some smoothing distance of s along the $\pm\vec{n}$ direction, (b) no more than $3d_s$ away from s in the direction orthogonal to $\vec{n}$, and (c) whose normals are aligned to within $\pi/8$ radians of $\vec{n}$. The smoothing distance must be less than the minimum allowable spacing between parallel walls, but at least as large as the expected registration error. This step is not necessary if the registration error is smaller than d_s. An example of this smoothing process is shown in Figure 3.4.

3.2.2 Labeling a Triangulation of Wall Samples

We wish to determine the topology of the 2D sample set S_k for each story $k \in \{0, 1, ..., f-1\}$. This task is accomplished by a 2D variant of the Eigencrust algorithm [12]. This will partition 2D space into regions labeled "inside" and "outside" and export the borders separating these regions. These borders form the faces of a 2D simplicial complex, which are guaranteed to be watertight and not self-intersecting. Eigencrust has been shown to be more robust to outlier samples than related algorithms, such as Powercrust [8, 12].

The Eigencrust algorithm first computes the 2D Delaunay triangulation, T, for the sample set S_k plus four bounding box corners. Eigencrust constructs a sparse graph where the nodes represent triangles and edges join the nodes with weights corresponding to the geometry of the triangles. Edges with positive weights indicate that triangles should have the same labeling, while negative weights indicate that the triangles connected should be labeled oppositely. Generalized eigensystems are solved to determine the best triangle labelings to fit these connections. For our 2D variant of Eigencrust, we keep the same criteria for placing edge weights as in Kolluri et al. [12], but modify the negative edge weight values. We exploit additional information of normal vectors for each sample location. If a negative edge is placed between two triangles u and v, we use the weighting


(a)

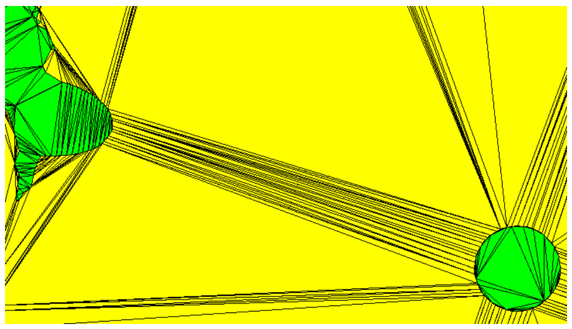

(b)

Figure 3.6: An example triangulation of wall samples: (a) a set of wall samples for a column; (b) each triangle is labeled as "inside" (yellow) or "outside" (green).

$$w_{u,v} = -e^{4+4\cos\phi+2\sin\theta_1+2\sin\theta_2} \tag{3.1}$$

where these parameters are shown in Figure 3.5. This weighting is very close to the one used in Kolluri et al. [12]. The value ϕ is defined to be the angle at which the circumcircles of u and v intersect. The intersection of the triangles u and v is a line segment whose endpoints are samples s_1 and s_2, which have normal vectors $\vec{n}_1$ and $\vec{n}_2$ respectively. The values of θ_1, θ_2 are defined to be the angles between $\overline{s_1 s_2}$ and $\vec{n}_1$, $\vec{n}_2$, respectively. This weight is defined to have a large magnitude when both the normal vectors are perpendicular to the line $\overline{s_1 s_2}$, which is a strong indication of a wall in the original point cloud P_k.

For each pair of samples $s,s' \in S_k$ such that (s,s') is an edge of the Delaunay triangulation T, let u and v be the poles of s, and u' and v' be the poles of s'. Poles are defined as the extrema vertices of the Voronoi cell for a given point. Each of the pairs (u, u'), (u, v'), (v, u'), and (v, v') are edges in E. If a negative edge is not already defined, each of these pairs has positive edge weight

$$w_{u,u'} = e^{4-4\cos\phi} \tag{3.2}$$

where ϕ is again defined as the angle at which the circumcircles of u and u' intersect [12]. Next, we can constrain some triangles to be interior or exterior based on a priori information. The label "inside" refers to locations of open area within a building where a person can exist. The term "outside" refers to all other areas, which include both the exterior of the building as well as the areas inside walls or other solid spaces. We can immediately force the label of "outside" to any triangles connected to the bounding box corner vertices. If the pose information for the scanner is available, we can force the label of "inside" to a subset of triangles. First, we investigate whether the 2D line-of-sight from a scanner pose to a scan sample crosses triangles. If the center 50% of this laser travel distance crosses a triangle, that triangle is intersected by that scan line. If a triangle is intersected by 10 or more scan lines, it is assumed to be "inside". Second, all triangles that contain a pose position of a scanner are marked as interior. If a mobile scanning system is used, then the path traversed by that system can also be used to mark interior triangles. Each pair of adjacent pose positions represents a line segment in 2D space. If both of these poses contributed scans to the wall sample set S_k, then any triangles that are intersected by this line segment are also constrained to be interior.

The remaining triangles are labeled inside or outside by solving generalized eigenvalue systems that minimizes the total positive edge weights that connect oppositely labeled triangles and negative edge weights that connect triangles with the same labeling [12].

The non-pole triangles are then put into a corresponding graph. Since the edge-weights of this non-pole graph in the 3D Eigencrust algorithm use the aspect ratio of bounding triangular faces, there is not a direct translation of these weight calculations to 2D. Thus the same weights are used as for the pole graph G, defined in Equations 3.1 and 3.2. A corresponding generalized eigenvalue system is solved for this non-pole graph, which provides a labeling for all triangles in T. An example output of this triangulation labeling process is shown in Figure 3.6.

All the "outside" nodes are contracted into one outside super-node and all "inside" nodes are contracted into one inside super-node, creating graph G'. The nodes and edges of G' are then converted into a matrix $L \in \mathbb{R}^{|G'|\times|G'|}$. Each element is defined as $L_{ij} = L_{ji} = -w_{i,j}$ with the diagonal $L_{ii} = \sum_{j\neq i} |L_{ij}|$. The diagonal matrix D is the same size as L and has the same diagonal elements as L. These values are used to find the solution to the generalized eigensystem $L\vec{x} = \lambda D\vec{x}$. The eigenvector $\vec{x}$ associated with the smallest non-zero eigenvalue λ is taken as the result [12].

Each element of x corresponds to a pole triangle in T. The i^{th} triangle is labeled as "inside" or "outside" based on the sign of x_i. This step partitions the poles into the inside/outside categories. To label all the non-poles, we construct another graph. All the "outside" poles in G' are collapsed into one outside super-node and all the "inside" poles of G' are collapsed into one inside super-node. All the non-pole triangles are also nodes of this non-pole graph, and edges are defined between every pair of touching triangles, assuming both triangles are not part of the same super-node. Any edge that happens to be between the two super-nodes are negatively weighted according to Equation 3.1. Any edge involving at least one non-pole is positively weighted according to Equation 3.2. The non-pole graph is then converted to the matrix H in the same manner as described above for G'. Let D_H be the diagonal component of the matrix H. Solving the generalized eigensystem $H\vec{y} = \mu D_H\vec{y}$ yields the eigenvector $\vec{y}$, which gives us the inside/outside labelings for each of the non-pole triangles as well [12]. An example output of this triangulation process is shown in Figure 3.6.

3.2.3 Forming Floor Plan Boundary Edges

We have found the following post-processing clean-up techniques useful for increasing the quality of the output model. First, we represent this topology of triangles as a set of unions. Any subset of triangles with the same label that forms a connected component is represented as a single union using the Union-Find algorithm [90]. Unions that are composed of a small number of triangles are most likely mislabeled, since the smallest building features should still be sufficiently sampled. The appropriate cut-off value depends on the value of d_s used to form samples, but a value of 30 triangles is typically used. All inside unions that have fewer than this many triangles are relabeled as outside. The set of unions is recomputed and then all outside unions that meet this criterion are relabeled as inside. Another post-processing smoothing process is to remove jagged edges from the boundaries between labelings. Every

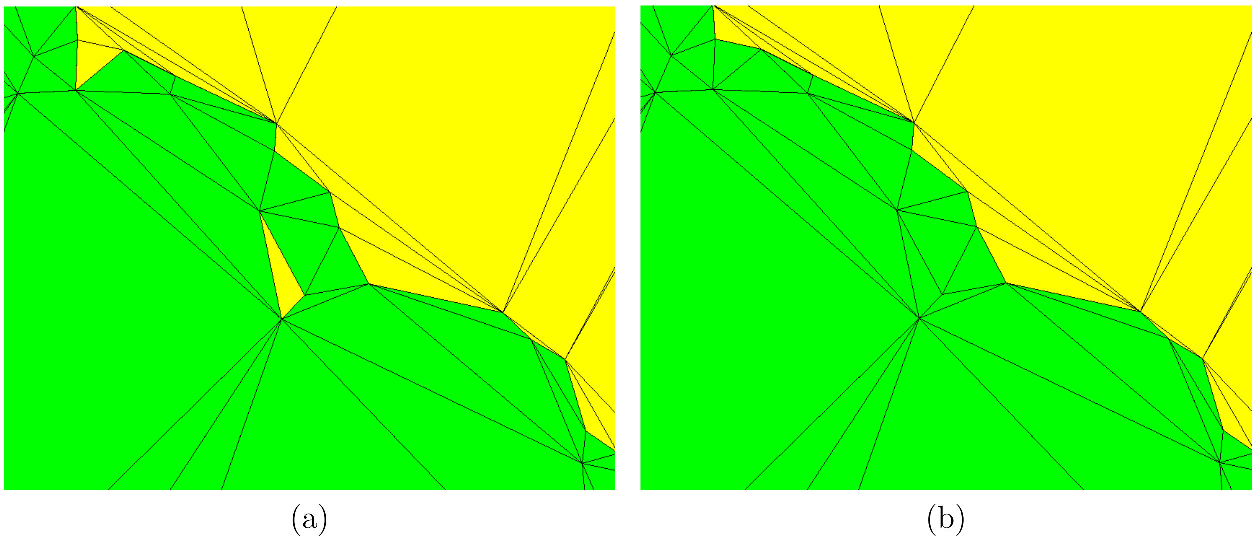


Figure 3.7: Original triangulation labeling (a) is cleaned by removing small unions and sharp protrusions (b).

triangle has three neighboring triangles, each sharing one of its edges. If a triangle $t \in T$ has only one neighbor that shares the same labeling and that neighbor is connected to t's shortest edge, then t is considered to be a protrusion. It is unlikely for building geometry to be correctly represented by such a protrusion, so all protrusions labeled as inside are relabeled as outside. After this relabeling, all outside protrusions are found and relabeled as inside. An example of this processing is shown in Figure 3.7.

The set of edges in T that are shared by an "inside" triangle and an "outside" triangle are exported as boundary edge set, B. Each element in B is a line segment. In total, B constitutes a watertight floor plan of the scanned area.

3.2.4 Fitting Edges with Lines and Curves

Since there may exist areas in the building geometry that are insufficiently scanned or errors incurred during processing, it is important to utilize common architectural patterns to reduce these errors. Additionally, applications may require a floor plan to be composed of parametric lines and curves.

Local model-fitting approaches such as region growing are sub-optimal because unconnected architectural features may use the same geometric model. For example, the back walls in a row of offices may lie on the same plane, even though they are in different rooms. Since we need to fit both curves and straight lines simultaneously, a reasonable technique for this situation is one that is non-local and flexible, such as Random Sample Consensus (RANSAC) [89].

We apply RANSAC to the subset of samples used as vertices in the boundary edges B. Each iteration randomly picks three samples from this set that have not yet been associated with a model. These three points uniquely define a circle. A line of best fit can also be obtained for these points. Both the circle and the line models are compared to the subset of samples still unassigned. Inlier sets for both of these models are computed. An inlier is

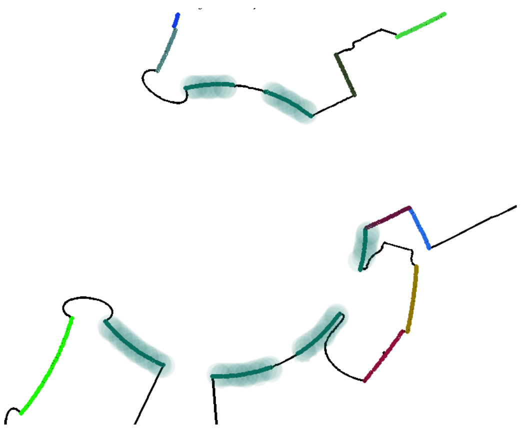


Figure 3.8: The walls of a circular room are fit to the same circle model (shown highlighted in teal), even though they are comprised of several connected components.

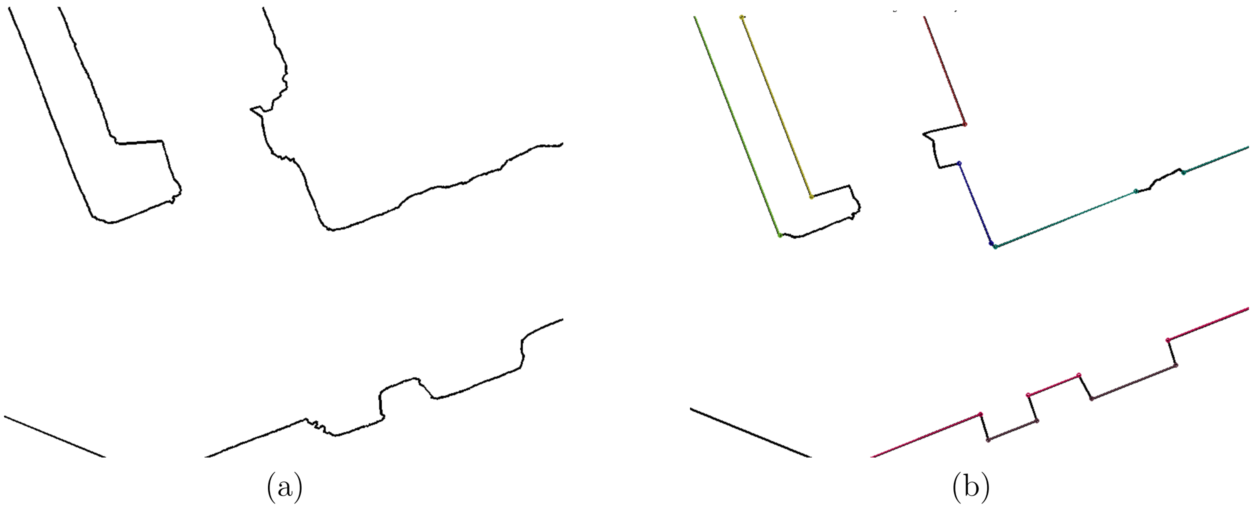


Figure 3.9: (a) Watertight wall edges; (b) curve-fitting via RANSAC reduces sample location error and sharpens corners.

an unassigned sample that is both within a threshold distance of a model and whose normal vector is within a threshold angle to the model's normal vector at that location. Only models that exceed a specified minimum number of inliers are considered. The line or circle model found with the smallest average error is returned as a valid model and its inlier vertices are no longer considered for subsequent models.

This process continues until no new parametric model can be found. The result is a set of models, where each model has a set of inlier samples and a parametric representation of either a line or a circle. The topology defined in B has not been altered, so its edge elements can be used to partition the inlier vertices of each model into a set of connected components, again with Union-Find [90]. Any of these connected components composed of too few samples is most likely a misclassification. Thus, any component that spans less than a meter in length is removed, with its samples reset to be unassigned. Choosing a value of 1 meter means that only large, primary building features will be grouped. Additionally, we wish to encourage models to extend along the edges defined in B. These unassigned samples are then associated with the closest model. These two steps encourages outlier samples to belong to models that are topologically close. These steps also grow models to be adjacent, encouraging sharp corners.

Once the revised parametric models and their respective inliers are computed, the inlier samples and edges in B are replaced by edges that conform exactly to their models' locations. This process reduces the overall number of samples and removes small perturbation errors from the floor plan. Figure 3.8 demonstrates the results of this process, which fits a circle to the walls of a round room with several entrances. Figure 3.9 shows how this process can also be applied to straight walls.

This section details a technique to generate 2D architectural floor plans. By projecting each story's point set to two dimensions and performing a density analysis, we can increase wall location accuracy over a direct 3D approach. In Section 3.4, we show several example models generated by this technique and compare these models both to the ground-truth blueprints of the building as well as to other automatic as-built floor plan generation techniques.

3.3 Floor Plan Generation with Room Partitioning

In this section, we present a technique to automatically generate accurate floor plan models at real-time speeds for indoor building environments [60]. This technique provides an expansion on the method discussed in the previous section. In Section 3.3.1, we discuss how we compute the interior space of the 2D floor-plan, which defines the building geometry. The partitioning of this interior space into separate rooms is considered in Section 3.3.2. With these room labels, the geometry can be simplified, as described in Section 3.3.3. In Section 3.3.4, the final 2D geometry is extruded into a 3D model output.

3.3.1 Defining the Interior Area

In this section, we discuss how the interior area is defined in the improved floor plan generation method. The main change from the previous method discussed in Section 3.2 is that we remove the global optimization step. The advantage of this change is that any errors in the resulting partition only have a local effect, whereas previously they could potential corrupt the entire model. We instead rely on light-on-sight information to identify interior area, which provides higher confidence in the resulting geometry.

We generate a floor plan by partitioning space into *interior* and *exterior* domains. The interior space of the floor plan is computed from the input wall samples. The desired output consists of not only the architectural features such as walls, but also a volumetric representation of the environment such as rooms and hallways. The exterior represents all space outside of the building, space occupied by solid objects, or space that is unobservable. Once this partitioning is completed, as described below, the boundary lines between the interior and exterior are used to represent the exported walls of the floor plan.

The input wall samples are used to define a volumetric representation by generating a Delaunay triangulation in the plane. These wall samples are the same discussed in Section 3.1.

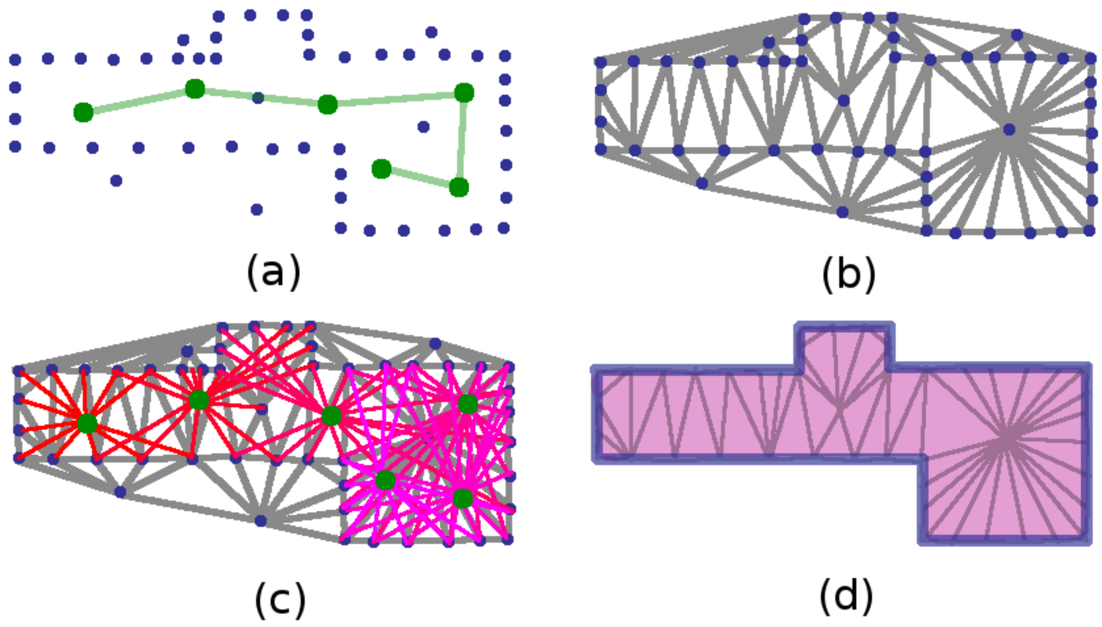


Figure 3.10: Example of the carving process to find interior triangles: (a) wall samples (in blue) with path of scanner (in green); (b) Delaunay triangulation of wall samples; (c) laser scans from each pose (in red); (d) triangles that intersect laser scans (in pink), used as interior triangles, with building model border (in blue).

Each triangle is labeled either interior or exterior by analyzing the line-of-sight information of each wall sample. Initially, all triangles are considered exterior. Each input wall sample, $p \in P$, is viewed by a set of scanner positions, $S_p \subseteq \mathbb{R}^2$. For every scanner position $s \in S_p$, the line segment (s, p) denotes the line-of-sight occurring from the scanner to the scanned point during data collection. No solid object can possibly intersect this line, since otherwise the scan would have been occluded. Thus, all triangles intersected by the line segment (s, p) are relabeled to be interior.

This process carves away the interior triangles with each captured scan. Since these scans are captured on a mobile scanner, the scanner poses are ordered in time. The line segment between adjacent scanner poses must also intersect only interior space, since the system traverses the open areas of the environment. In addition to carving via scanner-to-scan lines, the same carving process is performed with scanner-to-scanner line segments.

Figure 3.10 demonstrates an example of this process. Figure 3.10a shows the input wall samples, in blue, as well as the path of the mobile mapping system, in green. These points are triangulated, as shown in Figure 3.10b. The line-of-sight information is analyzed from each pose of the system, demonstrated by the laser scans from each pose to its observed wall samples in Figure 3.10c. The subset of triangles that are intersected by these laser scans are considered interior. Intersections are considered exclusive of triangle edges, since otherwise may result in over-carving. The interior triangles are shown in pink in Figure 3.10d, denoting the interior volume of the reconstructed building model. The border of this building model is shown in blue, comprising the estimated walls of the floor plan.

The method above may produce errors if there is mis-registration in the localization of the poses of the scanners during the data collection. For instance, if opposite sides of the same wall are scanned, this wall appears in the output with some thickness t. If the estimated positions of the scanners on either side of the wall are inaccurate, the observed wall thickness varies by the component of this localization error orthogonal to the wall plane. If this error is greater than t, then every triangle associated with the wall may be labeled interior, and the wall is carved away entirely from the output. To prevent these topological errors, we check for occlusions when carving each line segment (s, p). If another wall sample p' is located between the positions of s and p, then the line segment is truncated to (s, p'). This occlusion check is performed by determining if the line segment (s, p) in intersects the ball of diameter d_s centered at p', where d_s is the grid size used to generate wall samples. Thus, no features captured by wall samples are ever fully carved away, which preserves the details of the environment.

3.3.2 Labeling Rooms Within the Floor Plan

A novel aspect of this technique is the use of room labeling to enhance building models, e.g. for thermal simulations of interior environments [6, 60]. Partitioning a model into separate rooms can have many motivations. Existing work partitions models into sub-maps, similar to rooms, to capture line-of-sight information for fast rendering of building environments [91]. This technique requires axis-aligned rectilinear building geometry, which often is not a valid assumption. Others have partitioned building environments into sub-map segments with the goal of efficient localization and tracking [48]. This approach is meant to create easily recognizable subsections of the environment, whereas our proposed room labeling technique uses geometric features to capture semantic room definitions for both architectural and building energy simulation applications.

Once the volume has been partitioned into interior and exterior domains, the boundary between these domains can be exported as a valid floor plan of the environment. Keeping the full volumetric information can also be useful, such as for partitioning the interior into separate rooms.

We define a *room* to be a connected subset of the interior triangles in the building model. Ideally, a room is a large open space with small shared boundaries to the rest of the model. Detected rooms should match with real-world architecture, where separations between labeled rooms are located at doorways in the building. Since doors are often difficult to detect, or absent, there is no strict mathematical definition for a room, so this labeling is heuristic in nature.

We model room labeling as a graph-clustering problem. First, a rough estimate for the number of rooms and a seed triangle for each room is computed. A seed triangle is representative of a room, where every room to be modeled has one seed triangle. These seeds are used to partition the remainder of interior triangles into rooms. This process typically overestimates the number of rooms, so prior knowledge of architectural compliance standards is used to evaluate each estimated room geometry. With this analysis the number of ill-formed rooms is reduced, providing an update of the original seed points. This process

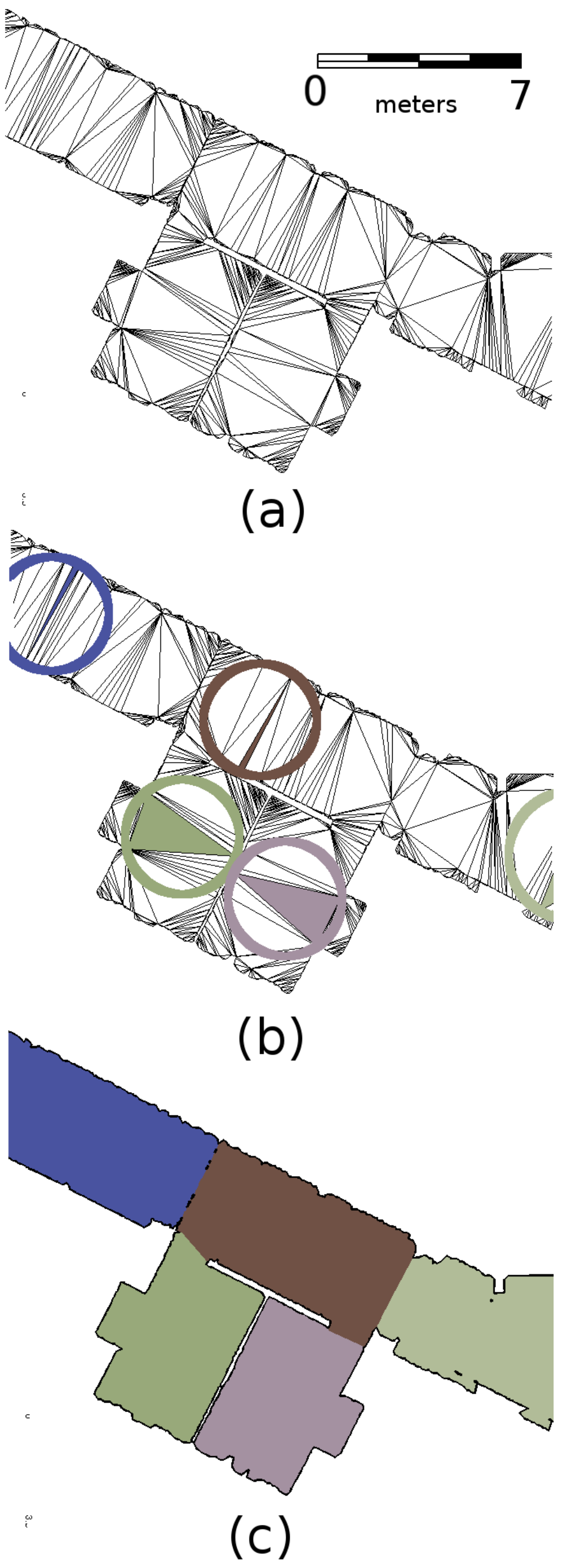


Figure 3.11: Example room seed partitioning: (a) interior triangulation; (b) the room seed triangles and their corresponding circumcircles; (c) room labels propagated to all other triangles.

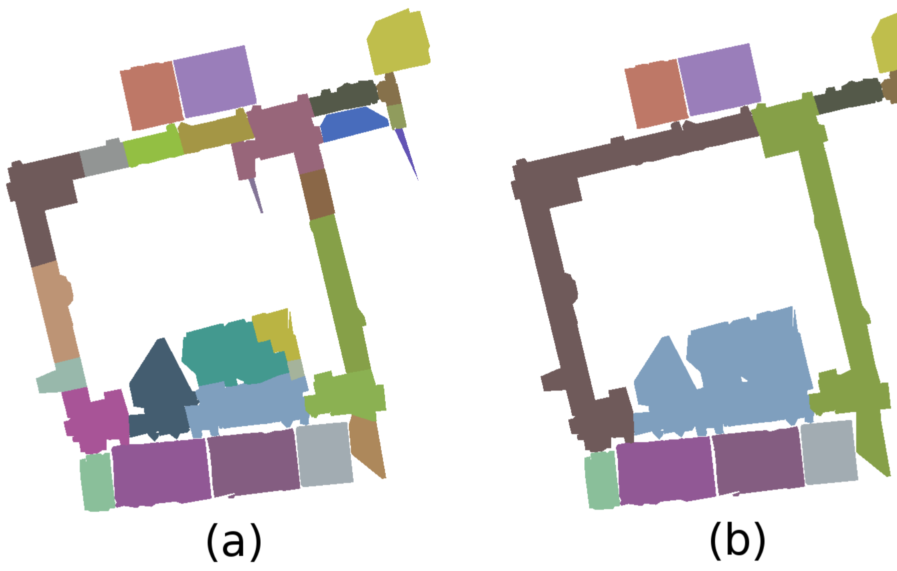


Figure 3.12: Room labeling refinement example: (a) initial room labels; (b) converged room labels.

is repeated until the set of room seeds converges.

We use the Delaunay property of the triangulation to identify likely seed triangle locations for room labels, which states that no vertex is strictly inside the circumcircle of any triangle in this triangulation. If we assume that the input wall samples represent a dense sampling of the building geometry, this property implies that the circumcircles of none of the interior triangles intersect the boundary walls of the carved floor plan, forcing these circles to represent only interior area. This make-up allows each triangle's circumradius to provide an estimate of the local feature size at its location on the floor plan boundary polygon. Given the example interior triangulation shown in Figure 3.11a, the highlighted triangles in Figure 3.11b show the chosen seed locations.

Triangles with larger circumradii are likely to be more representative of their rooms than those with smaller circumradii. We form the initial set of room seeds by finding all triangles whose circumcircles are local maxima. Specifically, given the set of interior triangles T, each triangle $t \in T$ has circumcircle c_t, which is tested against every other circumcircle in T that is intersected by c_t. If c_t has the largest radius of any intersecting circumcircle, then t is considered a seed for the room labeling. This process selects the largest triangles that encompass the space of rooms as the seeds for room labeling. Figure 3.11b shows example seed triangles and their corresponding circumcircles. The result is an estimate of the number of rooms and a rough location for each room.

Let K be the number of room seeds found, with the seed triangles denoted as t_1, t_2, ..., t_K. We wish to partition all triangles in T into K rooms. This step can be performed as a graph cut on the dual of the triangulation. Specifically, each triangle $t \in T$ is a node in the graph, and the edge weight between two abutting triangles is the length of their shared side. Performing a min-cut on this graph partitions rooms to minimize inter-room boundary length. In other words, rooms are defined to minimize the size of doors. This process propagates the room labels to every triangle, and the boundaries between rooms are composed of only the smallest edges in the triangulation T. The result of this process is shown in Figure 3.11c.

Alternatively, a greedy algorithm can be employed to flood-fill rooms. To do so, we initialize the labeling so that each seed triangle t_i has a unique label, while all non-seed triangles are unlabeled. We then construct a priority queue of all edges in T that separate a labeled triangle from an unlabeled triangle, sorted by decreasing edge length. Iterating through this queue, the top value represents the longest edge in the queue, adjoining a labeled triangle $u \in T$ and unlabeled triangle $v \in T$. We then pop this edge, propagate the label from u to v, then push all edges of v adjoining unlabeled triangles onto the queue.

Room labels partition T into a set of rooms $R = \{R_1, R_2, ..., R_K\}$, where each room R_i contains a disjoint subset of T and has seed triangle t_i. The initial room seeds over-estimate the number of rooms, since a room may have multiple local maxima. This case is especially true for long hallways, where the assumption that one triangle dominates the area of the room is invalid. An example is shown in Figure 3.11c, where two lower rooms, shown in green and purple, are properly labeled, but their adjoining hallway is broken into three subsections. The solution is to selectively remove room seeds and redefine the partition.

A room is considered a candidate for merging if it shares a large perimeter with another room. Ideally, two rooms sharing a border too large to be a door should be considered the same room. By the Americans with Disabilities Act (ADA) compliance standards, a swinging door cannot exceed 48 inches in width [92]. Accounting for the possibility of double-doors, we use a threshold of 2.44 meters, or 96 inches, when considering boundaries between rooms. If two rooms share a border greater than this threshold, then the seed triangle with the smaller circumradius is discarded. This process reduces the value of K, the number of rooms, while keeping the interior triangulation T unchanged. With a reduced set of room seeds, existing room labels are discarded and the process of room partitioning is repeated. This iteration repeats until the room labeling converges.

Another way room labels are refined is by comparing the path of the mobile mapping system to the current room labeling for each iteration. The mobile scanning system does not necessarily traverse every room, and may only take superficial scans of room geometry passing by a room's open doorway. Since the room is not actually entered, the model is unlikely to capture sufficient geometry, and so only a small handful of wall samples are acquired for such a room. It is desirable to remove this poorly scanned area from the model rather than keeping it as part of the output. After each round of room partitioning, if none of the triangles in a room R_i are intersected by the scanner's path, then we infer that room has not been entered. The elements of R_i are removed from the interior triangulation T. Since the topology of the building model is changed, the set of room seeds is recomputed in this event and room labeling is restarted. This process also removes areas that are falsely identified as rooms, such as ghost geometry generated by windows and reflective surfaces, which cause rooms to be duplicated outside the actual model.

Figure 3.12 shows an example of the room refinement process for the hallways and classrooms in an academic building. Figure 3.12a shows the initial room seeds that were found based on circumcircle analysis. The hallways of this building are represented by several room labels, but after room label refinement as shown in Figure 3.12b, the hallways are appropriately classified. Additionally, rooms that are insufficiently scanned and represented with triangulation artifacts are removed from the model in the manner described above.

3.3.3 Simplification of Floor Plan Geometry

The interior building model is represented as a triangulation of wall samples, which densely represent the building geometry. In many applications, it is useful to reduce the complexity of this representation, so that each wall is represented by a single line segment. This step is often desirable to attenuate noise in the input wall samples or to classify the walls of a room for application-specific purposes. The goal is to simplify the wall geometry while preserving the general shape and features of the building model.

We opt to simplify walls with a 2D variant of Quadric Error Metrics (QEM) [23, 60]. Since this mesh is in the plane, only vertices incident to the model boundary are considered for simplification. The error matrix Q_v of each boundary vertex v is used to compute the sum of the squared displacement errors from each adjoining line along the boundary polygon. Since error is measured via distance away from a line in 2D, each Q_v has size 3×3 and is

$$Q_v = \sum_{l \in \text{lines}(v)} E_l \tag{3.3}$$

where E_l is defined from the line equation $ax + by + c = 0$ with $a^2 + b^2 = 1$,

$$E_l = \begin{bmatrix} a^2 & ab & ac \\ ab & b^2 & bc \\ ac & bc & c^2 \end{bmatrix}. \tag{3.4}$$

The simplification of the boundary proceeds in a similar manner to QEM, but if a wall vertex v is contained in multiple rooms or if it is connected by an edge to a vertex that is contained in multiple rooms, then it is not simplified. This constraint is used to preserve the fine details of doorways between rooms, while freely simplifying walls that are fully contained within one room. Wall edges are iteratively simplified until no simplification produces error of less than the original wall sampling resolution, r. Thus, walls are simplified while preserving the geometric features of the building interior.

As we are interested in preserving the 2D triangulation T of the building model, in addition to the boundary polygon, every edge simplification is performed by collapsing an interior triangle. This computation simplifies the boundary polygon of the model while still preserving the room labeling of the model's volume. These triangle collapses do not preserve the Delaunay property of the triangulation, but do preserve the boundaries between room volumes, which is more desirable in the output.

We can compare this method of simplification to the RANSAC fitting described in the previous method, in Section 3.2. By using local simplification, each modification to the floor plan geometry only requires access to a small number of elements, allowing for rapid simplification. With the RANSAC method, each adjustment is computationally expensive, since it must iterate several times over the entire set of input wall samples. The one limitation to the QEM simplification method is that it does not support circular or curved surfaces. This limitation is reasonable, however, since curved features can be approximated with a set of piecewise-planar walls.

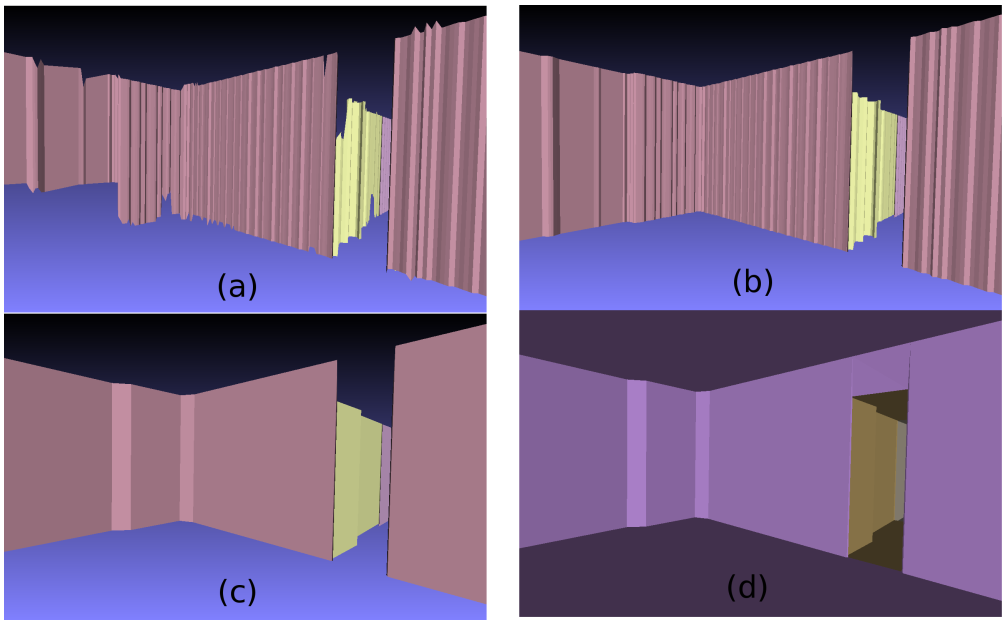


Figure 3.13: Example of creating a 3D extruded mesh from 2D wall samples: (a) walls of generated floor plan with estimated height ranges; (b) floor and ceiling heights are grouped by room; (c) simplification performed on walls; (d) floor and ceiling triangles added to create a watertight mesh.

3.3.4 Extruding Floor Plan Geometry to 2.5D Models

As mentioned in Section 3.1, each input wall sample also stores the vertical extent for the observed scans at that location. This information can be used to convert the labeled 2D interior building model to a 2.5D extruded model, by using the minimum and maximum height values for each scan as an estimate of the floor and ceiling heights, respectively [60].

Since these wall samples are collected by 2D planar scanners in an environment containing clutter, the minimum and maximum heights associated with each point are noisy. Figure 3.13a shows an example room with these initial heights. To produce aesthetically pleasing models, each room uses a single floor height and a single ceiling height. This assumption is reasonable since the goal of this processing is to produce a simplified building mesh. This step demonstrates the utility of room labeling to modeling. The height range for each room is computed from the median floor and ceiling height values of that room's vertices. An example is shown in Figure 3.13b and the corresponding result from the simplification process from Section 3.3.3 is demonstrated in Figure 3.13c.

The 2D triangulation of a room is then used to create the floor and ceiling mesh for that room, with the boundary edges of the triangulation extruded to create rectangular vertical wall segments. The result is a watertight 3D mesh of the building, capturing the permanent geometry in an efficient number of triangles. Figure 3.13d shows an example of this watertight extruded geometry, including the effects of wall boundary simplification on the resulting extruded mesh.

3.4 Results

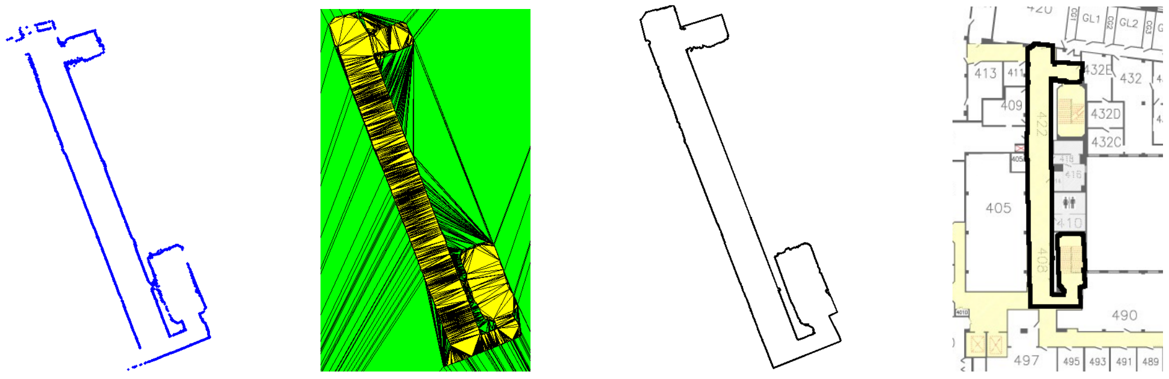
In this section, we show several results for each of the methods described in this chapter and compare these methods to each other and to ground truth models of the scanned environments. We first detail results of the Eigencrust floor plan generation method in Section 3.4.1. Next, we detail results of the room partitioning floor plan generation method in Section 3.4.2. Lastly, we compare these methods in Section 3.4.3.

3.4.1 Eigencrust Results

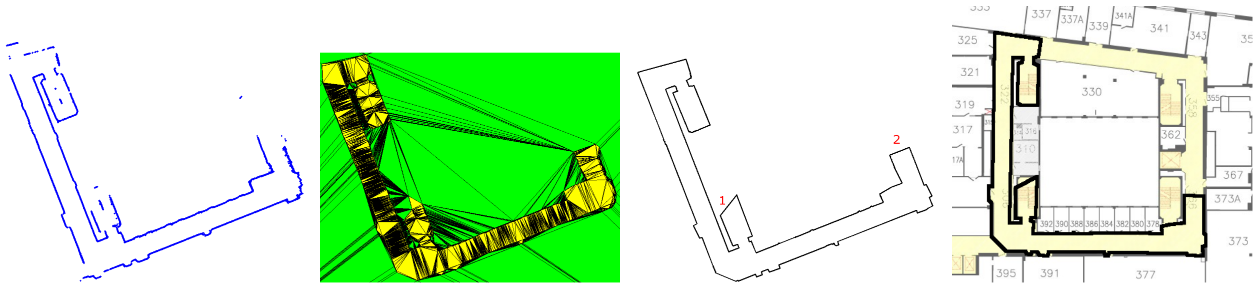
We apply the floor plan generation method discussed in Section 3.2 to a dataset representing the hallways in three stories of an academic office building, to demonstrate its accuracy as shown in Figure 3.14. The input point-set has 17.4 million triangles and the resulting wall samples for each story contain 5,544 samples, 5,357 samples, and 3,265 samples from bottom story to top, respectively. Our test code, written in unoptimized MATLAB and run on a personal laptop, processes these three stories in 371.4 seconds total. These results show how our algorithm enforces watertightness by fitting walls to the ends of any unscanned sections, such as the hallways labeled 1 and 2 in the third column of Figure 3.14c. The output models are compared against the current building floor plans, shown in the fourth column of Figure 3.14. In areas where the wall sampling is dense, our generated floor plan is accurate with respect to these ground-truth blueprints.

Figure 3.15 shows this same floor plan reconstruction method, applied to a scan taken of a single level of a hotel. The input point cloud has 5.8 million elements and the wall sampling produced 13,557 samples. This set demonstrates a model that is composed of both curves and straight line segments. Figure 3.16 shows the lobby of this building, which has been statically scanned. This dataset has an input point cloud of 2.7 million points from which 3,568 2D wall samples were extracted. This model demonstrates the ability of our algorithm to capture structural features such as columns. It shows how areas of high curvature can be modeled just as effectively as straight walls.

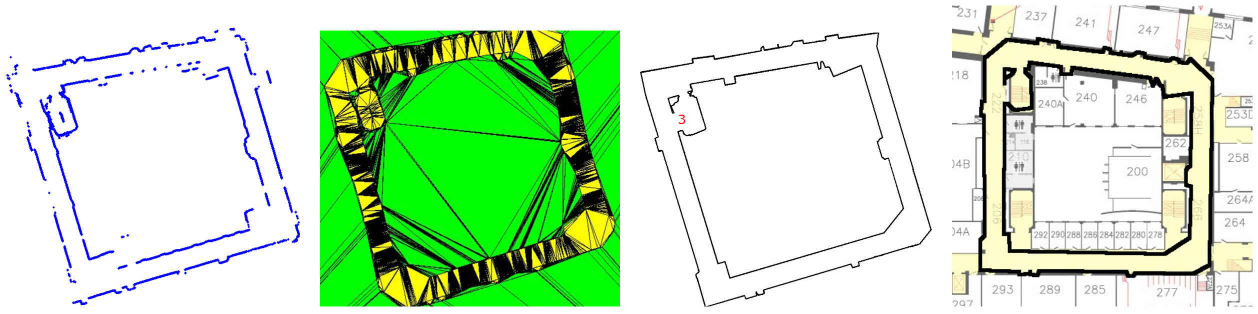
While enforcing watertight models allows for walls to be created in areas that are not scanned, it may also cause undesired homography changes. If two parallel walls are arbitrarily close, the space between them may become incorrectly labeled with neither wall existing in the final model. An example of this issue can be seen in the northwest stairway in Figure 3.14c. Another limitation is that the walls placed in areas of sparse scanning may be inaccurate. The corridor in the south-east corner of the floor plan in Figure 3.14c is not fully scanned, so a wall is represented at the edge of the scan area, even though the hallway continues much further in reality. While the sparsity of a scan is typically due to the placement of the scanning system, it can also be caused by the partitioning of the original point cloud into separate stories. Note that the southwest stairwell in the same model is sparsely sampled and thus the north wall is incorrectly modeled at an angle. In this case, the wall in question was thoroughly scanned, but higher in the stairway on the next story. The immediate partitioning of levels in a point cloud causes cross-level scans to become undersampled and introduces sparsity into the wall sampling.



(a)



(b)



(c)

Figure 3.14: Example floor plans on three-story building: (a–c) Processing of each story, with (left to right) wall sample locations, triangulation labeling, watertight curve-fit model, and comparison against ground-truth blueprints.

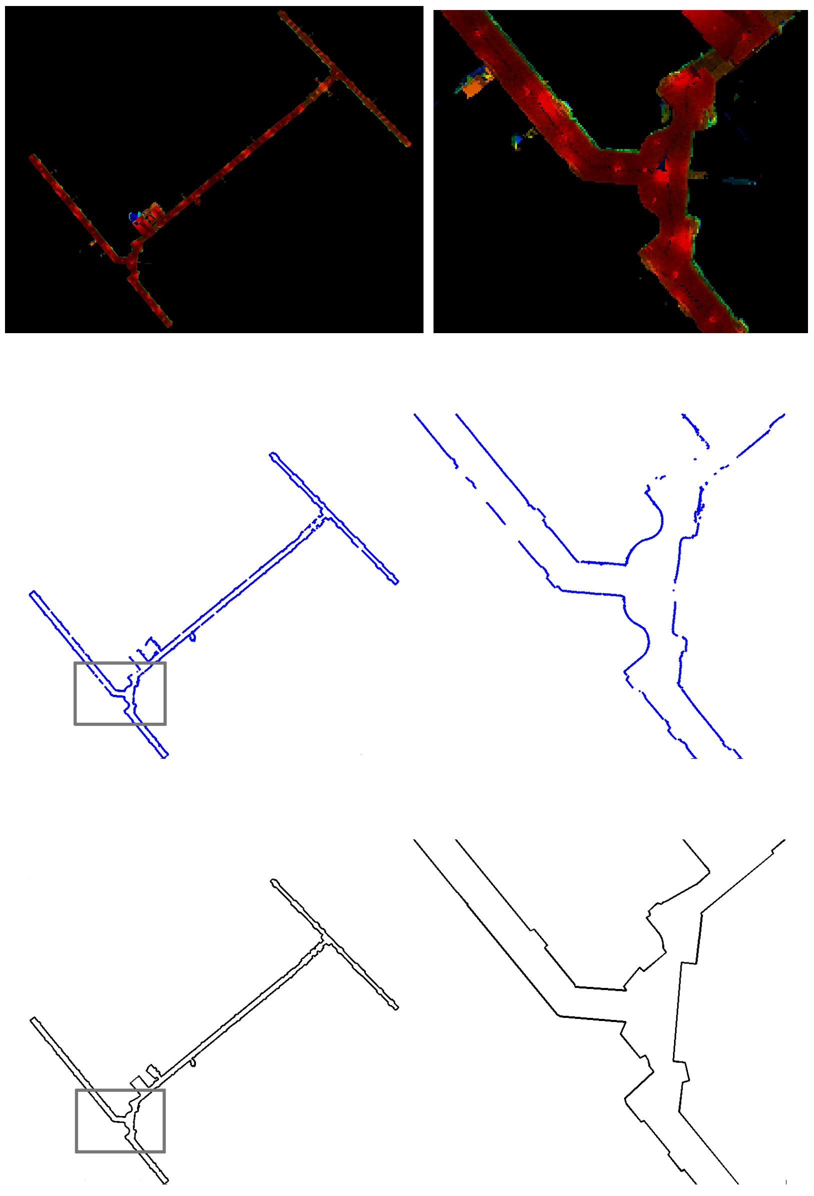
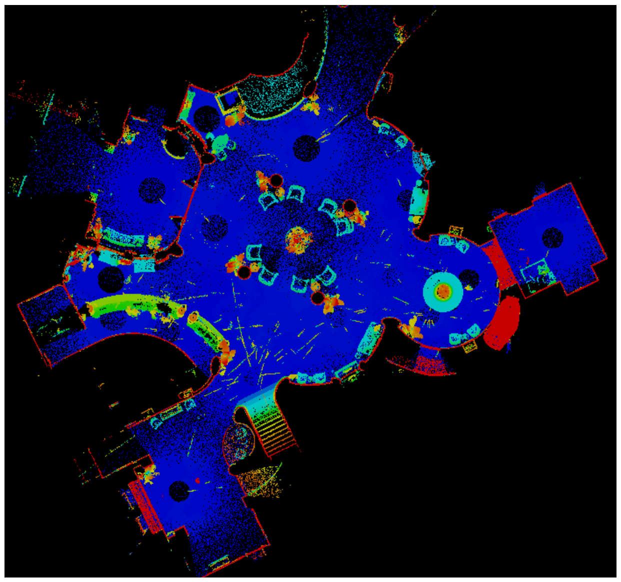
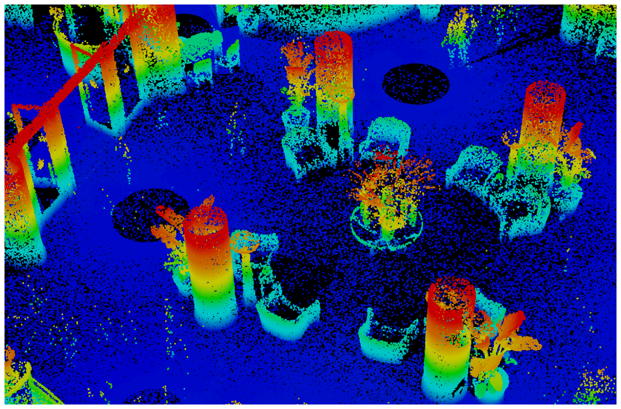

(a) (b)

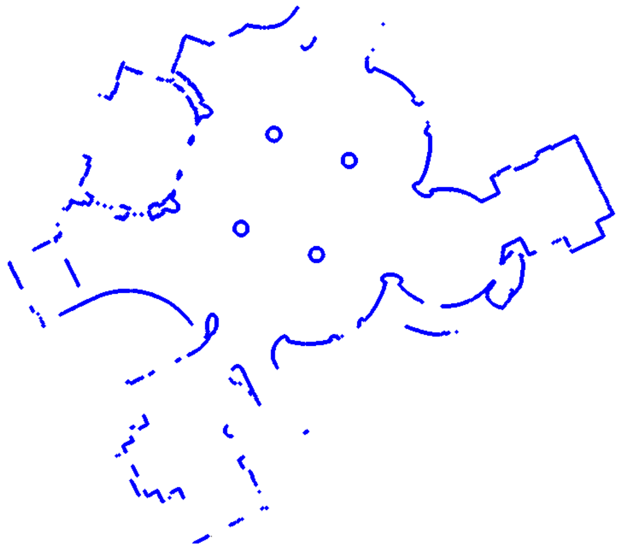
Figure 3.15: Hallway of hotel, captured with mobile scanner. (a) The full floor plan; (b) a close-up of a hallway intersection. The point cloud (top) was converted into wall sample locations (middle) and boundary edges were fit to these samples (bottom).



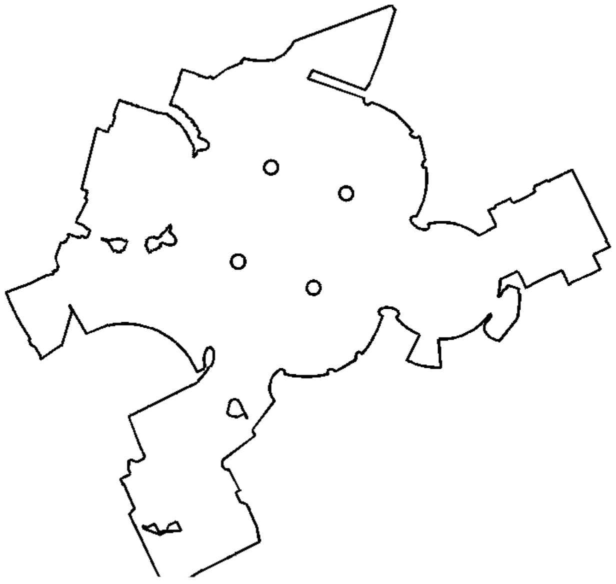
(a)



(b)



(c)



(d)

Figure 3.16: Hotel lobby, captured with a static scanner. (a) The captured point cloud; (b) a close-up of the center columns; (c) the wall samples extracted from this point cloud; (d) the set of wall boundary edges.

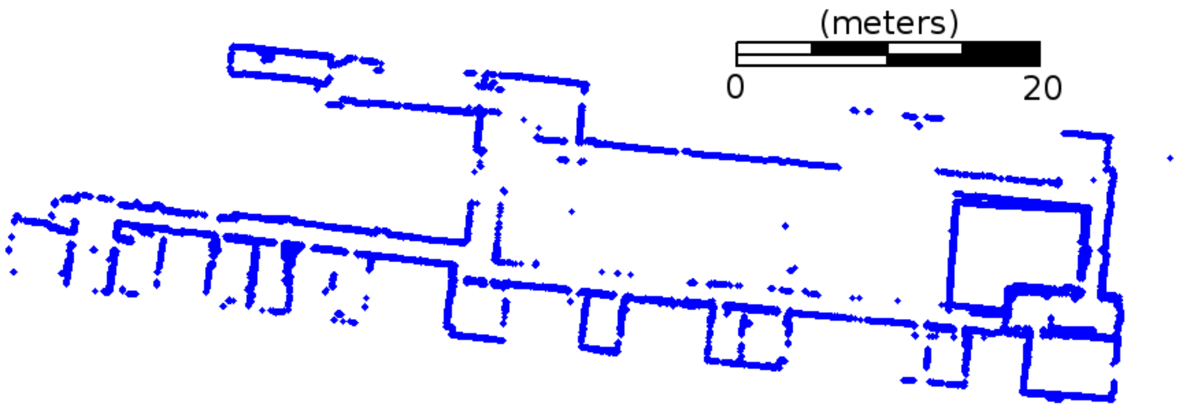
3.4.2 Room Labeling Results

The approach discussed in Section 3.3 works well on a variety of test cases, spanning several model types including offices, hotels, hospitals, and university buildings. For the largest models, total processing time to compute an extruded 3D model from 2D wall samples is under 10 seconds. Most of this time is spent on carving interior triangles, which can be performed in real time in a streaming manner during data acquisition, which typically lasts several minutes.

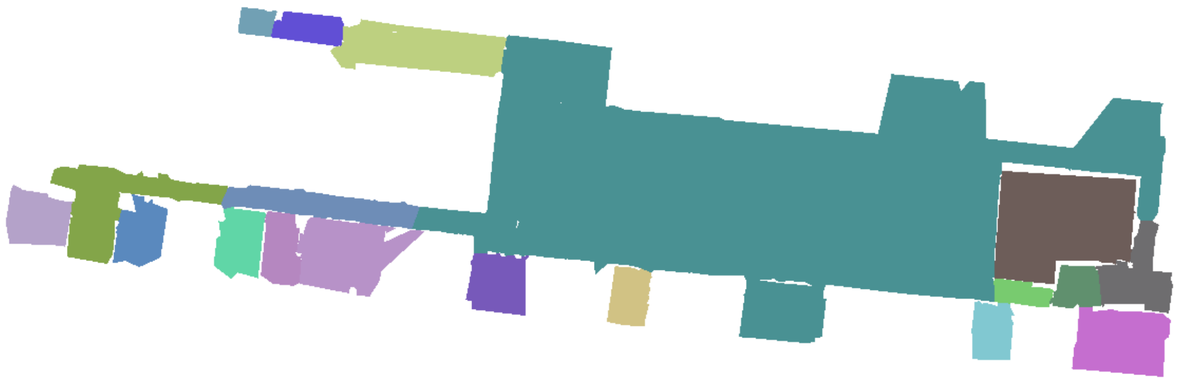
First, we show sample models generated by our method in five different environments. For all the models shown in Figures 3.17 through 3.21, the scale is in units of meters, and the resolution is 5 cm. Figure 3.17 models an office building, including cubicles and individual offices. The largest room in this model, shown in teal, containing cubicles. The cubicle walls do not meet our height threshold of $H = 2$ meters, so they are not captured by the wall samples. Since cubicles are not an architectural feature of the environment, this effect is desirable. The room shown in purple in the lower left corner of this model also exhibits an error in the building reconstruction. The adjacent room to the right was briefly seen through a window, but its area was considered part of this purple room rather than being removed in the manner described in Section 3.3.2, so a small extrusion remains in the model.

Figure 3.18 shows a small test model of an apartment office complex and Figure 3.19 shows a hotel lobby, hallways, and side rooms. Nearly all of this model is labeled as one room, consisting of the hallways of the building. Since no parts of these hallways are separated by doors, this result is desirable. Figure 3.20 represents an academic research lab, including conference rooms and student cubicles. The upper portion of the center room, shown in blue, is a kitchenette area, with a countertop. As the counter was not sufficiently captured by the wall samples, it is not represented in the 2.5D extrusion of the model. Figure 3.21 shows the hallways of an academic building.

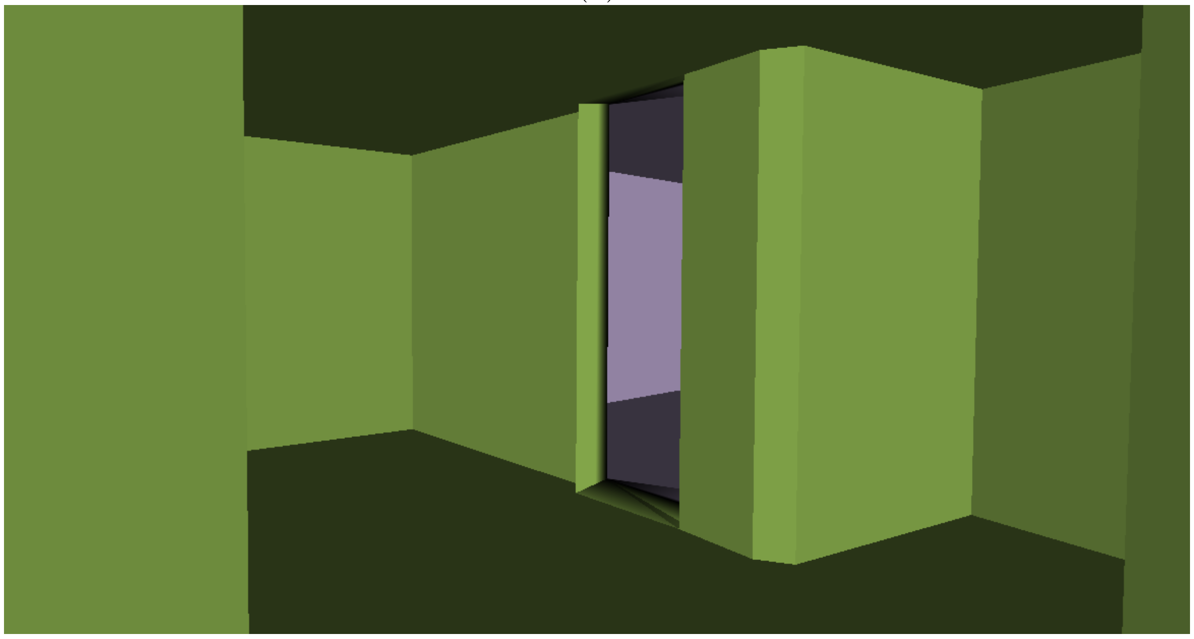
As these models were generated with a system that captures imagery in addition to laser range points, these models can also be texture-mapped with the scenery of the environment [85]. Figure 3.22 depicts the hallways of an academic building with and without texturing.



(a)

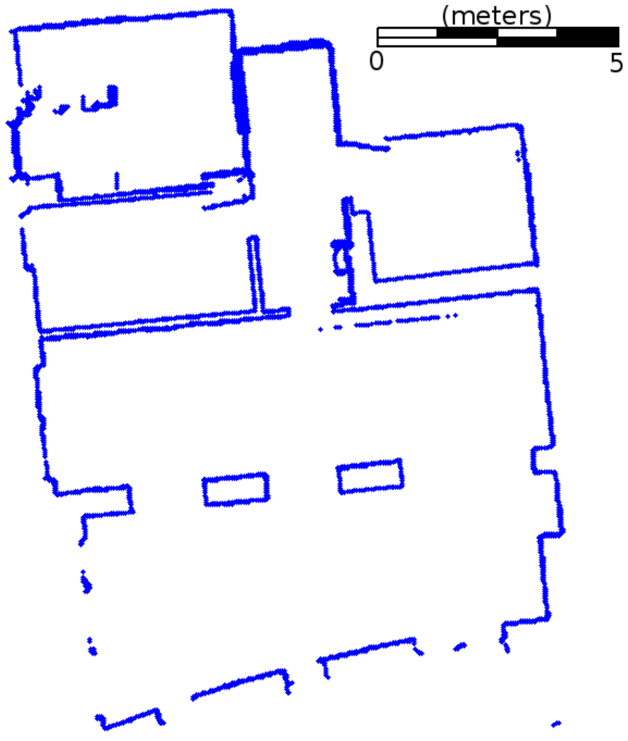


(b)

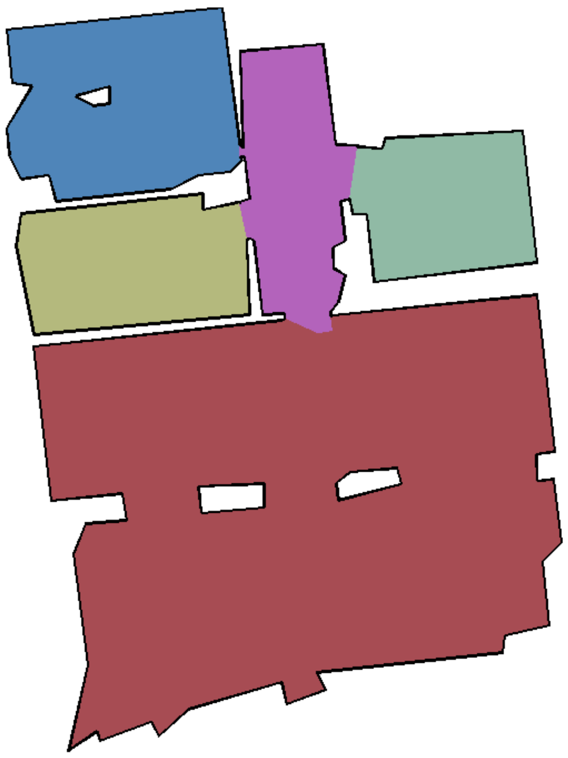


(c)

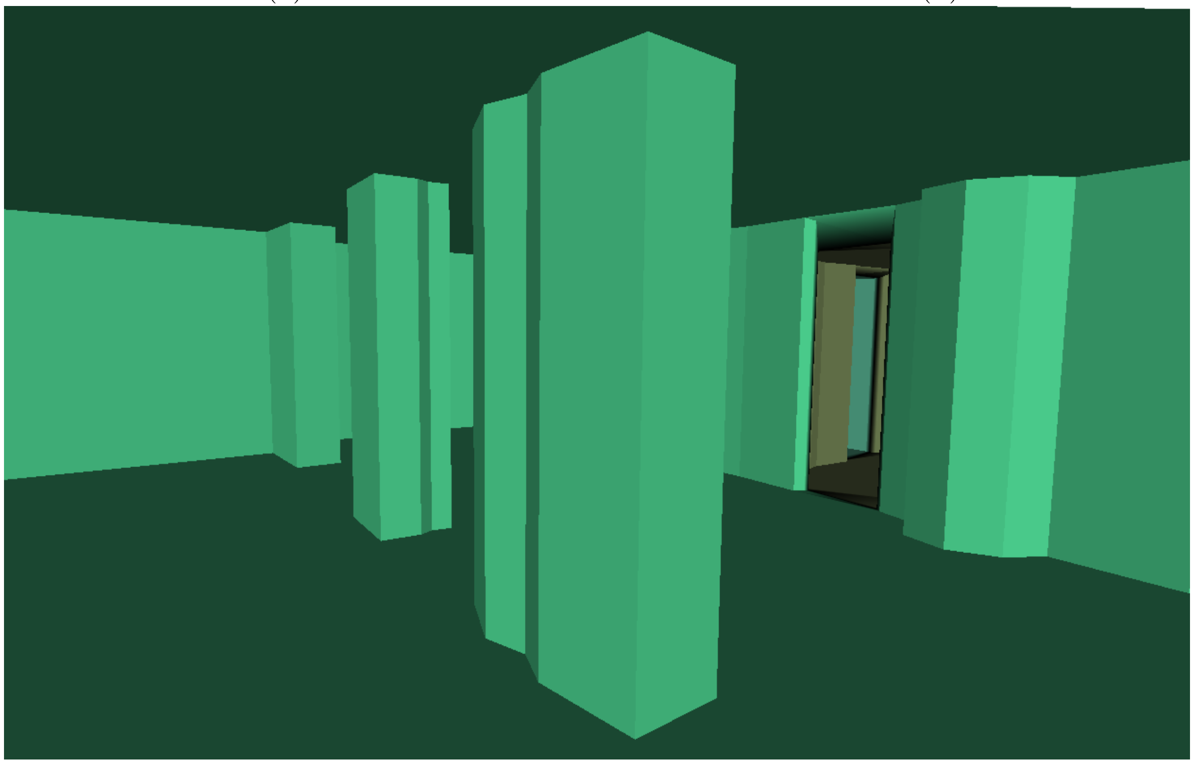
Figure 3.17: Office building: (a) Input represented by 12,823 wall samples; (b) generated floor plan with 19 rooms; (c) the extruded 2.5D mesh of the environment. The extruded 3D mesh is represented with 6,084 triangles. Total processing time required is 7.5 seconds.



(a)

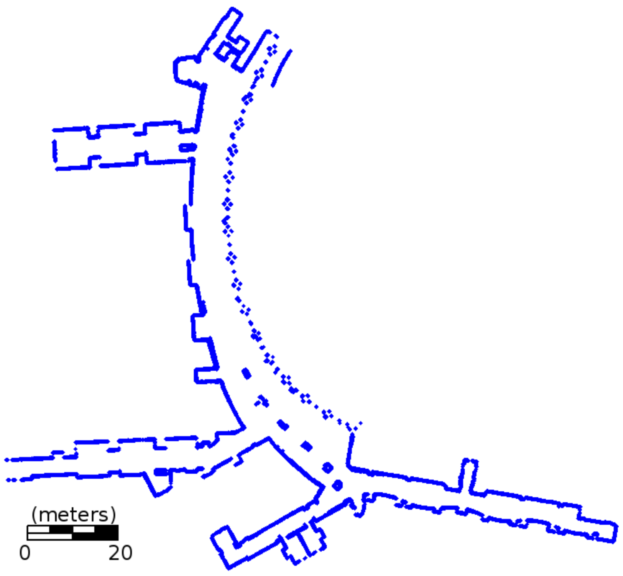


(b)

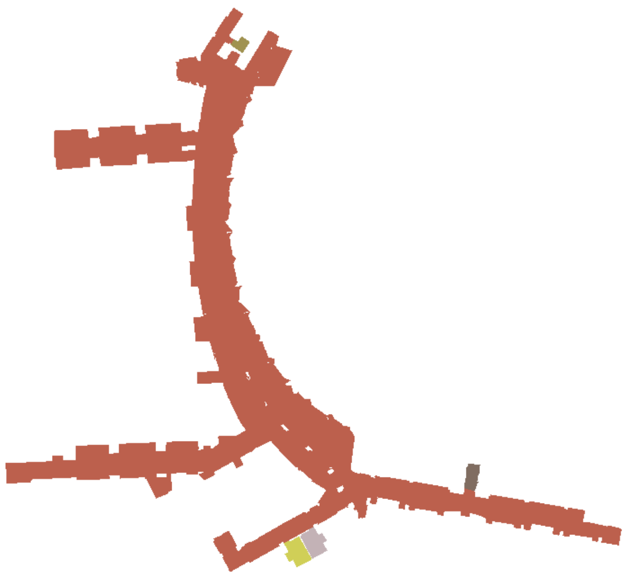


(c)

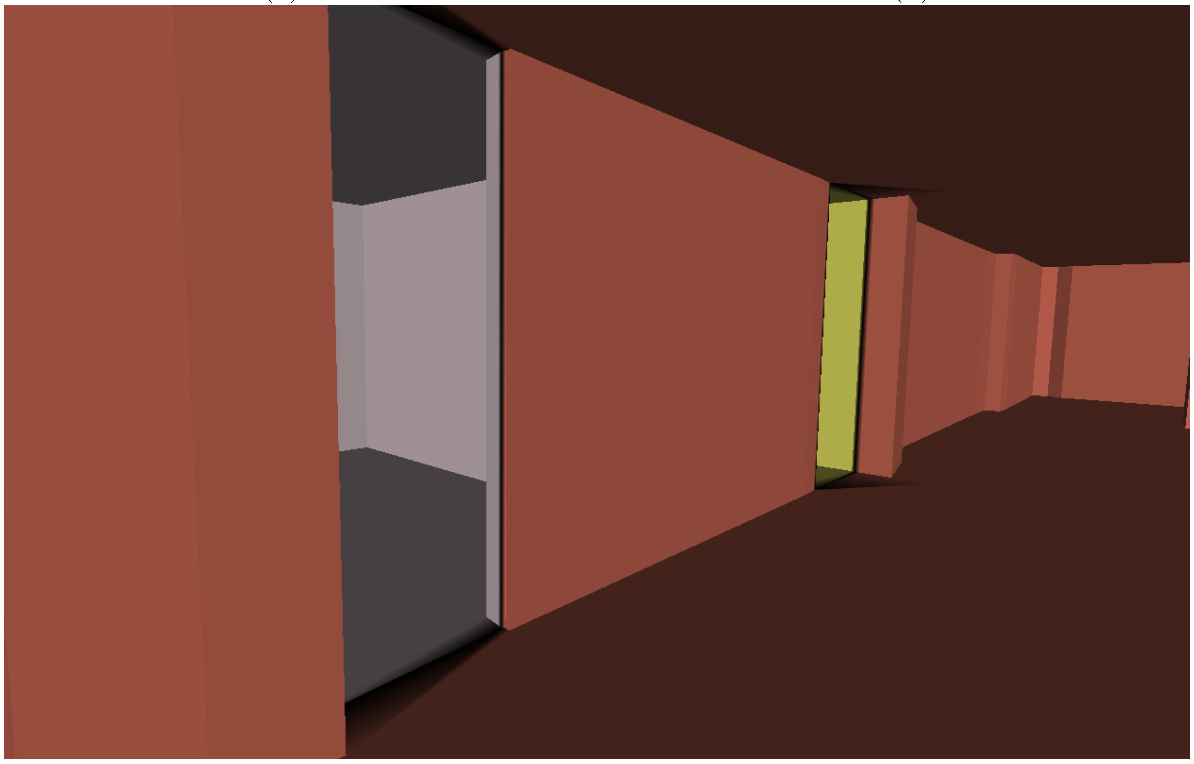
Figure 3.18: Apartment complex office: (a) Input represented by 3,462 wall samples; (b) generated floor plan with 5 rooms; (c) extruded 3D mesh represented with 512 triangles. Total processing time required is 1.2 seconds.



(a)

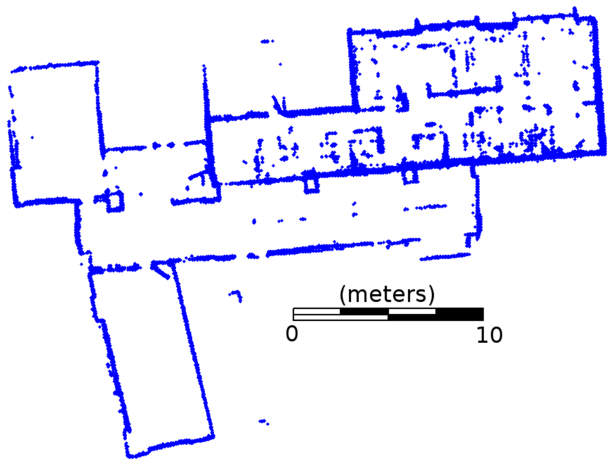


(b)

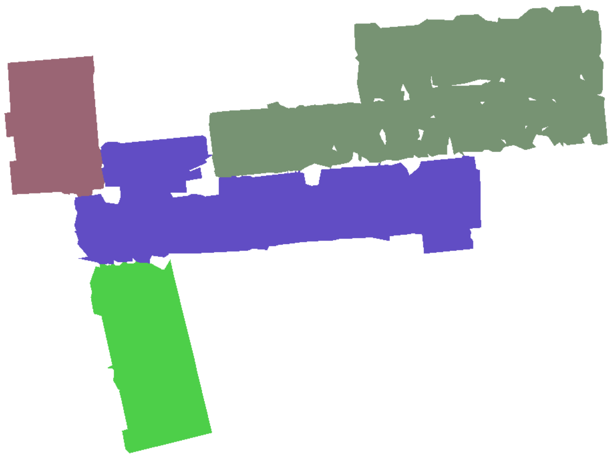


(c)

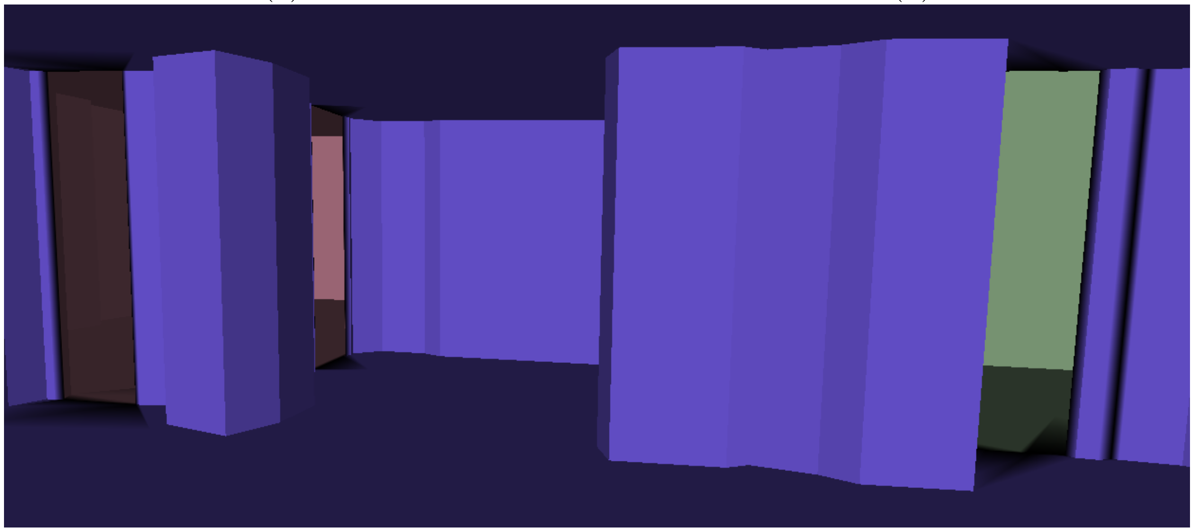
Figure 3.19: Hotel lobby and hallways: (a) Input represented by 33,582 wall samples; (b) generated floor plan with 5 rooms; (c) extruded 3D mesh represented with 5,012 triangles. Total processing time required is 8.5 seconds.



(a)

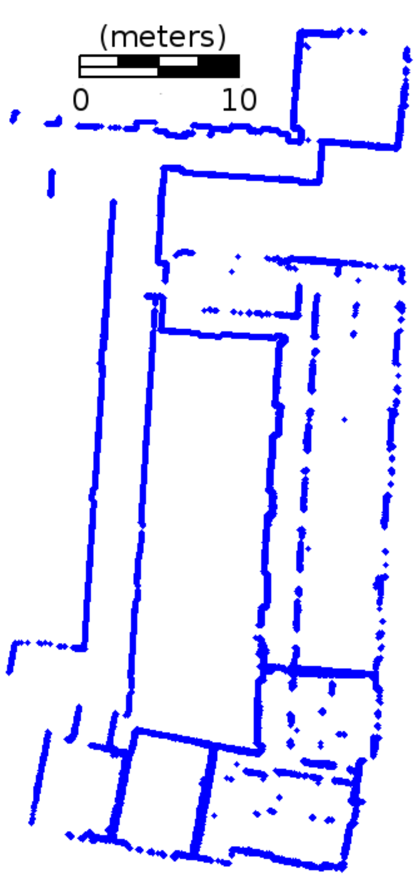


(b)

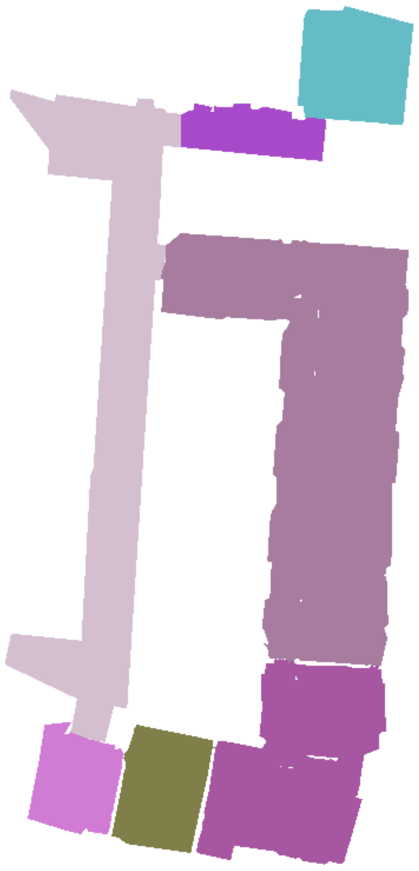


(c)

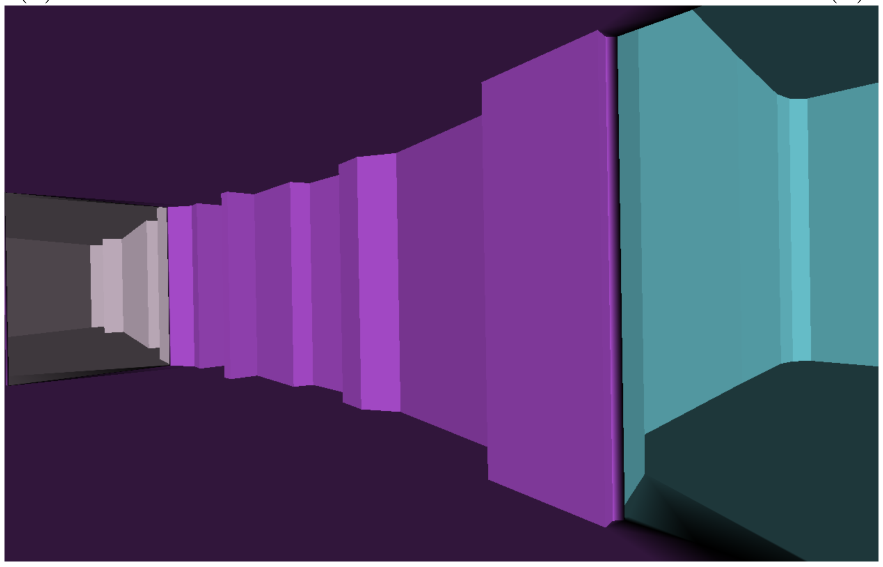
Figure 3.20: University office area: (a) Input represented by 12,183 wall samples; (b) generated floor plan with 4 rooms; (c) extruded 3D mesh represented with 4,912 triangles. Total processing time required is 7 seconds.



(a)

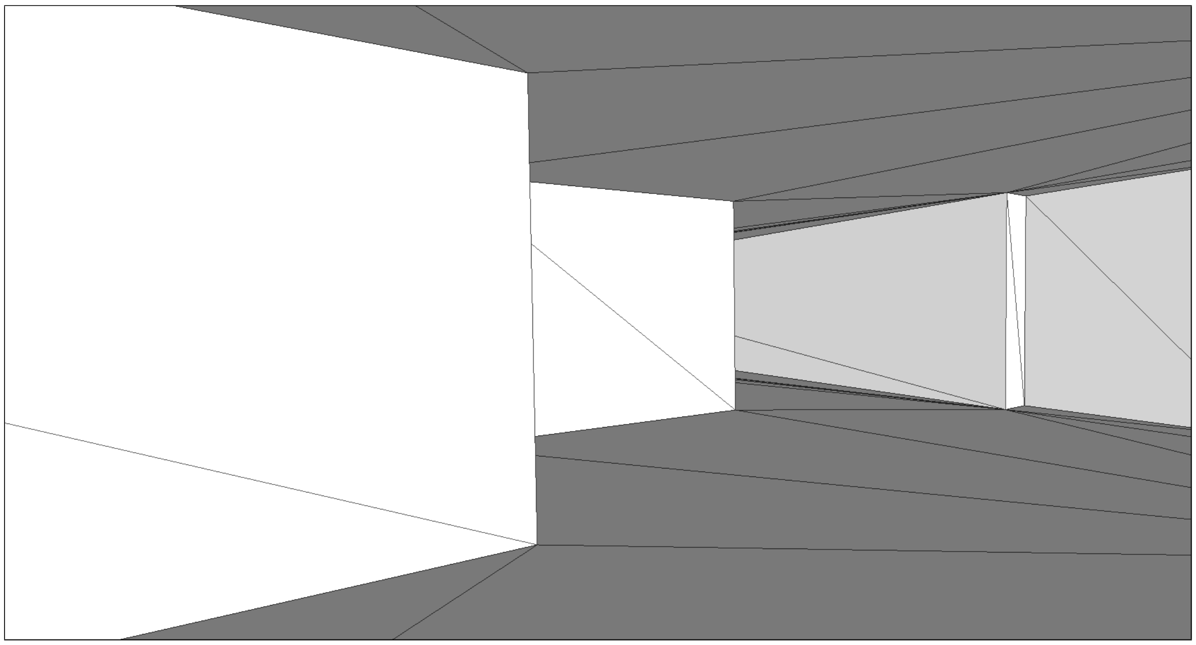


(b)

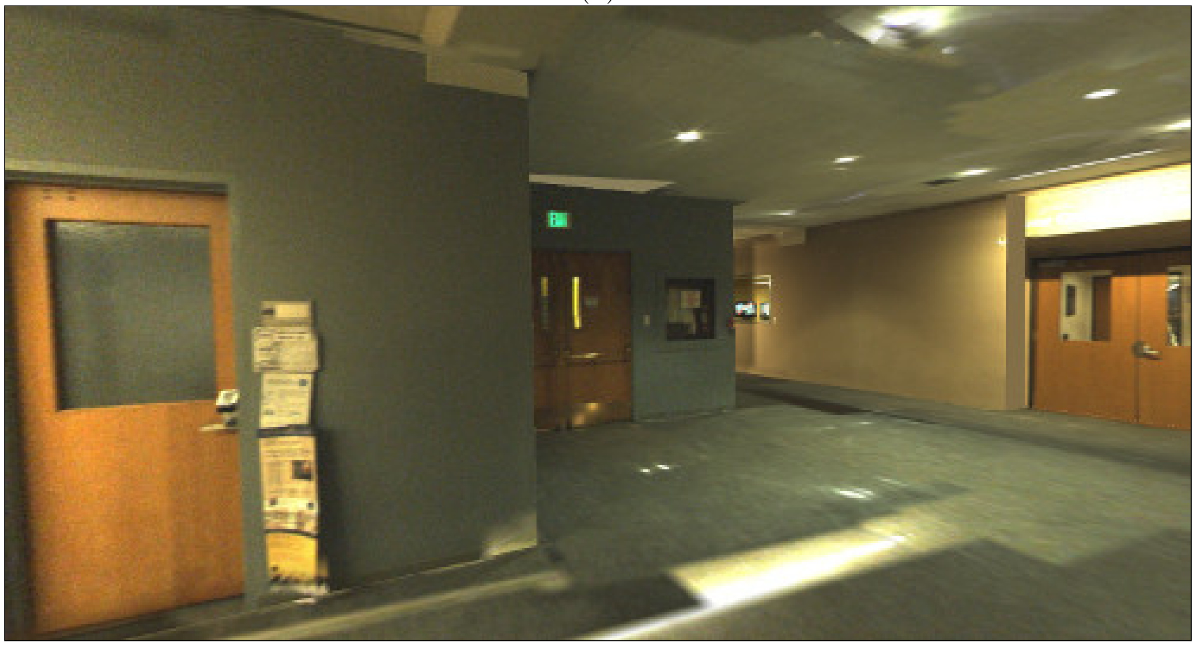


(c)

Figure 3.21: University office building: (a) Input represented by 12,125 wall samples; (b) generated floor plan with 7 rooms; (c) extruded 3D mesh represented with 3,604 triangles. Total processing time required is 4.5 seconds.



(a)



(b)

Figure 3.22: Interior view of 3D extruded reconstructed model: (a) without and (b) with texture-mapping [85].

3.4.3 Comparing Floor Plan Generation Algorithms

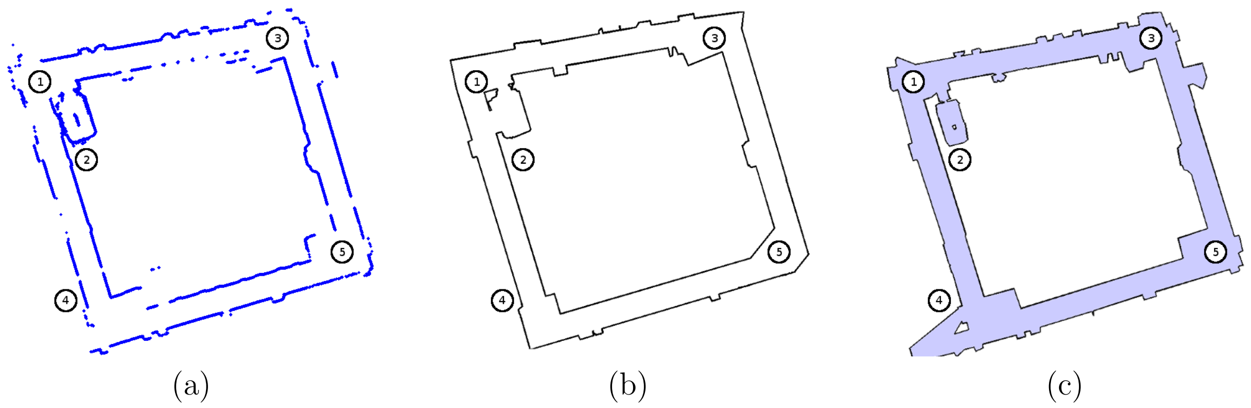


Figure 3.23: Comparison of the floor plan generation methods described in Sections 3.2 and 3.3: (a) the wall samples of an office hallway environment; (b) the floor plan generated by the method described in Section 3.2; (c) the floor plan generated by the method described in Section 3.3.

The floor plan methods described in this chapter use different techniques to accomplish the same goal of exporting 2D floor plan geometry. Although both methods are volumetric in nature, there are several areas where the accuracy of the output geometry differs. A comparison between the two methods is made in Figure 3.23. The input wall samples for an office hallway environment are shown in Figure 3.23a. From these wall samples, a floor plan was generated using both the methods described in Sections 3.2 and 3.3, shown in Figures 3.23b and 3.23c, respectively [59, 60].

The outputs of these methods are noticeably different in several locations. First, the location marked (1) indicates a corner of a hallway. Since the floor plan method described in Section 3.2 performs explicit line fitting on the output walls, the corner of this hallway is much simpler than when compared to that same location on the output of the method described in Section 3.3. However, this simplification produces a loss of the inlet geometry of doorways in that hallway. Next, the location marked (2) shows a dramatic topology error in Figure 3.23b, whereas the topology is correct in Figure 3.23c. This error is due to the Eigencrust computation performed by the former method, which can lead to topologically inconsistent labeling of narrow features such as a thin wall between two rooms. Location (3) is similar to the location marked by (1): a hallway corner that was oversimplified in Figure 3.23b yet preserved in Figure 3.23c. In this case, not only has the line-fitting process removed doorway inlet geometry, but this geometry has been replaced by corners that are no longer the correct angle or orientation. Another important distinction between the two models is shown at location (4), where a portion of a hallway extending to the left was scanned, but not fully traversed. The output shown in Figure 3.23b removed all geometry of this hallway, due to the lack of density in wall samples, yet the model shown in Figure 3.23c preserved some of this hallway, showing that it is less sensitive to wall sample density. Lastly, position (5) indicates one of the downsides of aggressive line fitting. In Figure 3.23b, line fitting was used, which caused corners to be clipped and replaced by walls at a 45-degree angle. Figure 3.23c shows this same geometry, but its simplification is less aggressive,

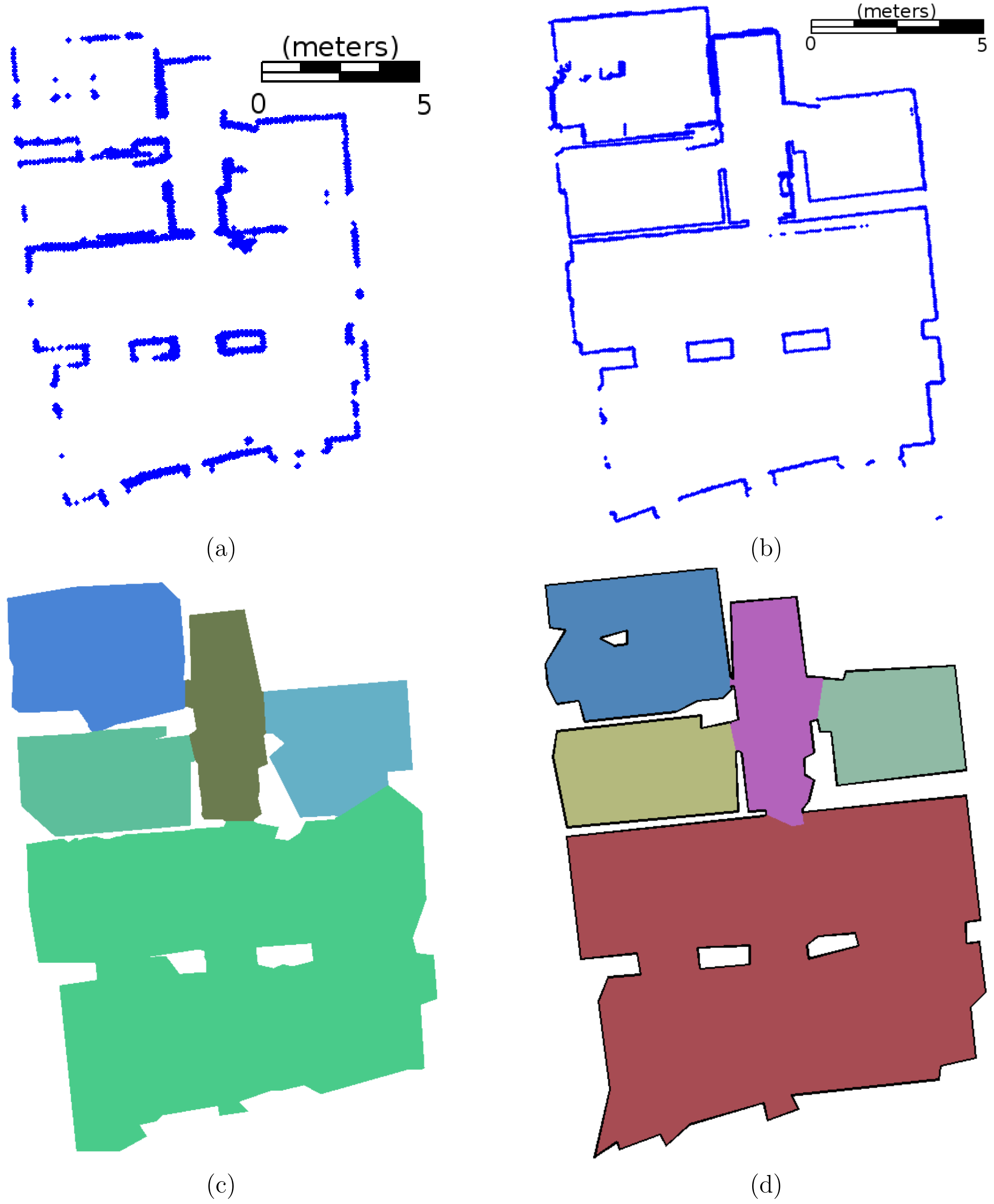


Figure 3.24: Comparison between floor plans generated with wall samples from point clouds and particle filter grid-maps: (a) Wall samples of a small office complex generated from 3D point cloud; (b) wall samples generated from particle filter grid-map; (c) floor plan generated with wall samples from (a); (d) floor plan generated with wall samples from (b).

producing more faithful geometry.

There are several advantages to the latter floor plan generation scheme [60]. In addition to generating watertight geometry of the scanned environment, it also partitions the geometry into separate rooms. This method also preserves height information of the walls of the floor plan, allowing the 2D geometry to be extruded into a 2.5D mesh. Since these attributes are not present in the previous method [59], no comparisons can be made for these features.

As mentioned in Section 3.1, floor plan geometry is generated from an input of wall sample positions. These wall samples can be generated either from a 3D point cloud or from the 2D grid-map generated by the particle filter of the localization procedure [54]. While either of these methods can produce wall samples, each has their benefits and detriments.

Figure 3.24 shows a comparison between floor plans generated using these two methods of computing wall samples. In Figure 3.24a, we show wall samples generated from a 3D point cloud acquired by scanning the environment, with the corresponding floor plan shown in Figure 3.24c. In Figure 3.24b, we show wall samples generated from the 2D particle filter grid-map of the same environment, with the corresponding floor plan shown in Figure 3.24d. Note that the 2D floor plan generated using the particle filter grid map is much cleaner, with sharper corners and fewer topological errors. The wall samples generated from the point cloud data are likely to miss some wall features if they are obstructed by non-planar objects, such as furniture or shelves occluding the wall area. However, the particle filter grid-map version of the wall samples shown in Figure 3.24b do not contain any height information, since they are strictly 2D data. As such, they are unable to accurately produce an extruded 2.5D model of the environment. In Section 5.3, we discuss methods to have the best of both worlds: floor plans that have accurate and clean features while still retaining full 3D data.

We demonstrate efficient approaches to automatically generate floor plans of building interiors at real-time speeds. Classifying and labeling the rooms within each generated floor plan allows for simplification schemes that can preserve fine details at doorways. These room labels allow for accurate 2.5D extrusion from noisy floor and ceiling height estimates of the input points. The resulting model is suitable for visualization, simulation, and navigation applications. Current limitations of this algorithm include the verticality assumption made about observed building features. If the horizontal cross-section of an environment changes dramatically between different heights, the modeling techniques presented in this paper do not accurately portray the actual geometry.

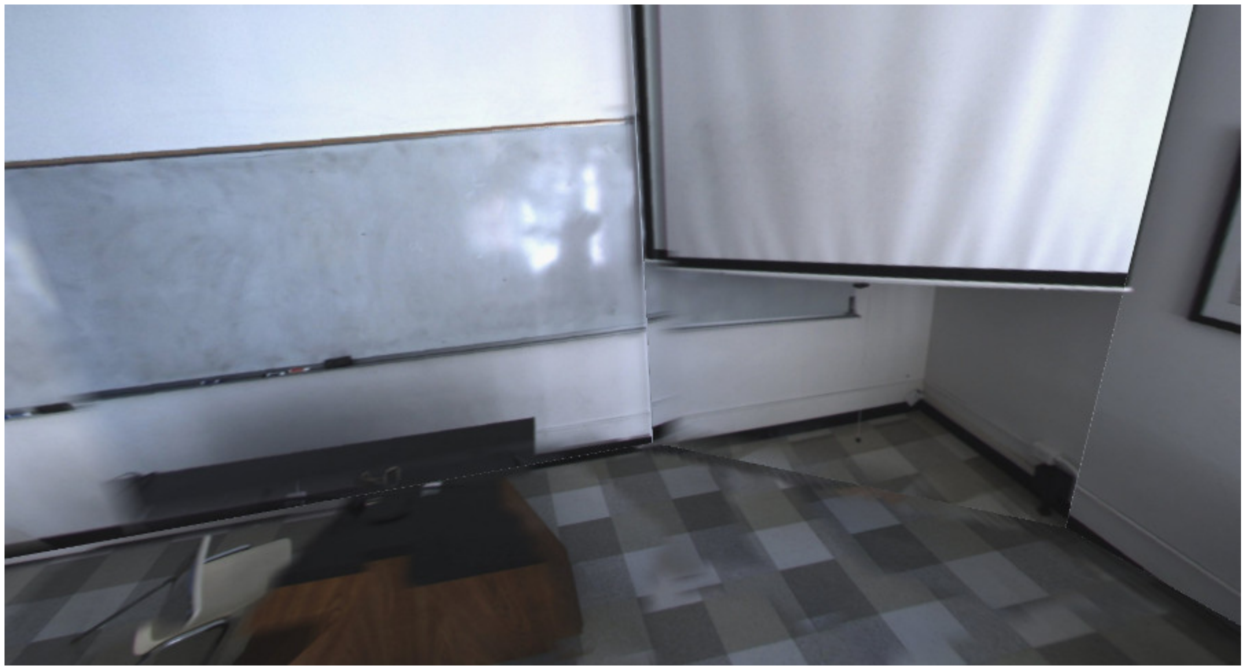
3.5 Limitations

Generating a 3D model by extruding a 2D floor plan entails assumptions about the building geometry, which can impose limitations in the accuracy of the final model. This method assumes the building environment has a 2.5D nature, in which all surfaces are either perfectly vertical or horizontal. This method also assumes a single height per room, which means drop ceilings and other vertical variations are not captured.

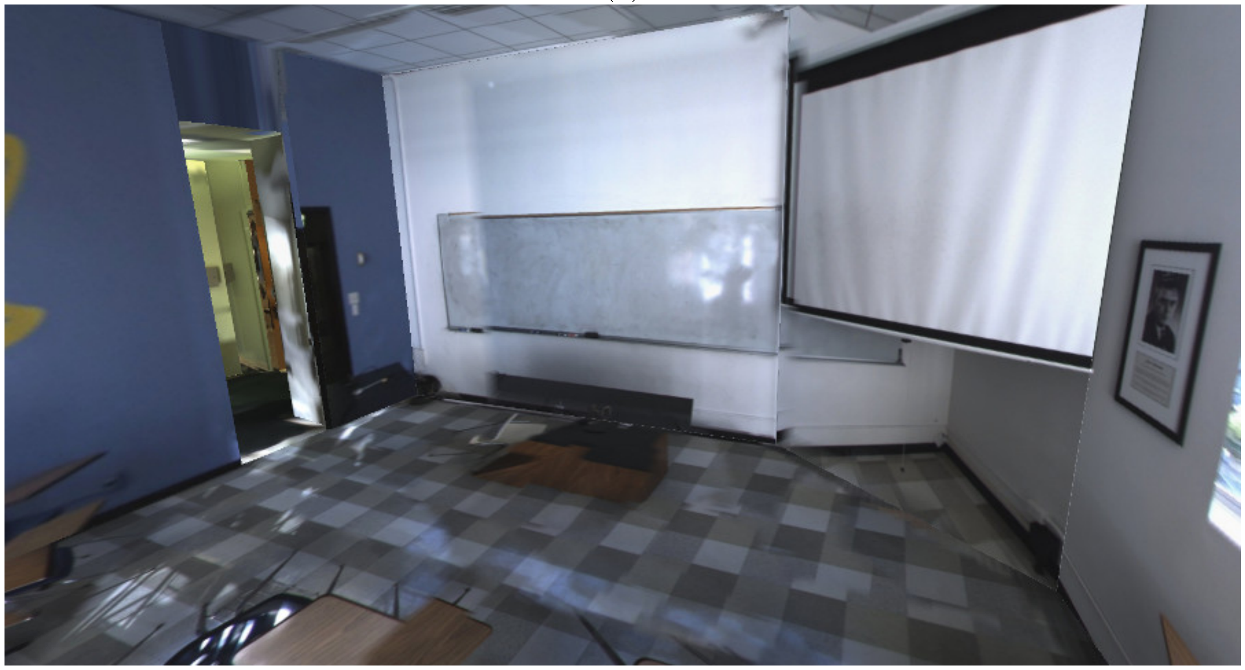
This reduction is especially noticeable in the lack of furniture geometry. As we discuss in Chapter 5, the lack of furniture in a 2.5D extruded model can be useful for further processing, but as a final model it presents some challenges. We can observe this limitation in Figure 3.25. This figure shows a classroom environment, represented by a texture-mapped

2.5D extruded floor plan model. The classroom contains several pieces of furniture as well as a projector screen in the front. Since the projector screen is a large vertical surface, it was classified as a “wall” and included in the model geometry. This model geometry represents the screen extending fully to the floor, whereas it does not do so in reality. As such, the texture-mapping of this geometry is incorrect in the lower section, since the texture does not match the reconstructed geometry. While the texture looks reasonable from the angle viewed in Figure 3.25a, the change in angle in Figure 3.25b highlights this inaccuracy.

This model also shows how the lack of furniture in the reconstruction geometry causes the texture of furniture from the imagery to be projected onto the floors and walls, causing distortion. These models were texture-mapped by the techniques described by Cheng et al. [13], which assumes that any geometry observed in the camera imagery is correctly represented in the reconstructed model. As this model is missing furniture geometry, the texture becomes distorted.



(a)



(b)

Figure 3.25: A texture-mapped extruded floor plan of a classroom environment: (a) the front of the room; (b) the same view from a different angle. Note that the projection screen is represented as a vertical surface extending from floor to ceiling in the 2.5D extrusion. Due to this limitation in the geometry representation, the texture on the lower portion of this surface is incorrect.

Chapter 4

Complex 3D Model Generation

The methods described in this chapter compute fully-detailed 3D meshed surfaces from the raw scans taken from our ambulatory scanning system discussed in Chapter 2. The goal of these approaches is to preserve all detail of the scanned environment. Such models are important for visualization and analysis purposes, where detail is more important than minimalism of representation. These models are also used to aid in the processing of indoor navigation algorithms, which require accurate depth maps [3]. Objects within the environment, such as furniture and light fixtures, are preserved in these methods. As we discuss in Section 5.6, these objects can be explicitly detected and modeled separately.

This chapter describes two separate algorithms for this task. Both methods generate watertight 3D meshes that utilize adaptive triangle sizes to ensure efficiency of representation. In Section 4.1, we discuss probabilistic analysis on the raw data scans, which can be used to aid the volumetric labeling of the model environment. Next, in Section 4.2, we detail our first complex carving technique, which represents the environment volumetrically by an advancing voxel boundary. We refer to this technique as "voxel carving", which produces a piecewise planar, watertight model of the interior environment of the building [93]. In Section 4.3, we introduce a new method that extends the voxel carving method by introducing the probabilistic models discussed in Section 4.1. This method performs the volumetric labeling with an octree structure rather than voxels, to allow for adaptive resolutions within the model and to allow efficient parallel processing. Again, the method generates a watertight mesh of the environment that can be represented via piecewise planar surfaces. Lastly, in Section 4.4, we show comparative results of these two methods on real-world data.

4.1 Statistical Models of Input Scans

In this section, we discuss how each input scan point is analyzed to generate a volumetric occupancy model of the scanned environment. Our goal is to convert the input set of laser scan points into a labeling of space where each location $\vec{x} \in \mathbb{R}^3$ is assigned a likelihood of being *interior* or *exterior*. We perform this task in two steps. The first step is to form a probabilistic model of each scan point's position, coupled with the position of the originating sensor. We discuss sources of noise and the probability model for each scan point in Section 4.1.1. The second step is to use this probability model to estimate the scan point's vote for the *interior*

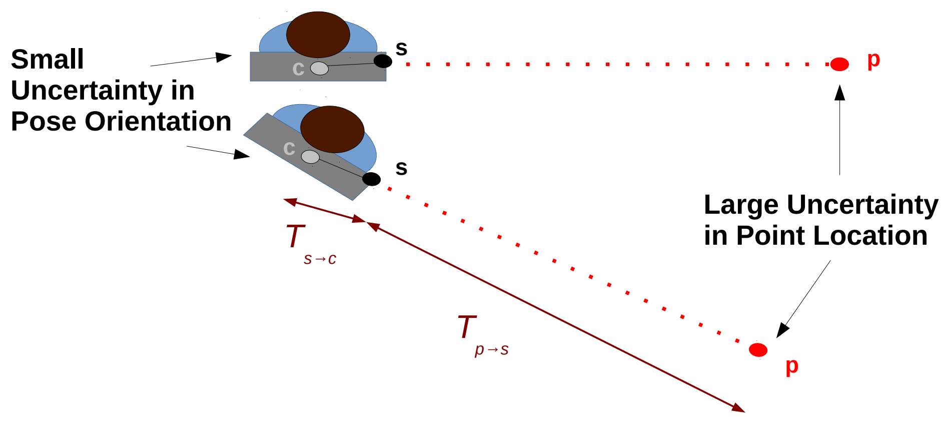

Small Uncertainty in Pose Orientation
c
s
p
c
s
$T_{s \to c}$
$T_{p \to s}$
Large Uncertainty in Point Location
p

Figure 4.1: Any error or noise in the localization process greatly affects the accuracy of scan point positions. A small variation in the orientation of a sensor in space can dramatically change the computed 3D position of the point.

likelihood of each location in space intersected by its scan ray. We refer to this step as "carve mapping", and detail its application in Section 4.1.2.

Once we obtain these estimates for each scan point, it is possible to generate an occupancy model of the entire scanned environment, as discussed in Section 4.3. Note that the method used in Section 4.2 simply takes the maximum likelihood position of each scan point, and does not consider the distribution of the points' positions during volumetric carving.

4.1.1 Gaussian Model of Point Noise

We compute an estimate of the 3D positions for each scan point and the point's originating sensor. These values are represented as two 3D Gaussian distributions. For each input scan, the sensor position is represented by Gaussian $N(\vec{\mu}_s, C_s)$ and the scan point position is represented by $N(\vec{\mu}_p, C_p)$. For tractability, we assume the scan frame's distribution is independent from the positions of the other scan frames.

The uncertainty in the position values of each scan point originate from three independent sources of error: the backpack localization estimate, the timestamp synchronization, and intrinsic sensor noise. Each of these noise sources affects the system's estimate of its position and orientation in different manners, but all must be accounted to generate accurate measurements in world-coordinates for the model.

Localization noise. Localization noise arises from errors in the estimate of the system trajectory, as detailed by Corso and Zakhor [54], and is by far the largest source of error with typical standard deviations on the order of 20 centimeters. The localization procedure is broken into two steps: dead-reckoning via odometry estimates and global optimization of path [87, 94]. Dead-reckoning is performed by estimating the incremental transformations between successive poses of the system, integrating measurements of the velocity and acceleration to determine a rough position estimate. Since this method represents open-loop integration, such measurements are susceptible to drift based on any system bias [45, 52]. There exist real-time systems that perform optimizations via secondary measurements of movement, such as laser scanners or visual cameras [95], which reduces the error of the dead-reckoning path. Such improvements to the dead-reckoning path are important, since global optimization redistributes any error within a pose uniformly across the traversed path [88].

Variations in uncertainty between poses are due to the optimized constraint-network of loop closures in the estimated path of the system. Such optimization systems provide a covariance matrix representing the uncertainties of the values for each pose in the trajectory path. These uncertainties depend on the timestamp, t, since the system moves around the path over time. These values include both position and orientation of the scanning system at a given time. The covariance of the system's world position (x, y, z) at a given time t is denoted by the 3×3 covariance matrix $C_{\text{pose}_T}(t)$. Similarly, the covariance of the system's orientation (θ, ϕ, ψ) at time t is denoted by 3×3 covariance matrix $C_{\text{pose}_R}(t)$. The uncertainty in the orientation of the sensor can cause major positional uncertainty for range measurements, as shown in Figure 4.1, and it is important to take it into account. While there are some cross-correlations between the position and orientation terms, these are ignored for tractability.

The next step is to combine these covariances to estimate the uncertainty of the position of a given scan point p taken at time t, and the corresponding position of the originating sensor, s. First, the uncertainty in the position of the sensor s due to localization noise is estimated to be

$$C^{(s)}_{\text{pose}}(t) = C_{\text{pose}_T}(t) + [T_{s\to c}(t)]_x \; C_{\text{pose}_R}(t) \; [T_{s\to c}(t)]_x^T \tag{4.1}$$

where $[.]_x$ denotes the cross-product matrix for a given vector. This equation combines the uncertainty of the pose translation given by the localization process, $C_{\text{pose}_T}(t)$, with uncertainty in the position of the sensor caused by the pose's rotation, $C_{\text{pose}_R}(t)$. This equation uses the small-angle approximation to apply the covariance of rotations specified by $C_{\text{pose}_R}(t)$ to the lever arm described by the translation between the system common coordinate frame and the sensor position, $T_{s\to c}(t)$. Since we assume the uncertainty in these orientations is small, this approximation is valid. Since the sensors are rigidly mounted to the system, this distance is independent of time, but the displacement in world coordinates depends on time due to the system changing orientation over the course of the data acquisition. The final value represents the covariance of the position of sensor s, in model coordinates. Similarly, we can compute the effect of the localization uncertainty on the position of scan-point p,

$$C^{(p)}_{\text{pose}}(t) = C_{\text{pose}_T}(t) + [T_{p\to c}(t)]_x \; C_{\text{pose}_R}(t) \; [T_{p\to c}(t)]_x^T \tag{4.2}$$

where $T_{p\to c}(t)$ denotes the translation between the point p and the system common position at time t. The lever arm of this translation can be affected by the uncertainty in the rotation specified by the localization pose, $C_{\text{pose}_R}(t)$. The value $C^{(p)}_{\text{pose}}(t)$ specifies the uncertainty in the scan point position, in world coordinates, due to estimated noise from localization.

Timestamp synchronization noise. Timestamp synchronization errors occur because the hardware system combines measurements from several sensors, whose timestamps need to be transformed to a common clock. Since each sensor operates independently on a separate clock, it is important to compute the transformation from a given sensor's time frame to that of the common system. This synchronization is done with a linear fitting of the individual sensors' clocks with the common system clock and produces an estimate for the standard deviation of each sensor's synchronization, σ_t. Note that σ_t is constant for a given sensor across the entire dataset, since it is derived from a linear fit of all sensor frames. Mis-synchronization of timestamps can produce spatial errors of scan points, especially when the system is moving or rotating rapidly while scanning distant objects. In these cases, an estimate of the scan point's position changes depending on our estimate of when a scan is taken. However, since our sensors are synchronized to an accuracy of $\sigma_t \approx 1$ millisecond, synchronization error is usually the lowest source of noise in the scan points, contributing uncertainty to scan point positions of under 1 centimeter. We represent the uncertainty in the sensor position estimate and the scan-point position estimate at a given time t caused by the time synchronization process as zero-mean Gaussians with 3×3 covariance matrices $C_{\text{ts}}^{(s)}(t)$ and $C_{\text{ts}}^{(p)}(t)$, respectively

$$C_{\text{ts}}^{(s)} = [T_{s\to c}(t)]_x \; \sigma_t \, \vec{\omega}(t) \, \vec{\omega}^T(t) \, \sigma_t \; [T_{s\to c}(t)]_x^T \, , \tag{4.3}$$

$$C_{\text{ts}}^{(p)} = [T_{p\to c}(t)]_x \; \sigma_t \, \vec{\omega}(t) \, \vec{\omega}^T(t) \, \sigma_t \; [T_{p\to c}(t)]_x^T \, , \tag{4.4}$$

where $\vec{\omega}(t)$ represents the estimated angular velocity of the system at time t. This equation denotes how much uncertainty exists in the sensor and point positions based on the rotational velocity of the system. If the system is rotating rapidly and there exists an uncertainty in the timestamping of the sensor frames, then the resulting position of the sensor or the scan point could vary greatly. Note that this model only considers rotational velocity, and not linear velocity. This simplification is due to our estimates of linear velocity being more difficult to formulate, since they cannot be directly measured. Additionally, rotational velocity produces the dominating source of noised caused by timestamp mis-synchronization, since the estimated position of a point scanned far away is affected more by rotation of the sensor than its translation.

Sensor hardware noise. The third source of noise depends on the sensor hardware. Our system uses Hokuyo UTM-30LX sensors, whose intrinsic noise characterization is given by Pomerlaeu et al. [78]. These sensors are popular among robotics groups due to their low noise characteristics relative to their range [96]. Typically this noise contributes on the order of 1 to 2 centimeters to the standard deviation of the positional estimate of scan points. This uncertainty value increases as the range distance of the point increases, with accuracy dropping dramatically after 10 meters and accurate measurements stopping entirely at a range of 30 meters.

We represent the uncertainty a scan-point position caused by intrinsic sensor noise as a zero-mean Gaussian with covariance $C_{\text{noise}}^{(p)}(t)$. The sensor noise only affects our estimate for the position of the scan point, not the position of the sensor itself. The variance magnitude depends on the range distance of each point, so the value changes from point-to-point with respect to time t.

For a sensor s on our scanning system scanning a point p, we can model the total estimate of the uncertainties of the positions of s and p at a given time t by combining the terms above. The resulting 3×3 covariance matrices $C_s(t)$ and $C_p(t)$ are given by

$$C_s(t) = C^{(s)}_{\text{pose}}(t) + C^{(s)}_{\text{ts}}(t) \tag{4.5}$$

$$C_p(t) = C^{(p)}_{\text{pose}}(t) + C^{(p)}_{\text{ts}}(t) + C_{\text{noise}}. \tag{4.6}$$

We now have estimates of the means and covariances of the positions of both the originating sensor and the scan-point itself for each scan-point collected by our system during a dataset collection. In the next section, we discuss how these values are used to develop a volumetric model for each individual scan-point, allowing it to be used to label volumes in the scanned environment as either *interior* or *exterior*.

4.1.2 Carve Map Generation

Given position estimates of both the sensor and each scan-point taken at time t, with covariances $C_s(t)$ and $C_p(t)$ respectively, we now model the positions of the sensor and scan-point to be represented by Gaussians $N(\vec{\mu}_s, C_s(t))$ and $N(\vec{\mu}_p, C_p(t))$, respectively. An example of this model is shown in Figure 4.2a.

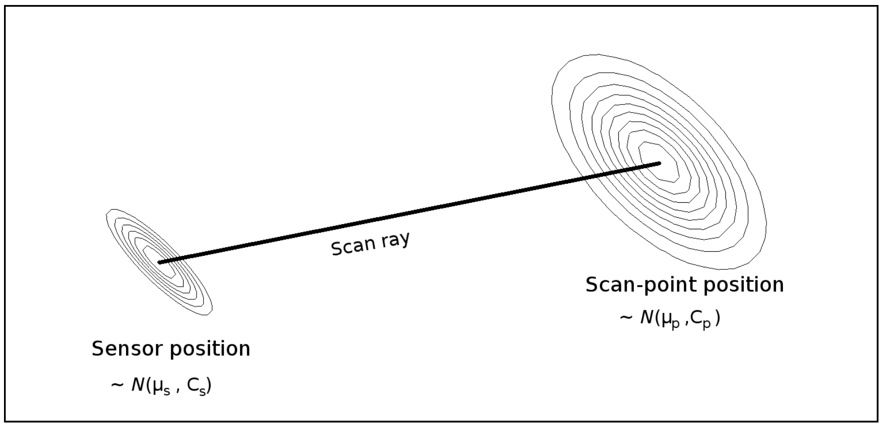
Next, we use these distribution models for each scan point to form a "carve mapping" $p(\vec{x}) : \mathbb{R}^3 \mapsto [0, 1]$, which describes the likelihood of any location $\vec{x} \in \mathbb{R}^3$ of being *interior* or *exterior* based on the position estimates from a scan point. If $p(\vec{x}) \leq 0.5$, then x is more likely in an *exterior* location and if $p(\vec{x}) > 0.5$, then x is more likely to be *interior*.

As each individual scan ray is represented by a probabilistic model, we wish to first determine how each ray would label each position $\vec{x}$ in the environment, and then merge the estimates of that position from all available scan rays to determine the maximum likelihood estimate of $p(\vec{x})$. To determine how best to label $p(\vec{x})$ with a single scan ray, we can infer environment geometry from the orientation of each ray. A time-of-flight scan indicates that there exists a light-of-sight path between the sensor and the scan-point, which indicates that volume near this scan ray should be marked *interior*. The scan-point also indicates that a solid object exists in the environment that occluded the ray at the measured distance. Positions past the scan point should be marked *exterior*. However, positions that are far away from the scan ray should be less influenced by the presence of the ray, since they remain unobserved.

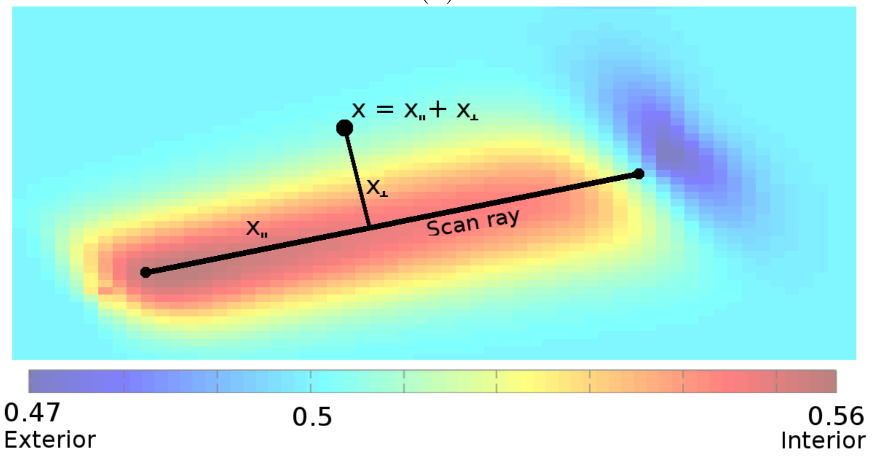
For a single scan ray, we encompass these qualities in our model by defining $p(x)$ as

$$p(\vec{x}) = F_s(x_\parallel)\, f_\perp(x_\perp)\, (1 - F_p(x_\parallel)) + 0.5(1 - f_\perp(x_\perp)) \tag{4.7}$$

where $F_s(.)$, $F_p(.)$, and $f_\perp(.)$ represent one-dimensional marginal distributions derived from the scan ray model, as described below. We split a location $x = x_\parallel + x_\perp$, as shown Figure 4.2b, where $x_\parallel$ is the distance along the length of the scan ray, and $x_\perp$ is the distance orthogonal to the scan ray. This decomposition allows us to weigh the influence of the scan ray on our estimate $p(x)$ so represent a fall-off as $x_\perp$ increases.



(a)



(b)

Figure 4.2: Example carve mapping: (a) the spatial distribution of sensor location and scan-point positions; (b) the computed carve mapping, indicating the areas estimated to be interior and exterior based on this scan.

$F_s(x_{\|})$ is the marginal one-dimensional (1D) Cumulative Distribution Function (CDF) of the sensor position's distribution along the length of the scan ray, derived from the Gaussian distribution of the sensor position. Recall, the marginal distribution of a multivariate Gaussian is also Gaussian. Similarly, $F_p(x_{\|})$ is the marginal CDF of the scan-point position's distribution along the length of the ray.

Lastly, $f_{\perp}(x_{\perp})$ is the 1D Probability Mass Function (PMF) of the lateral position of the scan ray. This function is derived as the marginal distribution along the direction $x_{\perp}$ of the Gaussian defined by $N(x_{\|}, C^*)$, where C^* is defined as the weighted average between the covariances of the sensor and scan-point positions: $C^* = (1 - f)\, C_s(t) \; + \; f\, C_p(t)$ and $f = \|\vec{x}_{\|}\| \;/\; \|\vec{\mu}_p - \vec{\mu}_s\|$, or the fractional distance of $x_{\|}$ along the scan ray.

The combination of these values in Equation 4.7 yields the mapping shown in Figure 4.2b. Values in blue are less than 0.5, indicating a likelihood of a location being *exterior*. As shown, these values occur just past the scan position. Values in red are greater than 0.5, indicating a likelihood of a location being *interior*. These values occur around the sensor position and along the scan ray. As the query location x moves away from the scan ray and $x_{\perp}$ increases, then $p(\vec{x})$ approaches 0.5, indicating unknown state. In other words, locations close to the scanned ray are likely to be *interior*, locations past the scan point are likely to be *exterior*, and locations far away from the scanned ray are unknown.

In addition to the value $p(\vec{x})$, which serves as an estimate for the probability position x is *interior*, we also compute a weighting on this estimate, $w(\vec{x}) = f_\perp(x_\perp)$. This weighting approaches zero as $\vec{x}$ moves farther away from the scan ray, ensuring that a scan ray is only used to estimate the *interior* probability for locations close to it. We use this weighting when consolidating estimates of the value $p(\vec{x})$ from many different scans, as we discuss in Section 4.3.

The above mapping allows us to use each scan ray to "carve" the volume of the model by assigning a probability to all scanned areas indicating the likelihood of areas being interior or exterior. In Section 4.3.1, we discuss how the estimates from all input scans are used to generate a model of the entire environment.

4.2 Voxel Carving Surface Reconstruction

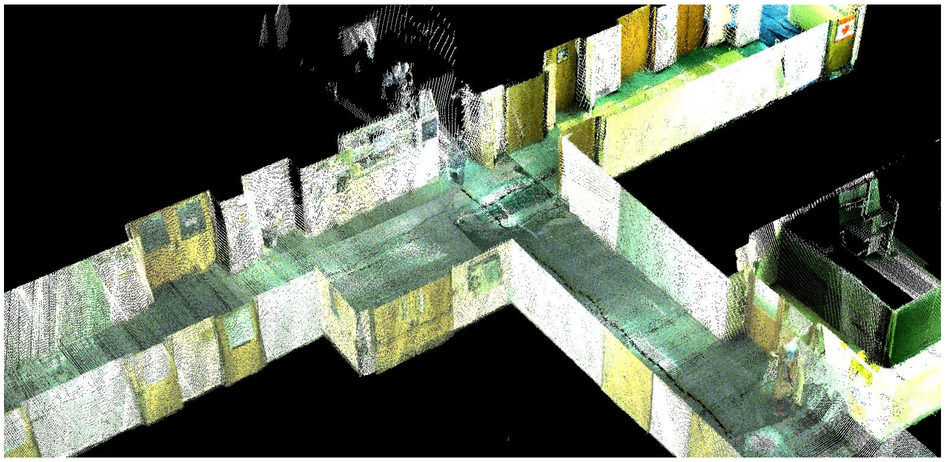
This section describes our voxel carving method for surface reconstruction of building interiors. First, a point cloud of a scanned environment is generated, as shown in Figure 4.3a. The point cloud is used to determine the locations in the volume that are *interior* and *exterior* via a voxel carving scheme. We introduce a novel data structure that allows the carving to be computed in a memory-efficient and scalable manner. Second, once interior and exterior voxels are labeled, the surface defined between these two labelings is segmented into planar regions, as shown in Figure 4.3b. Each region is meshed with triangles that are proportional to its size, as shown in Figure 4.3c. The result accurately and efficiently depicts the geometry of the building.

4.2.1 Voxel Labeling

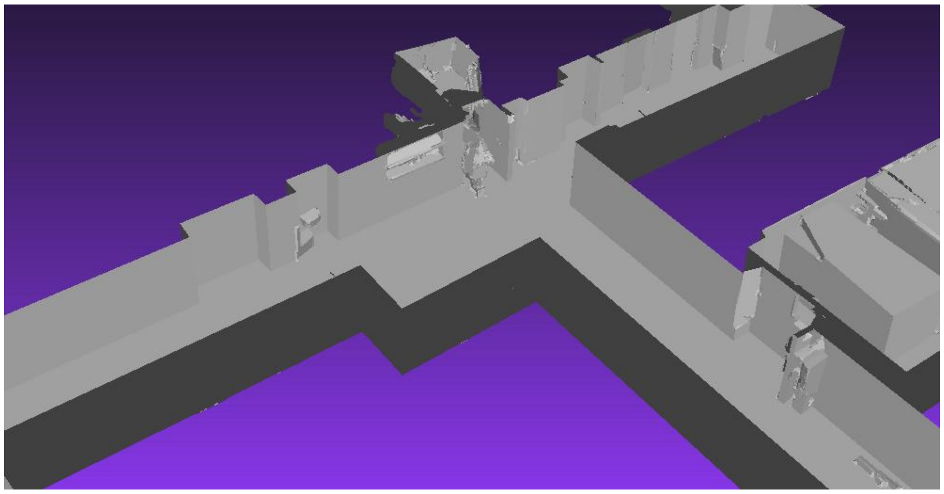
Our proposed interior/exterior volume classification is performed on a voxel grid [93]. Given an input resolution size r, each voxel is a cube whose sides are length r. Initially, all voxels are assumed to be *exterior*, referring to any space outside of the scanned volume or space that is represented by solid objects. The process of *carving* refers to relabeling a voxel from exterior to interior, which occurs when a voxel is found to intersect the line segment from the scanner to a corresponding scan point. If a laser passes through a voxel, that voxel is considered interior space.

For each laser scanner, there exists a track in space that represents the scanner's movement during data collection. This track is represented by a sequence of scanner positions $S = (\vec{s}_1, \vec{s}_2, \vec{s}_3, ..., \vec{s}_N)$, where N is the number of locations sampled during data collection for the given sensor. These track samples are shown as purple circles in Figure 4.4a.

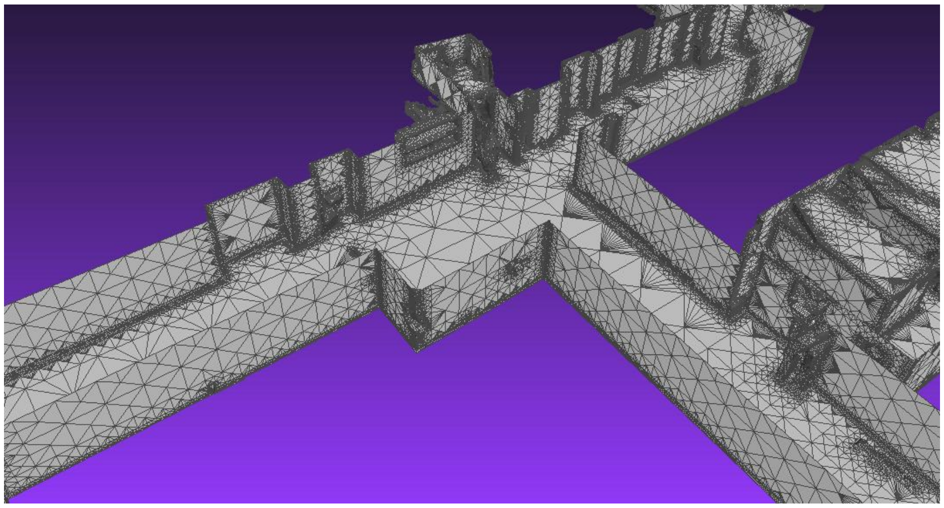
At the i^{th} timestep, each scanner sweeps an arc defined by the set of points $P_i = \{\vec{p}_{i,1}, \vec{p}_{i,2}, ..., \vec{p}_{i,j}, ..., \vec{p}_{i,M}\}$, where M is the number of samples along the arc. Each scan-line is shown in solid red in Figure 4.4a. A scan-line for a given point goes from the sensor position s_i to the scan point position $p_{i,j}$. These scan-lines can be interpolated in two dimensions, indexed by i and j. The first interpolates the laser scans temporally, while the



(a)



(b)



(c)

Figure 4.3: Example processing of office building interior. Ceiling has been removed for visualization: (a) input point cloud; (b) output surface; (c) triangulation of surface.

second interpolates the scans spatially along the scan arc. These interpolations are shown as dashed lines in Figure 4.4a. By performing bilinear interpolation, a continuous surface of scans can be estimated from each scan point $\vec{p}_{i,j}$, shown as the interior of the quadrilateral $(\vec{p}_{i,j},\ \vec{p}_{i,j+1},\ \vec{p}_{i+1,j+1},\ \vec{p}_{i+1,j})$. To efficiently determine which voxels are intersected by this interpolation, the carving operations are performed by ray-tracing between the interpolated scanner position $\vec{s}$ and the interpolated scan point position $\vec{f}$. Each pair $(\vec{s}, \vec{f})$ denotes a line segment to carve. An example set of these segments is shown in green in Figure 4.4b. By spacing these segments no more than distance r apart, each voxel within the interpolated volume is assured to be carved. We perform ray-tracing on each segment and relabel every intersected voxel as *interior*. This step produces a set of voxels as shown in Figure 4.4c.

The method described above is the equivalent of a wedge/cube intersection by interpolating the wedge volume by a series of line segments. Such a computation is efficient since it allows arbitrary orientations of the original scan positions. As we discuss further in Section 4.3, this technique can be improved further by performing a series of triangle/cube intersections, thus reducing the total number of intersection tests when the successive scan points are spaced far apart, requiring significant interpolation.

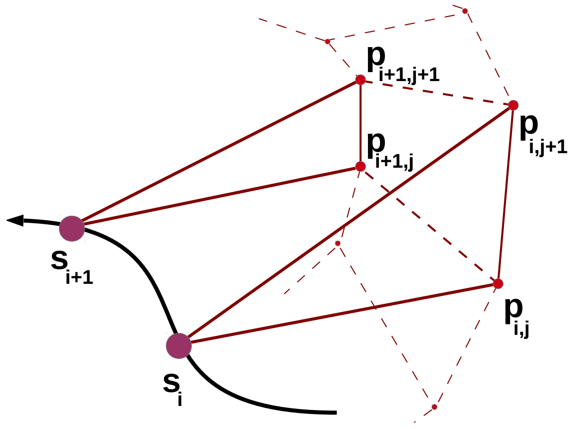
4.2.2 Voxel Data Structure

In most common voxel representations, each voxel in 3D space is explicitly stored in an array in memory. Even though this approach is straightforward and easy to use, its memory use is proportional to the volume represented. For sizable models, this memory footprint rapidly becomes intractable, necessitating splitting models into smaller chunks and processing each separately [5]. This step adds redundant computation and storage overhead. Adaptive approaches such as octrees reduce memory consumption by only representing the subset of relevant volume, but they still explicitly represent volume, an approach that rapidly fills memory [9, 28].

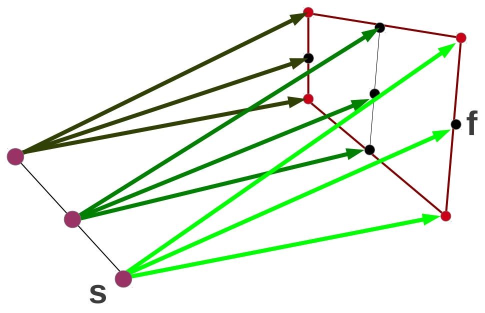
Rather than storing all relevant voxels in memory, we propose a data structure that implicitly represents the interior and exterior voxels by only explicitly storing the boundary voxels. A boundary voxel is defined to be one that is labeled as exterior, but has at least one face incident to a voxel labeled interior. The number of boundary voxels is proportional to the surface area of a model, so storing the boundary only requires $O(n^2)$ memory, whereas the full volume would require $O(n^3)$ memory to store, where n is the diameter of a model.

The data structure used during carving is a map between boundary voxel locations $v \in \mathbb{Z}^3$ and six boolean flags $(f_1, f_2, ..., f_6) \in \{\texttt{false}, \texttt{true}\}^6$, with the following invariants. Each of these flags represents one of the six faces of the referenced voxel. Marking $f_i = \texttt{true}$ indicates that the neighboring voxel of v that shares face f_i must be interior. If $f_i = \texttt{false}$, then this neighboring voxel is exterior, which may mean it is also a boundary voxel.

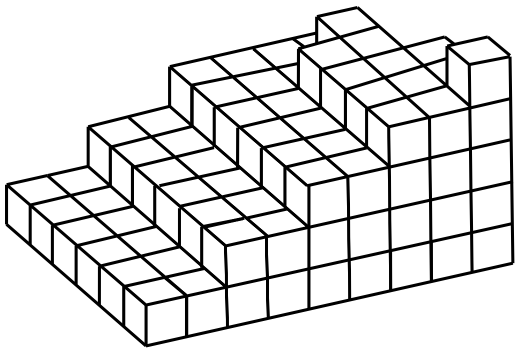
Figure 4.5 demonstrates in 2D how a voxel representation of the full model can be built from a starting configuration using ray-tracing as a primitive operation, while still respecting the above invariants. The starting configuration for the 2D map is shown in Figure 4.5a, with a single interior voxel represented using four boundary voxels. This interior voxel is initialized to be at the scanner's start position, which is known to be interior. Dark green lines indicate faces marked as `true`. During the carving process, if a voxel v is designated to be carved then any of its faces that are flagged as `false` must be incident to exterior voxels,



(a)

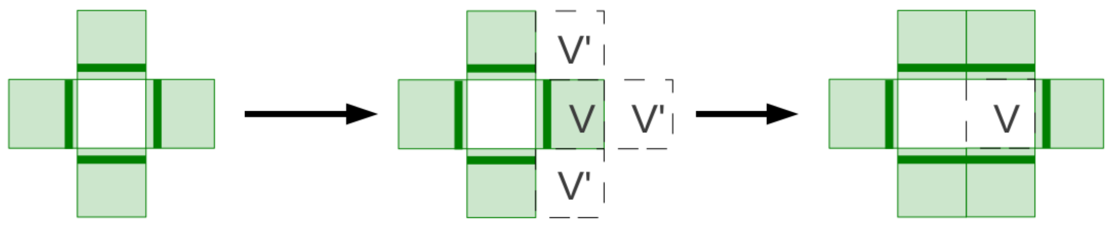


(b)



(c)

Figure 4.4: (a) The input point cloud is used in conjunction with the track of each scanner to define interior space to carve; (b) carving is performed using ray-tracing from the scanner location to an interpolation of the input points; (c) this ray-tracing produces is a set of voxels labeled *interior*.



(a) (b) (c)

Figure 4.5: A 2D example of carving a voxel. Stored boundary voxels are shown in green. White voxels are not explicitly stored. (a) The initial map configuration; (b) voxel v is carved by removing v from the map and adding additional boundary voxels v' to the map; (c) v is represented as interior volume.

as shown in Figure 4.5b. Each of these neighboring exterior voxels, v', is added to the map, as they are now boundary voxels, and the face of v' that is incident to v is flagged as `true`. Lastly, v is removed from the map, which now represents that v is part of the interior volume, as seen in Figure 4.5c. Any carving attempt on a voxel that is not in the map can be ignored, since all carving initiates from within the interior volume. By using only this operation, the map invariants are preserved and always consistently define an interior volume.

This data structure uses less memory than the corresponding adaptive octree structure discussed later in Section 4.3, since it only explicitly stores elements at the boundary between interior and exterior volumes. There are a number of downsides, however, such as the inability to query if a given point in space is interior or exterior, and the uniform size of the voxels. As we discuss further in Section 4.3, an octree structure may have more memory overhead, but it allows for further computation than what is accomplished here.

4.2.3 Preserving Fine Detail

The above carving process may fail to preserve features for point clouds with low noise, but high detail. Objects whose feature length is on the order of one voxel size may be carved away. This issue can be a serious problem if two rooms are separated by a thin wall. Scanning both of these rooms may carve away this wall, yielding a final model that shows only one, double-sized room. We store a second voxel set that specifies the voxels that are intersected by the input point cloud to preserve these features. While the original point cloud is often much too large to be stored in memory at once, this discretization is much smaller and is of the same order of storage space as the map of boundary voxels.

During the carving of each line segment, if ray-tracing encounters a voxel that is marked to contain input points, then the carving of that segment is truncated just before this voxel. No features that are represented in the point cloud are ever carved away. Since ray-tracing already occurs on a voxelized grid, this occlusion check does not add any appreciable complexity to the computation.

This process does not use any of the probabilistic models discussed in Section 4.1, but rather assumes that a voxel intersected by a scan point is *exterior*, any voxels intersected

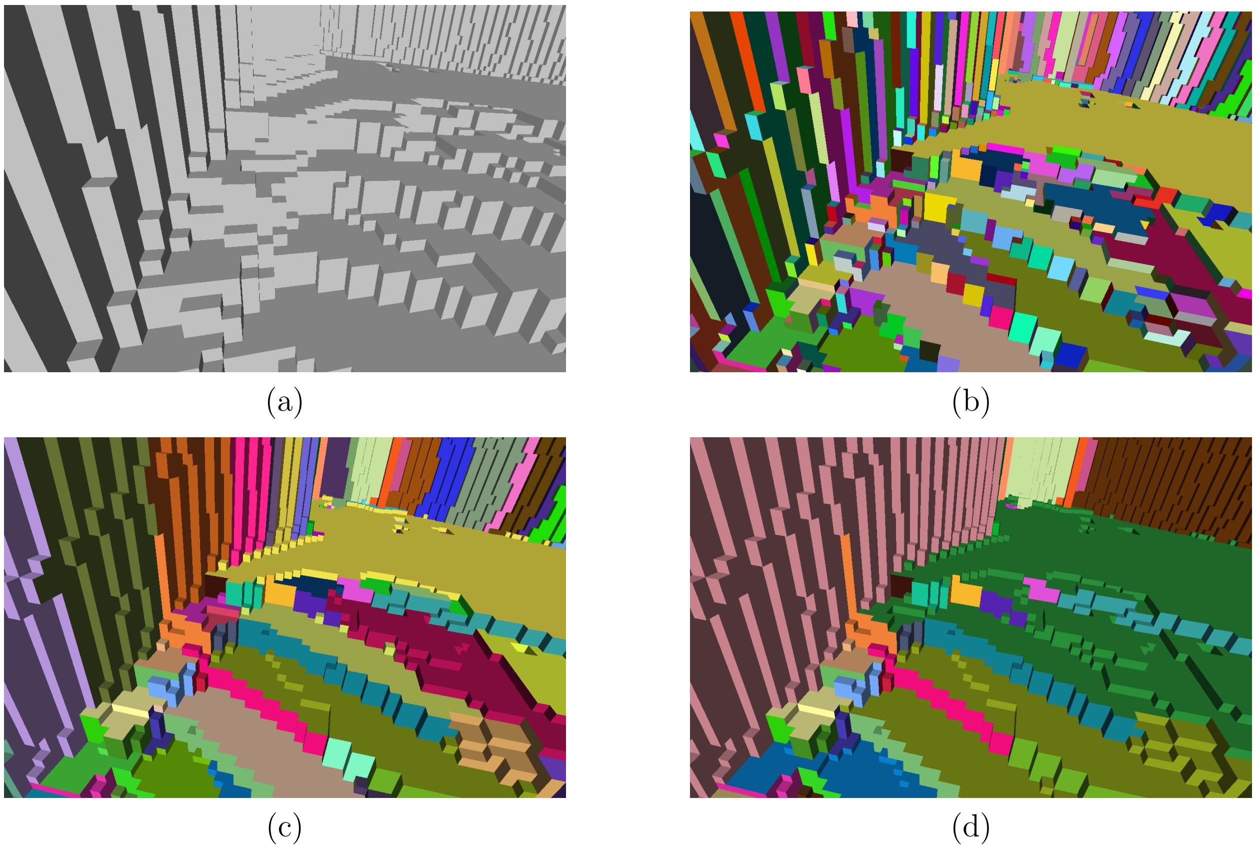


Figure 4.6: (a) Carved voxels at the top of a flight of stairs; (b) regions colored based on voxel face flood-fill; (c) region growing by finding best-fit planes to voxel faces; (d) regions relaxed to merge planes that are nearly parallel

by a scan line are *interior*, and all other voxels are unaffected. As such, the presence of noisy points may disrupt the quality of the output surface. Such concerns are mitigated in the procedure we describe in Section 4.3, where we introduce a carving mechanism that incorporates the probabilistic models discussed in Section 4.1.

4.2.4 Planar Region Fitting

Our procedure for surface reconstruction of voxels can be broken into two parts. First, estimates of planar regions are found around the boundary faces of these voxels. These regions are formed from connected sets of voxel faces, all of which are positioned on best-fit planes. Second, each region is triangulated, forming a mesh. This triangulation lies along the best-fit plane for each region, with elements whose sizes are proportional to the size of the region.

The first task is to determine the connectivity along the carved voxel faces. An example of such a carving for a flight of stairs is shown in Figure 4.6a. Since these faces are squares that form a watertight surface and lie on an axis-aligned grid, each face has exactly four neighbors. If a voxel face and its neighbor are both oriented in the same direction, e.g.

both have normal vectors in the Z+ direction, then one can immediately perform a flood-fill operation to group these faces into planar regions. The faces belonging to each region lie exactly on a plane. The result of this flood-fill operation is shown in Figure 4.6b.

Since the voxels are a discretized representation of the volume, any flat surface of the environment that is not axis-aligned is represented by a zig-zag pattern of voxels. By fitting planes that only approximate the voxel faces, the output model can contain surfaces that are not axis-aligned. The approximating planes are found by performing Principle Component Analysis (PCA) on connected subsets of voxel faces [86]. For any connected set of voxel faces V, PCA is performed on the four corners of all the faces to estimate a best-fit plane. If V is well-modeled by this plane, then the elements of V are grouped together as one planar region. V is considered well-modeled if the maximum distance of V from the plane is at most the grid size, which is typically chosen to be 5 cm. This threshold guarantees that any voxels intersected by the modeling plane are incident to the faces in V.

Starting with the regions found by the flood-fill operation above, adjacent regions of voxel faces are progressively merged by attempting to model their union with a single best-fit plane. If this plane meets the threshold described above, then the two adjacent regions are replaced by one region representing their union. This step is referred to as *region growing*. Even though this stage reduces the total number of regions, it typically produces an overfitting of too many regions. An example of this stage is shown in Figure 4.6c.

We further relax these region definitions to yield a more aesthetically pleasing output. If two adjacent regions are fit by planes whose normal vectors are within 15°, then they are replaced by a single region defined by their union. The result of this processing yields plane definitions that closely resemble an intuitive labeling of the floors, walls, and ceilings. This final region labeling is shown in Figure 4.6d.

4.2.5 Triangulation of Regions

Once the set of voxel faces has been partitioned into planar regions, it is necessary to triangulate these regions. As the output mesh represents the planar regions found in the previous section, an optimal approach would adapt the sizes of triangles based on the sizes of these planar regions.

Taking advantage of the existing voxel grid ensures that each region is represented by good-quality triangles.[1] This grid allows for regions to be triangulated with a 2D variant of Isosurface Stuffing techniques, which provide strict bounds on triangle angles [24]. An example of a region of voxel faces is shown in Figure 4.7a. As this region is best fit by a plane that is not axis-aligned, the region is composed of voxel faces in a zig-zag pattern. The voxel faces that are most aligned with the normal vector of the region's plane, shown in red, are considered the dominant faces of the region. These dominant faces are projected along their corresponding axis to generate an axis-aligned 2D projection of the region. This projection is shown with black dashed lines in Figure 4.7a. The triangulation is found by populating a quadtree that is aligned to the projected grid with the faces of this region. An example of this quadtree is shown in Figure 4.7b. The tree is triangulated by placing vertices at the

[1]Quality of triangles is based on bounding the minimum or maximum angles. Having overly acute or obtuse angles can cause artifacts in texture-mapping or normal analysis.

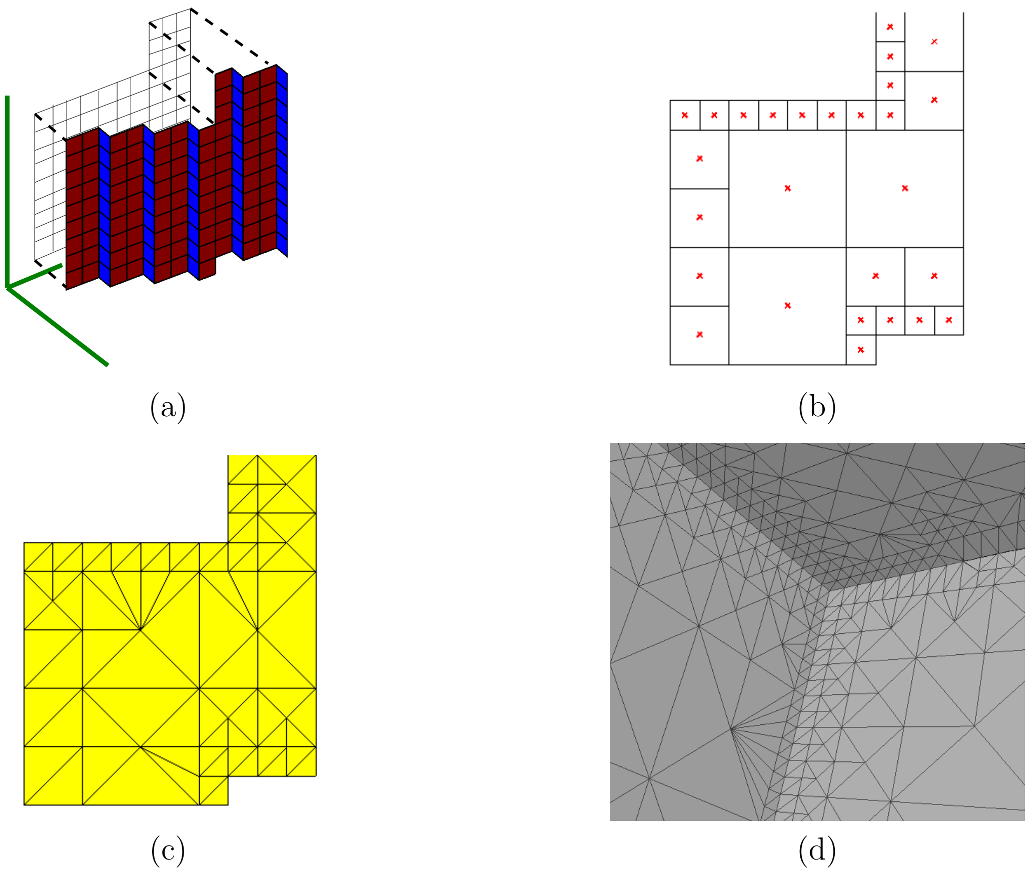


Figure 4.7: (a) The dominant faces of a planar region (red) are projected to the dominant axis-aligned plane; (b) projected faces represented in a quadtree structure to reduce the number of elements; (c) this quadtree can be triangulated efficiently with high-quality triangles; (d) an example output triangulation of three regions in the corner of a room.

center and corners of the leaf nodes, as shown in Figure 4.7c. This step generates larger triangles for larger leaf nodes, while still controlling the quality of the output triangles. This triangulation is projected back onto the plane defined by the region, to produce a triangulated representation of this region in 3D space.

As the connectivity among voxel faces is well-defined, the connectivity of the output triangulation is also well-defined. To ensure that the borders between planar regions are represented sharply, the vertices shared by multiple regions are snapped onto the intersection of those regions. This fits the intersection between two regions to a line, and the intersection of three or more regions to a point in space. This step yields a watertight mesh across regions, as can be seen in the intersection of three regions at the corner of a room in Figure 4.7d. To limit self-intersections in the final surface, the vertices shared by multiple regions are allowed to be displaced up to a distance threshold from their original position in the voxel grid. This threshold is increased as the angle between the regions in question approaches 90°. The corners between walls and ceilings remain sharp, whereas the transitions between regions close to parallel is smooth. If such a threshold does not exist for the boundaries between near-parallel regions, their shared vertices could produce undesirable artifacts.

4.3 Probabilistic Octree Surface Reconstruction

In Section 4.1, we discuss how a probability model can be computed for each scan point, allowing that point to perform a volumetric labeling of nearby space. In Section 4.2, we discuss how a voxel carving method can be used to partition the scan environment into *interior* and *exterior* domains by performing a set of cube/line-segment intersection tests. In this section, we combine these two ideas to improve upon the voxel carving method described in the previous section.

Rather than simply labeling any voxel that is intersected by a scan ray as *interior*, we consolidate all carve mapping estimates of nearby scan rays, ensuring that our estimate of a particular location's *interior* probability is as consistent with the entire scan dataset as possible. In Section 4.3.1, we discuss how we consolidate and interpolate each individual scan's probability estimates to develop a unified estimate for a given location across all scans. In Section 4.3.2, we describe the octree data structure that is used to house all consolidated estimates for the entire scan environment. Then, in Section 4.3.3, we detail how the volumetric octree structure is converted to a mesh representation of the boundary of the *interior* volume. We describe two different meshing techniques that can be applied to the octree structure and compare the qualities of each.

4.3.1 Wedge Generation

In Section 4.2.1, we discussed how successive scan rays are interpolated in a bilinear fashion: both between adjacent scan points within a single frame and between the same point index in adjacent frames. These interpolations represent a "wedge" volume between four adjacent scan rays, as shown in Figure 4.4. The goal of this interpolation is to determine which voxel elements intersect a particular scan. In this improved method, we do not simply label any intersected cube as *interior*, but rather consolidate estimates of the probability of an element being *interior* or *exterior* from all nearby scans. Thus each "wedge" is represented to extend to a volume beyond the original four scan rays. We extend each ray of the wedge by an additional three standard deviations in length, taken from the Gaussian model of the scan point position distribution. In this way, we ensure that all volume relevant to the scan rays of this wedge is covered. We precompute all wedges of the dataset so that they can each contribute to the volumetric labeling of the entire scan environment.

The scan wedges are originally sorted temporally as the operator moves around the environment. We first organize the input scans into spatial chunks to process the scanned volume in a parallel fashion. The environment volume is tessellated into large cubes, and each cube contains a list of all the scan wedges that intersect it. When we populate the octree structure, each chunk can be processed independently and concurrently via a thread pool. The scan wedges are inserted into each chunk they intersect, refining the nodes under the chunk node to a fine resolution. We typically use chunks of size 2 meters on a side, aligned to our octree structure as non-leaf nodes of the environment. This size allows each individual chunk to easily fit into memory while still being large enough to efficiently subdivide the

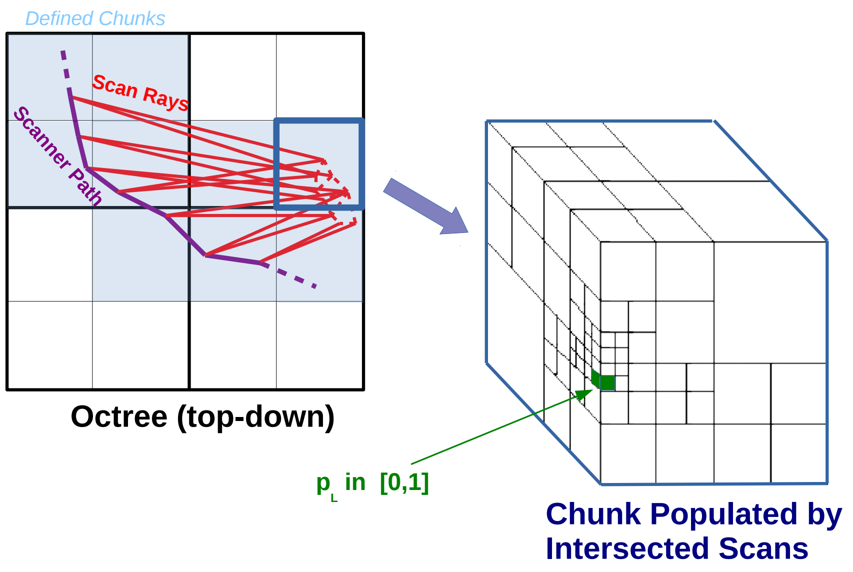


Figure 4.8: As each system sensor traverses the environment, scan rays from that sensor are collected and sorted spatially into chunks. All scans from each chunk are inserted into the octree structure in a parallel fashion. The leaf nodes of the octree represent the probability of that volume element being *interior* or *exterior*.

entire environment. The choice of chunk size is independent of sensor type or scan density, but rather is based on the expected sizes of building elements.

4.3.2 Octree Data Structure

As discussed above, the original scans of a dataset generate probability estimates for each spatial location in the scan environment, determining whether a unit of volume is labeled as *interior* or *exterior*. To perform this labeling, we require a data-structure that supports efficient representation of spatial information, including fast lookups for geometric intersection tests. For these purposes, we use an octree data structure to store *interior* probability estimates for each location in space. The octree is represented to a finite node depth. The finest resolution of the octree is typically set to be on the order of 1 centimeter. This resolution gives high level of detail even for small pieces of furniture in the scanned environment. Figure 4.8 shows a diagram of how scan wedges are sorted spatially into chunks and inserted into the octree structure.

Octrees have been utilized successfully in the past to represent volumetric scan data [9,97, 98]. They provide an improvement over the voxel grid structure mentioned in the Section 4.2, as they permit adaptive resolutions to be used in different locations of the model and provide fast intersection tests against scan geometries. We store a probabilistic estimate at every leaf node that indicates the likelihood of that volume element being *interior* or *exterior*. The following process inserts all the scans into an octree in a chunk-wise fashion, so that it can be performed in a parallel- and memory-efficient manner.

We use the octree structure to identify all the locations in the environment volume that are intersected by at least one scan wedge. For each leaf node L that contains at least one wedge, we merge all the intersecting scan wedges to obtain a fused probabilistic estimate p_L. If any wedge intersects L, then all four scan rays from that wedge are used to contribute to the value p_L. Let S be the list of intersecting scan rays. The carve mapping estimates from each individual scan $i \in S$ are combined to generate the fused, maximum-likelihood probability estimate for this leaf node

$$p_L = \frac{\sum\limits_{i \in S} p_i(\vec{x}_L)\, w_i(\vec{x}_L)}{\sum\limits_{i \in S} w_i(\vec{x}_L)}, \tag{4.8}$$

where $\vec{x}_L$ represents the center position of cube L, $p_i(.)$ represents the carve-mapping function from Equation 4.7, and $w_i(.)$ is the weighting function of each individual scan ray such that $w(\vec{x}) = f_\perp(x_\perp))$. Once all scan rays are inserted into the octree, each leaf node contains its respective probability estimate p_L, the total weighting of this estimate, the variance of this estimate, and the number of scans that intersected L. All these statistics are used later in the pipeline for analyzing the properties of L. As an example, if p_L is 0.5 or less, then the node is considered *exterior*. Nodes that are never intersected by any scans are assumed to be *exterior* and are assigned a value of $p_L = 0.5$. If the value of p_L is strictly greater than 0.5, then the node is considered *interior*. The faces between *interior* nodes and *exterior* nodes are considered boundary faces of the octree, and are useful for determining the position of generated meshes. As we discuss in Section 4.3.3, the position of the mesh between two such nodes is determined using the means and variances of our estimates of p_L for each node.

Once each chunk is completely carved, we simplify its tree structure based on the dynamic range of the contained values. A node is a candidate for simplification only if all of its sub-nodes are leaf nodes and report the same label, i.e. all are *interior* or all are *exterior*. Additionally, all sub-nodes must have either been observed at least once or none of the nodes observed. This condition prevents oversimplification in parts of the volume that may have been only partially scanned. If these conditions are met, then the sub-nodes are deleted and their data are averaged and stored into their parent, which is now the new leaf node at that location. This process is performed recursively to produce a minimally populated tree for the building area being modeled. The size of each chunk can be adjusted to improve processing. While large chunks require more memory to be used at a time, they result in fewer chunks to be processed overall.

The final tree is only generated to full depth in the locations that require it, which are boundaries between *interior* and *exterior* spaces. As we discuss in Section 5.6, areas of the environment that are segmented as objects in each room will be re-carved to an even finer resolution, since these locations are likely to contain high detail.

4.3.3 Computing Boundary Surfaces

Once we obtain a volumetric representation of the scanned environment in an octree, we wish to extract the surface of the environment from this structure. In this section, we discuss two methods for representing the surface boundary. The first method performs dense meshing to preserve all detail stored in the octree structure. This technique is useful for representing fine detail observed in the scan environment, such as the geometry of furniture and objects. The second method represents the boundary surface of the volume via a set of piece-wise planar regions. This method is useful for reducing noise on large surfaces observed in buildings, such as floors, walls, and ceilings. As we discuss in Section 5.6, both methods are used in one output model once furniture and other objects in the environment are segmented.

Dense octree meshing As discussed earlier, a boundary face is the surface between two abutting leaf nodes of the octree, where one leaf node has $p_{L1} \leq 0.5$ and the other has $p_{L2} > 0.5$. To determine the offset position of the surface between these nodes, we preserve sub-node accuracy by positioning the boundary at the isosurface $P = 0.5$ generated by linearly interpolating the probability estimates of the two nodes, as shown in Figure 4.9.

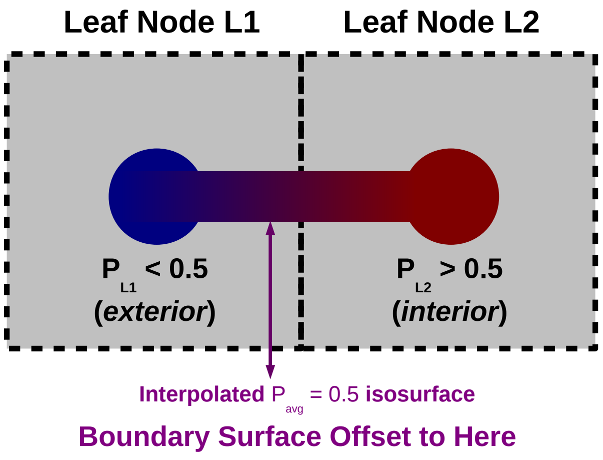


Figure 4.9: We place the boundary surface of the octree between two leaf nodes with opposite labeling. Rather than placing the surface half-way between, we use the value of the probability estimate to determine optimal surface offset, resulting in sub-node surface accuracy.

Once we have identified the position offset of the surface at all boundary leaf nodes, we can generate a dense mesh of this boundary. We use a variant of Dual Contouring [18]. It works well with adaptively-sized nodes in an octree and well-represents both curved and sharp features in the output geometry. Since our data labels are divided into node centers of the tree, rather than node corners, we perform dual contouring by mapping each boundary face of the octree to a vertex in the mesh and each corner of the octree into a face in the meshed output. Examples of this meshing technique is shown later in this section in Figure 4.13.

Planar region meshing To mesh building geometry in a planar fashion, we first partition the boundary faces of the octree into planar regions, in the same fashion as the method described in Section 4.2.4.

Each planar region represents a set of node faces along with fitting plane geometry. The fitting plane of each region is formed by running Principal Component Analysis (PCA) on the centers of the boundary faces. To maximize the accuracy of this surface position, we perform PCA on the isosurface positions found above, rather than the grid-aligned node faces. Once the set of node faces is partitioned into planar regions, we generate a watertight mesh by finding the intersection points between each pair of neighboring planes and insert vertices for our output mesh.

Naively intersecting the fitting planes of each region to position the output vertices may generate artifacts or self intersections at locations where two nearly-parallel regions are neighbors. Figure 4.10 shows an example of this phenomenon in 2D. Since the two regions shown are close to parallel, their intersection point shown as the black star is far away from the adjoining endpoints of the regions. Rather, we use a pseudo-intersection point that is closer to the original corner position, shown as the green star in Figure 4.10. We therefore use the following technique to prevent over-constraining the intersection points of degenerate region neighbors.

These intersection points are represented by corners in the octree that intersect faces from more than one planar region. For a given node corner with position $\vec{x}_0$ that intersects a set of planar regions R, we wish to find a final position of the mesh vertex that corresponds to this node corner. By merely taking the intersection of all planes, the vertex position may be under-constrained if only two regions are touching the corner, or if some of the regions are close to being planar with each other. We compute this point with the following process.

Let matrix M be defined so that each row $\vec{n}_i^T$ is the normal vector of the i^{th} intersecting planar region, and vector $\vec{p} = \texttt{diag}(P\,M^T)$, where matrix P is defined so that its i^{th} row is any point on the i^{th} plane. We compute the desired position of the intersection point $\vec{x}$ with the following:

$$\vec{x} = \sum_{i=0}^{3} \left(\delta_i \frac{\vec{p} \cdot \vec{u}_i}{s_i} + (1 - \delta_i)(\vec{x}_0 \cdot \vec{v}_i) \right) \vec{v}_i \tag{4.9}$$

where $\vec{u}_i$ and $\vec{v}_i$ are the i^{th} columns of matrices U and V in the SVD decomposition of $M = U\,S\,V^T$, respectively, and s_i is the i^{th} singular value of this decomposition. The term δ_i is defined as 1 if $s_i \geq \alpha$ and as 0 otherwise, for some threshold α. Note this approach finds the intersection of all given planes associated with this corner, but does not incorporate the constraints from two nearly-parallel planes. Equation 4.9 sums over all excited dimensions of the set of intersecting regions, and either projects the intersection point along a dimension if it is highly constrained or merely takes the point's original coordinate in that dimension $\vec{x}_0$ if it is not well constrained. In the 2D example in Figure 4.10, the normal vectors $\vec{n}_1$ and $\vec{n}_2$ are near enough to parallel to be considered degenerate. The corresponding nullspace is shown in grey. As a result, the pseudo-intersection is the original corner position projected onto this nullspace rather than the true intersection shown in black. We use the threshold parameter of $\alpha = 0.2 * s_{min}$, where s_{min} is the smallest singular value.

Once the locations of vertices shared by two or more planar regions are computed, then the interior area of each region is triangulated using a 2D variant of isosurface-stuffing [24], as described in Section 4.2.5 [93]. This produces an efficient number of triangles for large, planar areas and preserves the sharp corners at the intersection of these regions. Examples

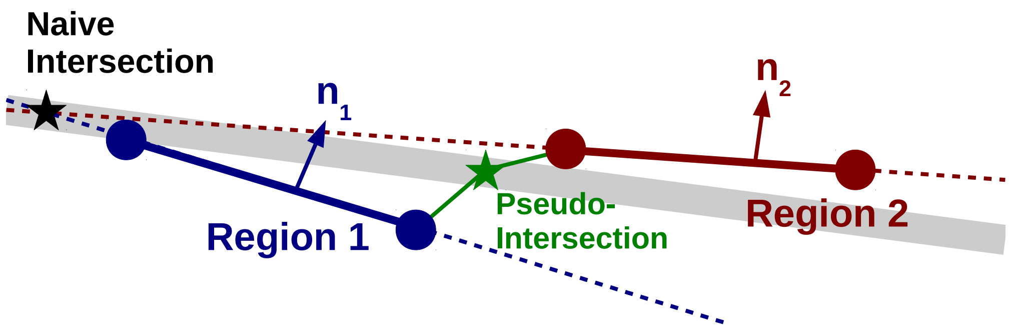


Figure 4.10: An example of plane intersection in 2D. Two adjoining regions are shown, with normals n_1 and n_2. The naive intersection of the two regions is shown as a black star. Rather, we join the two regions with a vertex shown at the green star, which prevents sharp discontinuities in the output.

of this meshing technique are shown later in this chapter in Figure 4.12.

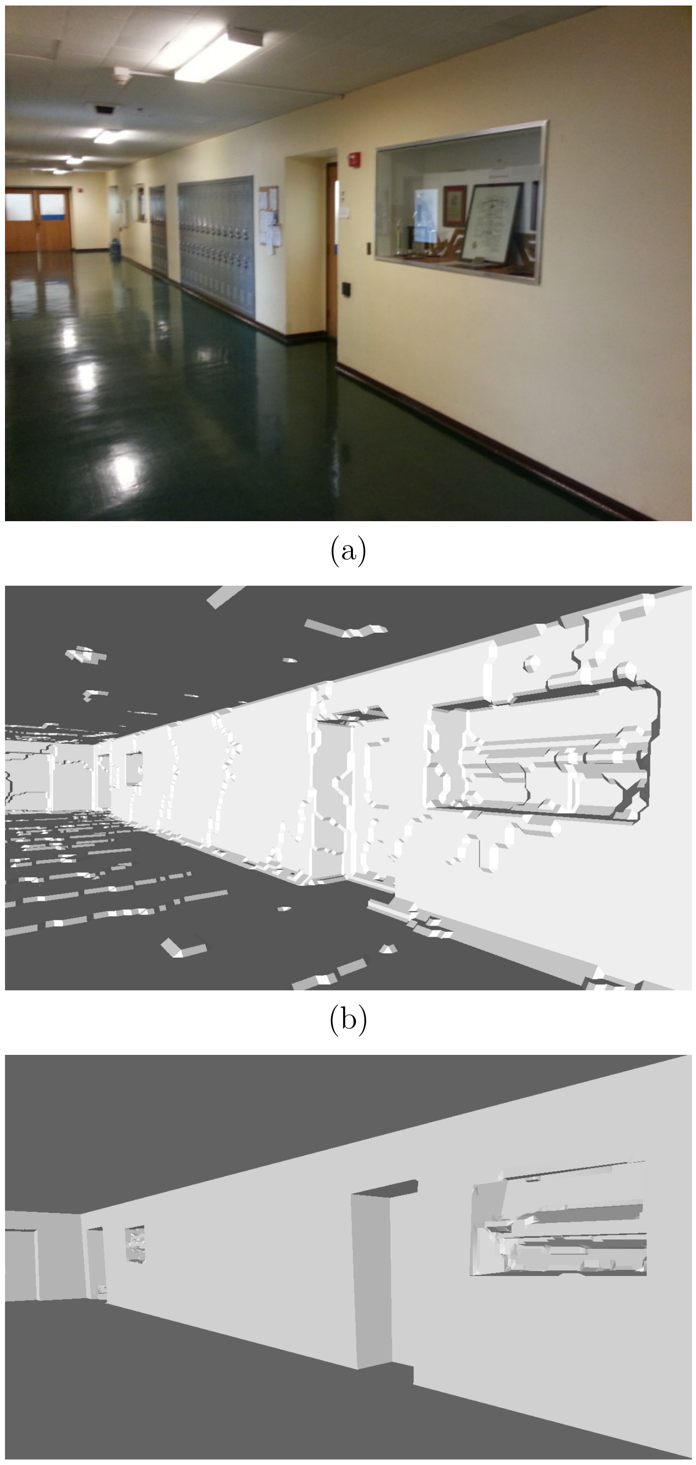
4.4 Results

In this section, we present several comparisons and results of the methods described in this chapter. In Section 4.4.1, we compare and contrast our proposed methods with an existing carving approach and with each other. In Section 4.4.2, we show several results of applying our methods on a variety of datasets, as well as discuss the scalability of our methods. In addition, more results from the method described in Section 4.3 are also shown in Chapter 5.

4.4.1 Comparison of Methods

The results of our surface reconstruction procedure are analyzed quantitatively and qualitatively. For quantitative analysis, the resulting mesh is compared to an existing voxel carving scheme, which uses Marching Cubes to generate a final output [5]. Figure 4.11 shows a qualitative comparison of the two schemes. The corners where the floor meets the walls or where the walls meet the ceiling are preserved in our proposed method, whereas in traditional marching cubes, they are rounded off. Further, since our method explicitly fits planar regions to the mesh, the output mesh contains less surface noise. Additionally, by using adaptive triangulation of these regions, the mesh can be represented with far fewer elements.

Next, we compare the voxel carving method described in Section 4.2 with the probabilistic octree carving method proposed in Section 4.3. Among the modifications are that probabilistic models are used to assess volumetric labeling of the input scans, an adaptive octree data structure is used to store the *interior/exterior* labelings, and a more sophisticated method is used to position the intersecting vertices between planar regions. The effects of



(a)

(b)

(c)

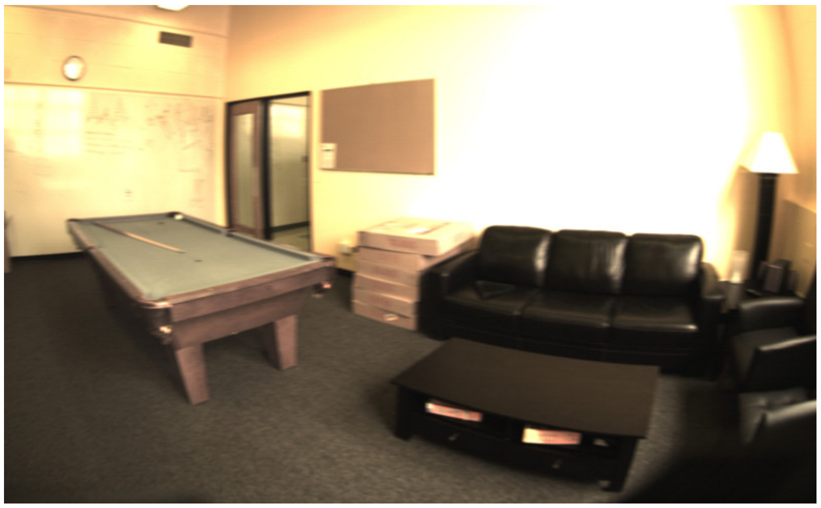
Figure 4.11: A visual comparison of meshing method for a hallway model: (a) photograph of scanned area; (b) an existing voxel carving method [5]; (c) the proposed method from Section 4.2 at 5 cm resolution.

these modifications can be seen in Figures 4.12 and 4.13, which show a small lounge environment that includes a couch, a coffee table, and a pool table. Figure 4.12a shows a reference photograph of this environment and Figure 4.13a shows the raw input scan points used to generate the resulting models.

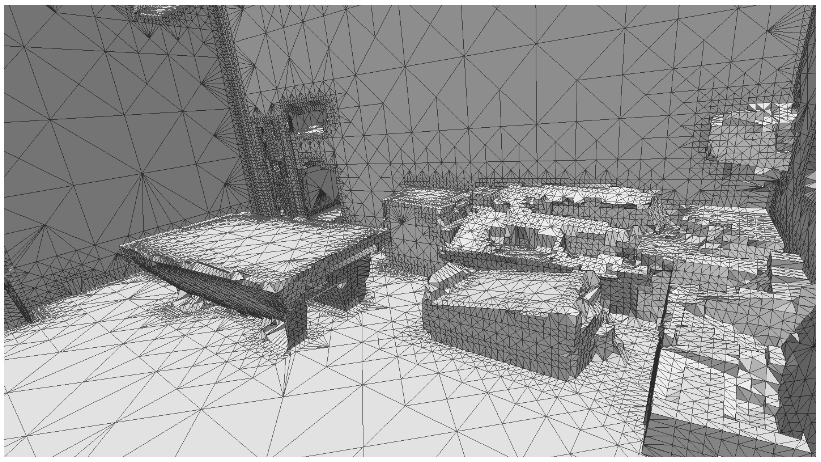
The voxel carving method is shown in Figure 4.12b [93]. Note how all surfaces in the model are represented as piece-wise planar. The borders of these planar regions are often represented as rough or jagged. Figure 4.12c shows the probabilistic octree carving method, using planar region fitting. Again, all surfaces are represented as piece-wise planar regions, as discussed in Section 4.3.3. In the modified case, however, the borders between regions are preserved as sharp and clean. This characteristic is especially noticeable on the couch, where the separation between the seat and back of the couch is much noisier in Figure 4.12b than in Figure 4.12c. Additionally, the top of the coffee table is skewed in Figure 4.12b, whereas in Figure 4.12c it is level.

We can also see the effect of using the probabilistic carving between these two models. Since the method described in Section 4.2, and shown in Figure 4.12b, assumes that any voxels intersecting scan points are exterior, the result is a "bloated" version of all objects, where their surfaces are offset from the scan points by half a voxel size on average. Comparing to Figure 4.12c, which represents the probabilistic octree carving, there are visual differences. Specifically, since the full probability model for each scan point is used to determine the optimum position of the exported surface no bloat occurs, and the modeled output objects are the correct size.

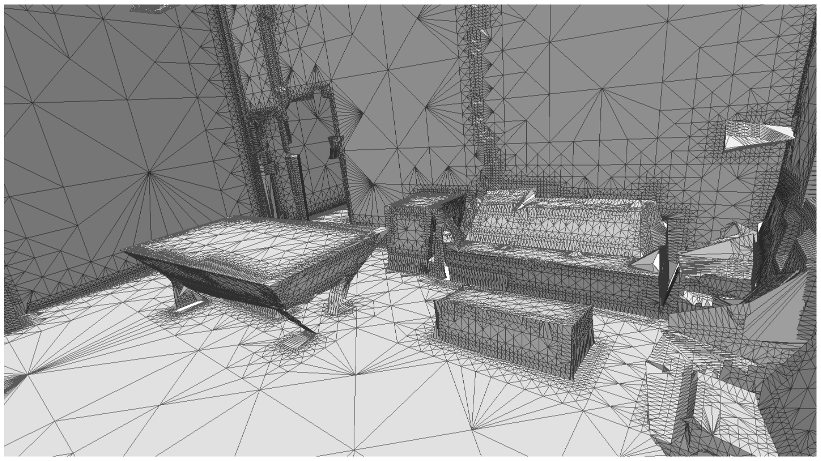
Since, in both Figures 4.12b and 4.12c, all surfaces are represented with planar regions, a great deal of detail is removed during region fitting. We can observe the original detail of the probabilistic octree model in Figure 4.13b. This figure shows a dense surface reconstruction of the same environment, which does not use any planar fitting. An alternate view of this same output geometry is shown in Figure 4.13c, which shows the same model with Gouraud shading. Note that this version of the model preserves more detail, including the individual cushions in the couch and the legs of the pool table. By not performing planar region fitting, however, we also preserve noise on flat surfaces, such as on the walls and floors. In Section 5.6, we discuss methods to keep the best of both worlds: high-fidelity dense surface reconstruction on small objects such as furniture and light fixtures and aggressive plane-fitting methods on the large surfaces of the building such as floors, walls, and ceilings.



(a)

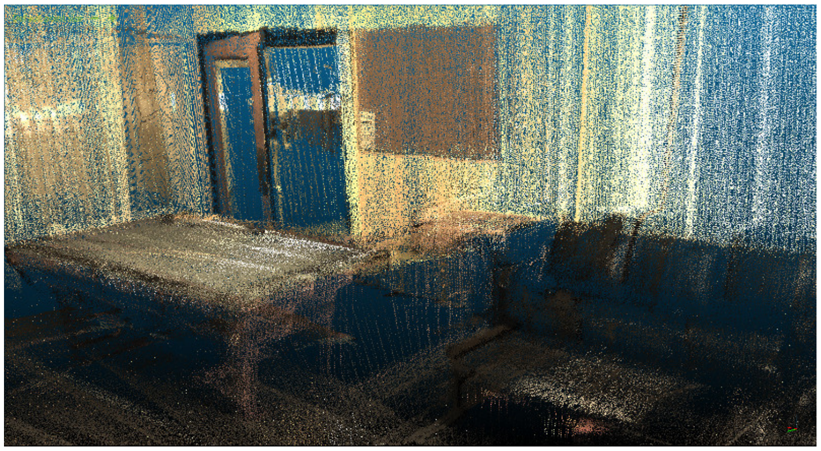


(b)

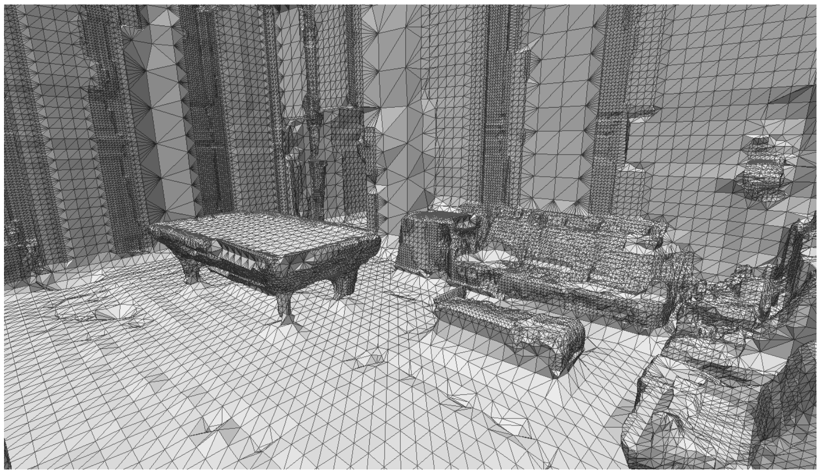


(c)

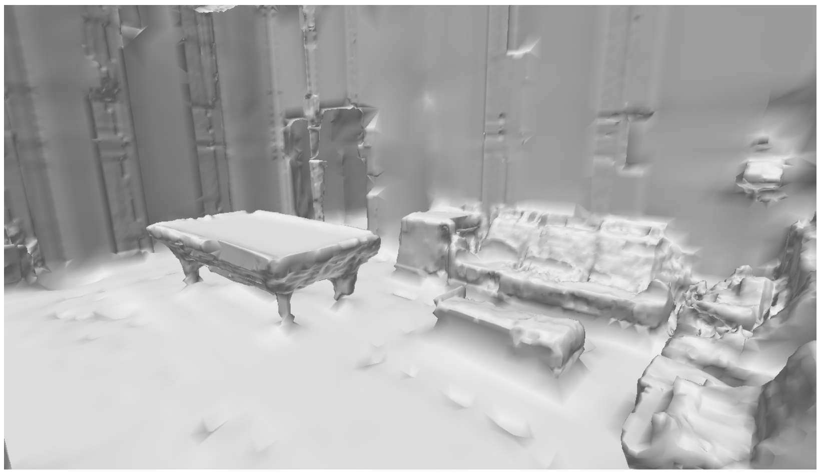
Figure 4.12: Comparisons between the various reconstruction methods discussed in this chapter. Example shown on small lounge area: (a) photograph of scanned environment; (b) voxel carving output as described in Section 4.2 with planar region fitting; (c) probabilistic octree carving output with improved planar region fitting, as described in Section 4.3.



(a)



(b)



(c)

Figure 4.13: A visualization of the dense meshing method described in Section 4.3. Example shown on small lounge area: (a) raw point cloud scans of environment; (b) octree carving output with dense meshing; (c) alternate shading on (b).

4.4.2 Voxel Carving Results

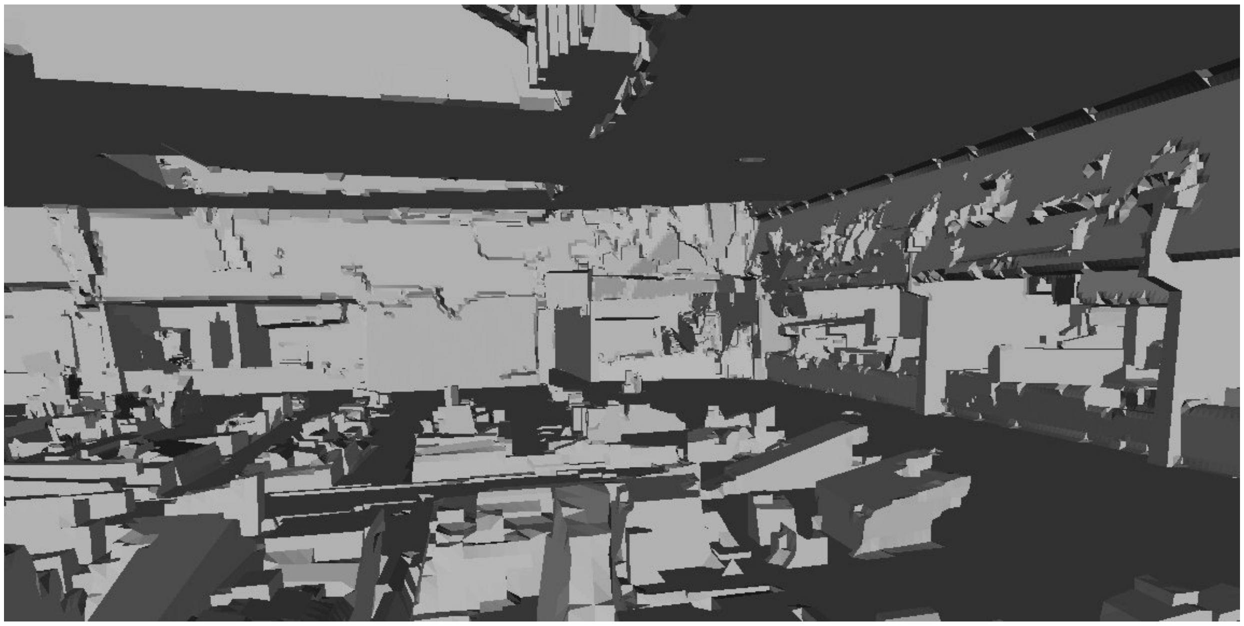
The voxel carving method from Section 4.2 was run on several datasets, which range in size from a single conference room to full floors of buildings such as hotels and shopping malls. The results are shown for sections of these models, along with the corresponding views of the original point clouds. For these models, large flat areas are represented by fewer, larger triangles.

Figure 4.14 shows the reconstruction of a shopping mall's food court. The input point cloud contains significant noise due to the amount of glass surfaces in the model. In the food court, the restaurants are well-modeled, as well as the ceiling and skylights. Figures 4.15 and 4.16 show our largest model, with 220 million input scan points. This model represents the aisles of a retail store, covering an area of 112.2m $\times$ 77.5m using 2.7 million triangles. A smaller dataset is shown in Figure 4.17, representing a 10.5m $\times$ 9.5m conference room with a hexagonal table in the center. This table, along with the podium to the left, is well-represented in the output. The ceiling of the conference room is inset with hanging lights, which can also be seen in the model.

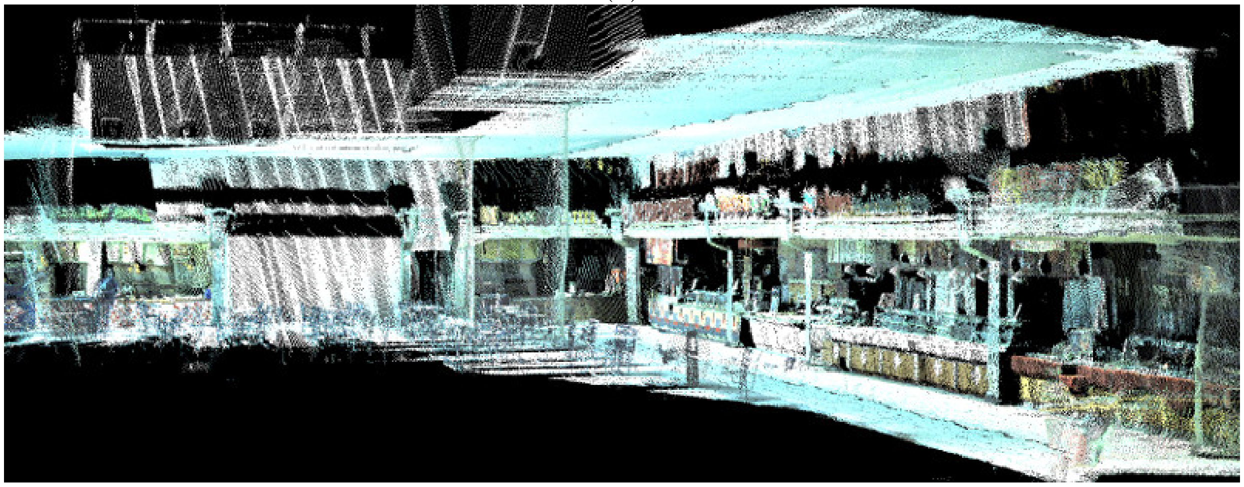
Figure 4.18 shows a model of a construction site. The surfaces of the room are represented by large regions, while the detail of objects is preserved. Run-time analysis was performed on the dataset shown in Figure 4.18. The input to this dataset contains 25 million points. The code was run on a personal laptop with an Intel i7-2620M processor. The voxel carving, at 5 centimeter resolution, took 55 minutes of processing time. The surface reconstruction of these voxels took 1 minute and 2 seconds. Existing voxel carving schemes processed similar models of 15 million points in 16 hours at the same resolution [5]. Computation time was recorded for this same dataset with a resolution of 2 cm. Voxel carving took 12 hours and 10 minutes for this resolution, while surface reconstruction took 9.5 minutes.

Figure 4.20 shows an example of the full extent of a 96.7m $\times$ 75.7m H-shaped hotel hallway. The input point cloud has 84 million points while the output model contains 933,000 triangles grouped into 3,096 planar regions. While the scheme described above only requires a small subset of the point cloud to be in memory at any given time, it is also important to make sure that the intermediary and output data structures are reasonably sized. Figure 4.20 shows a moderate-size model depicting the corridors in a hotel, represented with a 6.3 GB point cloud. The voxel carving, at a resolution of 5 cm, requires 12.7 MB of memory. If a conventional voxel grid structure were used to represent the entire bounding box, then 94.4 MB of memory would be required at this resolution.

Lastly, we can apply texture-mapping to these generated models to aid in assessing the quality of the geometry. A major motivation for producing models that are composed of large, planar regions is to aid in texture-mapping [85]. This method of texture-mapping produces a single image for each region in the model. Since all major building features – such as floors, walls and ceilings – are represented by large planar regions, there is a reduction in the number of seams in the applied texture. Objects that are significantly non-planar, such as the ladder that is along the side of the hallway, have some projection artifacts from the texture-mapping process, since these objects are represented with piece-wise planar geometry.



(a)



(b)

Figure 4.14: Voxel carving result for a large shopping mall: (a) Surface reconstruction of a shopping mall; (b) input point cloud. Resolution is 10 cm.

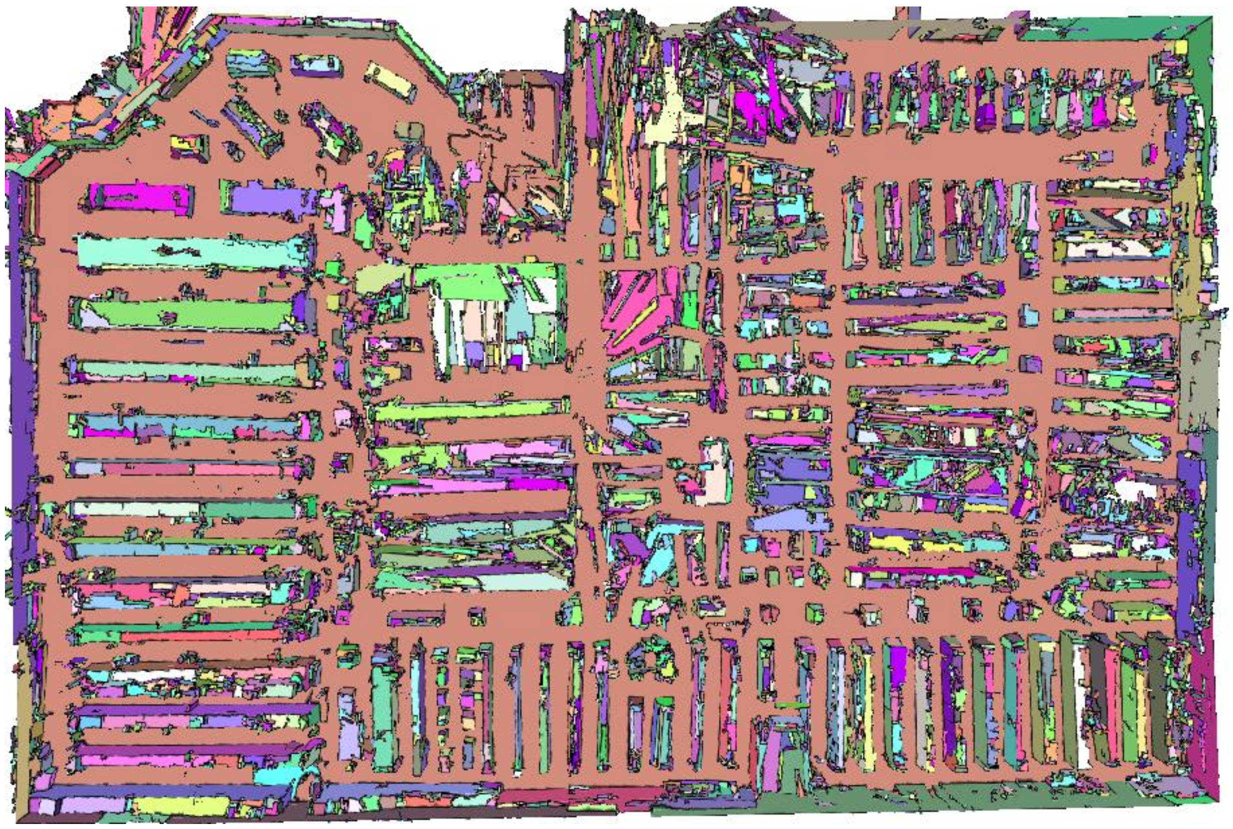
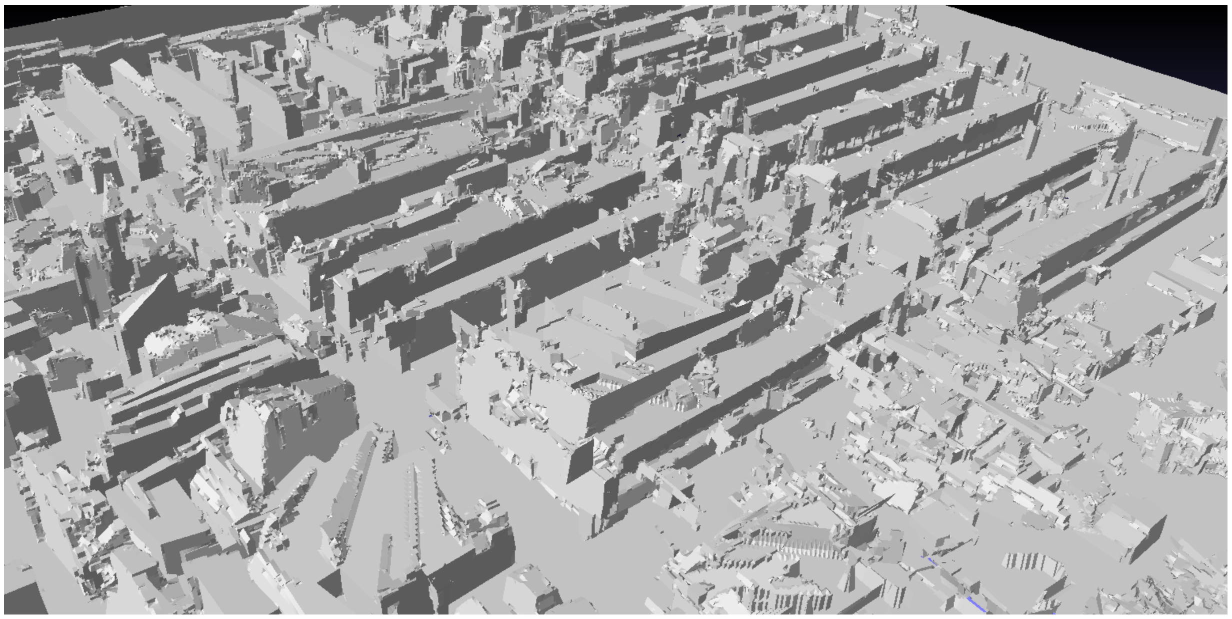
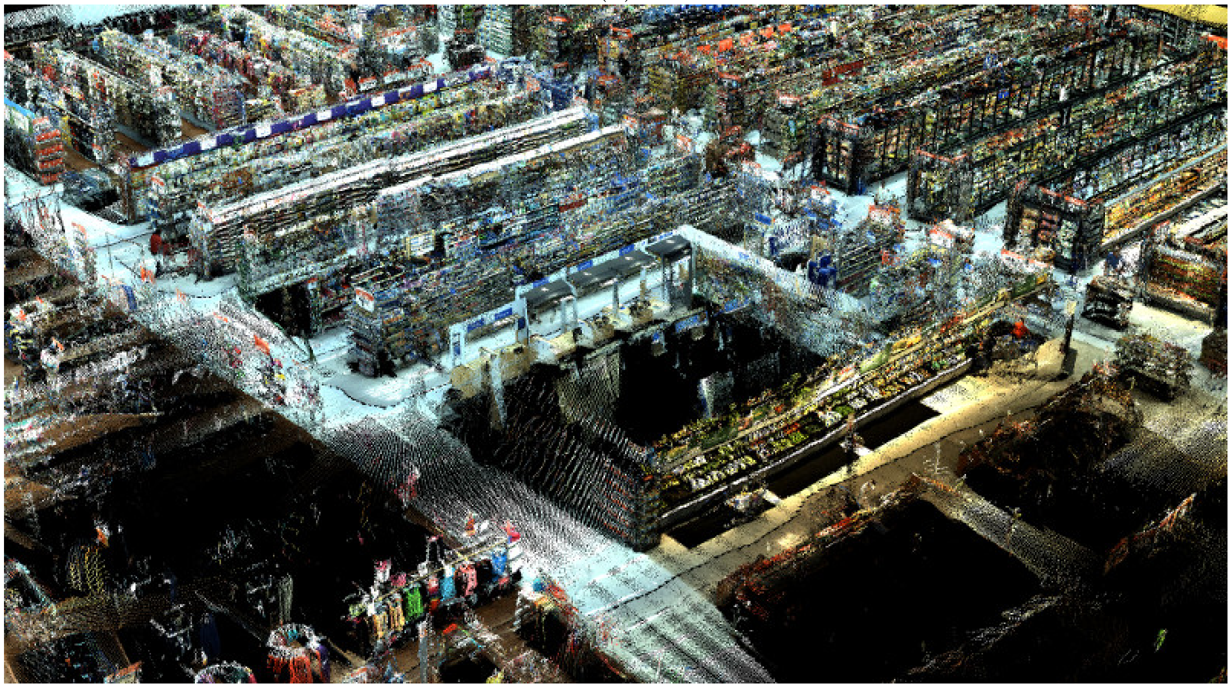


Figure 4.15: Surface reconstruction of a warehouse-sized retail shopping center using voxel carving technique, shown from top-down, with each region given random color. Resolution is 10 cm.



(a)



(b)

Figure 4.16: Surface reconstruction of a warehouse-sized retail shopping center using voxel carving technique: (a) view of shopping aisles in output mesh; (b) corresponding view of raw point cloud scans. Resolution is 10 cm.

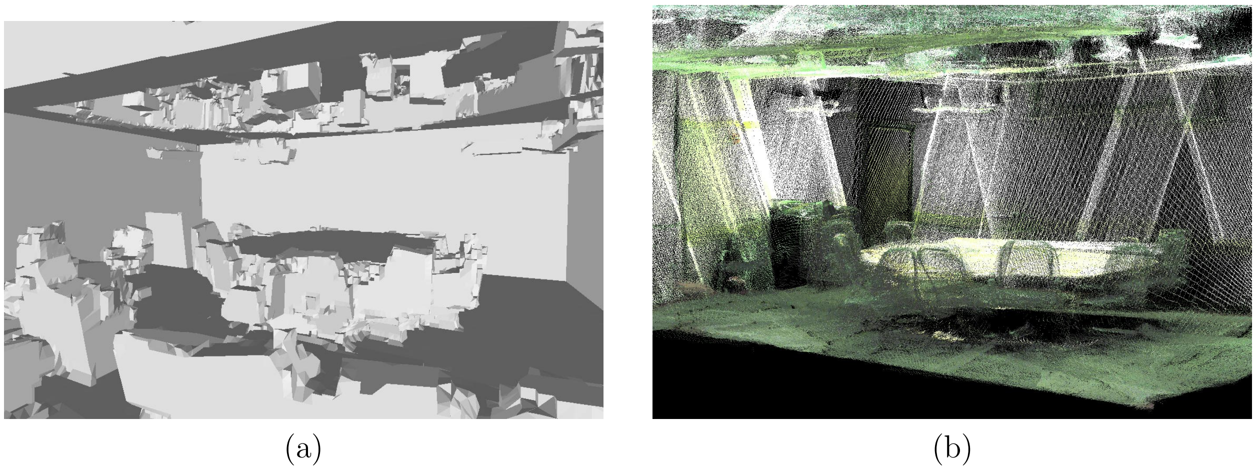


Figure 4.17: Voxel carving result of a 10.5m × 9.5m conference room with table: (a) The output mesh; (b) corresponding input point cloud. Resolution is 5 cm.

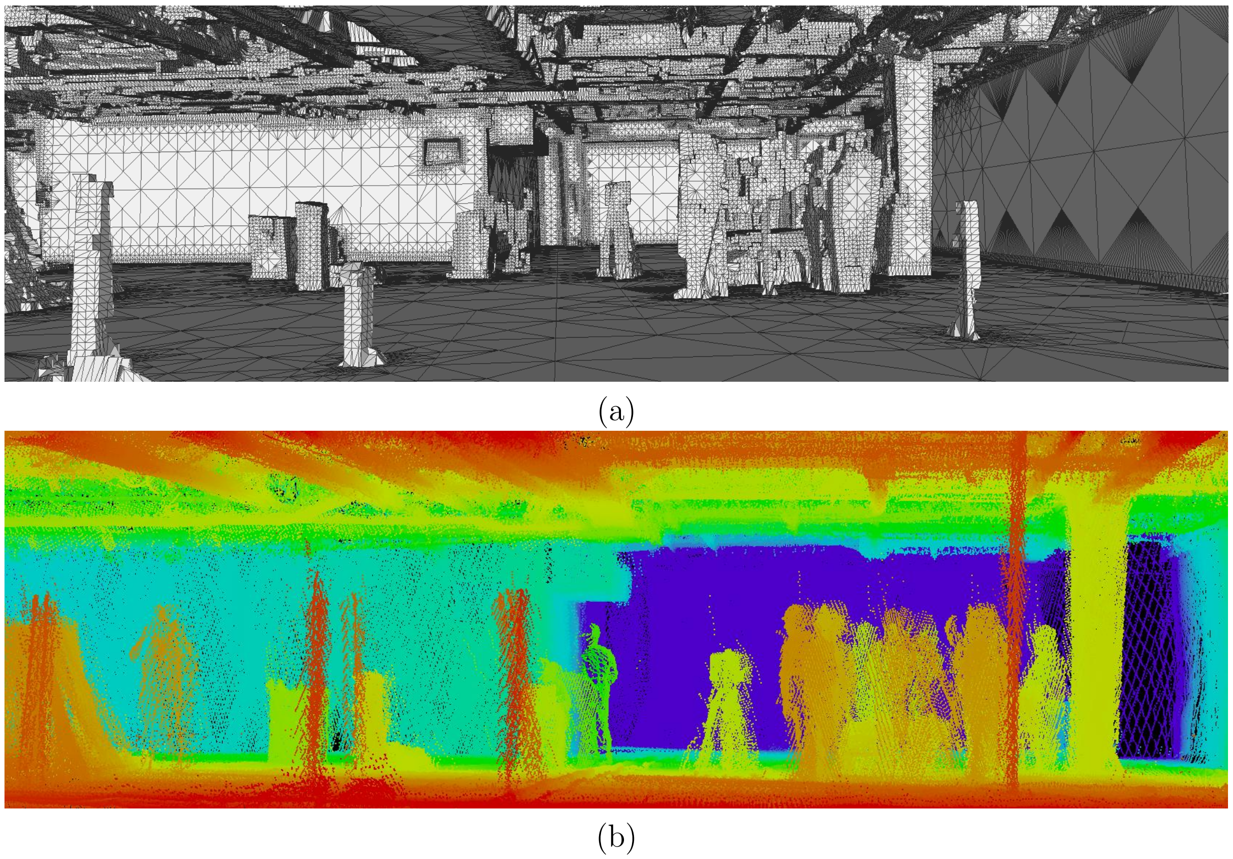


Figure 4.18: Voxel carving results of a construction site: (a) output mesh includes several people and a static scanning station; (b) point cloud of same view, colored by depth. Resolution is 5 cm.

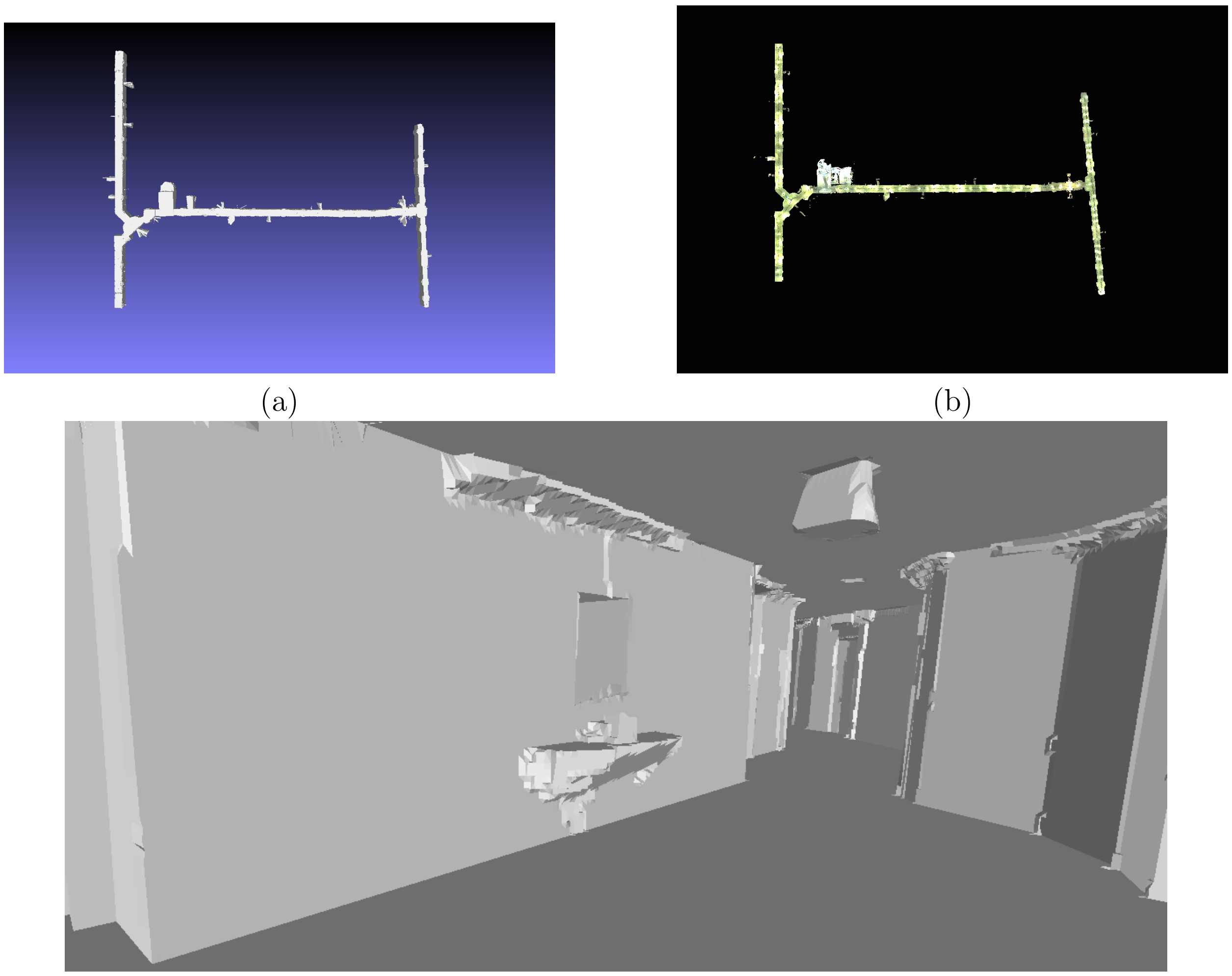
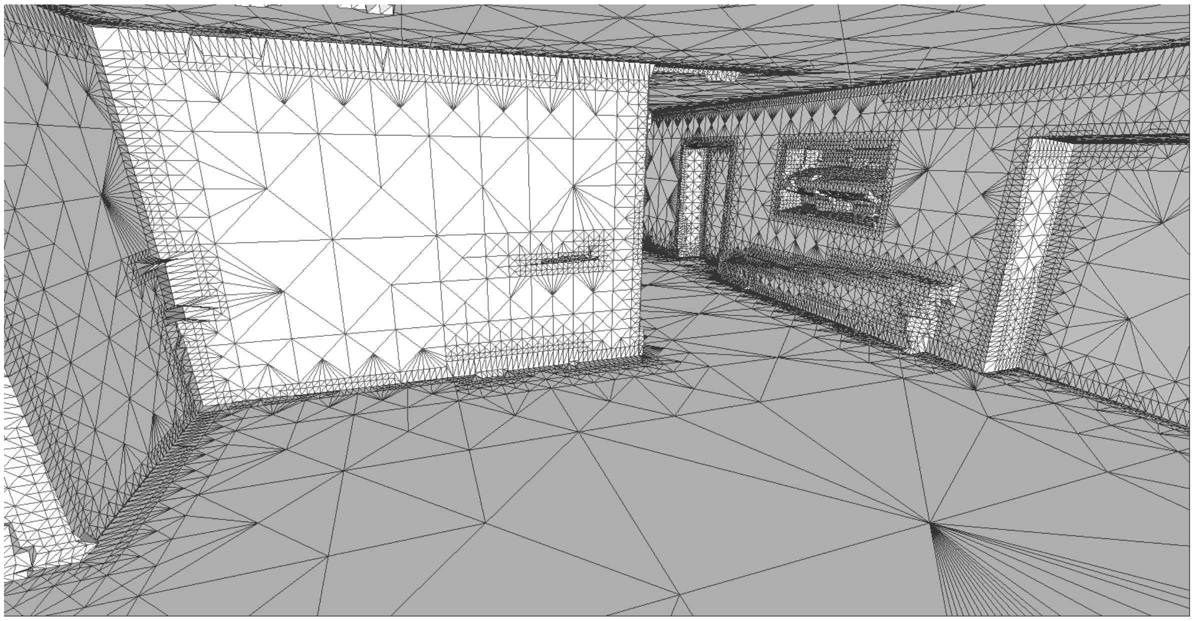
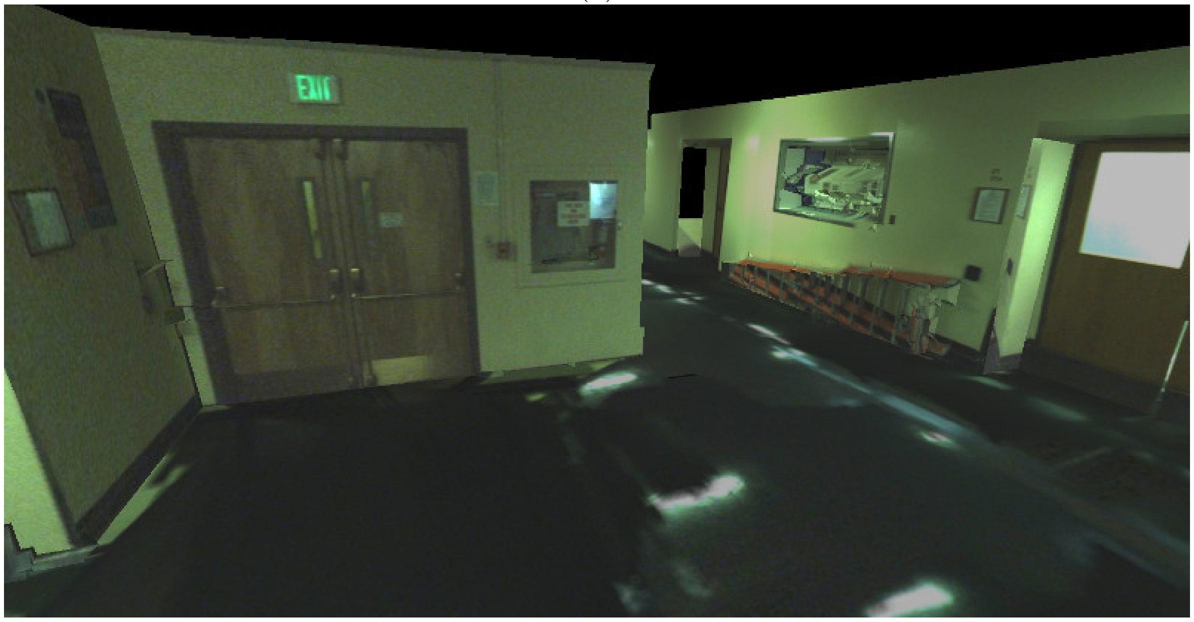


Figure 4.19: Voxel carving results of the hallways in a hotel: (a) output mesh of hotel hallway, showing full top-down view; (b) corresponding full top-down view of input point cloud; (c) output mesh of interior of hallway, showing a mirror and shelf. Model generated at 5 cm resolution.



(a)



(b)

Figure 4.20: A voxel carved model of the hallway of an academic building: (a) the geometry of the model; (b) with texture-mapping applied [85].

Chapter 5

Merging 2D and 3D Techniques for Indoor Modeling

So far in this dissertation, we have presented techniques to produce three different types of models for indoor building environments: 2D floor plans, 2.5D extruded models, and 3D fully complex models. In this chapter, we demonstrate how these different models can be combined to provide richer analysis of the scanned environment. In Section 5.1, we compare the quality of the techniques described in previous chapters. In the remaining sections of this chapter, we discuss mechanisms used to combine information from multiple of these techniques to improve the content.

For the first half of this chapter, we show how the processing steps used to generate the octree data structure, which represents a complex 3D model as described in Section 4.3, can be used to improve the quality of the 2D floor plan models discussed in Chapter 3. We discuss how using the octree data as input rather than raw point cloud values can improve the accuracy and aesthetics of the output floor plans. In Section 5.2, we describe how the octree structure can be utilized to improve how we split a multi-story model into individual levels. In Section 5.3, we discuss how the existing floor plan technique can be improved by using octree input. In Section 5.4, we show how these octrees can also be used for unique 2D visualizations of the scanned environment.

For the second half of this chapter, we discuss methods that improve the 3D complex mesh by incorporating the 2D floor plan model. In Section 5.5, we detail how floor plan intersection tests can be used to remove unsightly artifacts in the 3D complex models. Such artifacts are primarily due to scans that extend beyond the building environment, such as scans of the outdoors through windows. In Section 5.6, we discuss how combining floor plan and octree volumetric models allow for segmentation of objects and furniture in the environment. Such segmentation allows for more detailed meshing and improved resolution in the areas of the model that require it, fully watertight models of the interior furniture, and more accurate building models. Lastly, in Section 5.7, we show several real-world applications of this technique, show-casing how such analysis can improve model output quality.

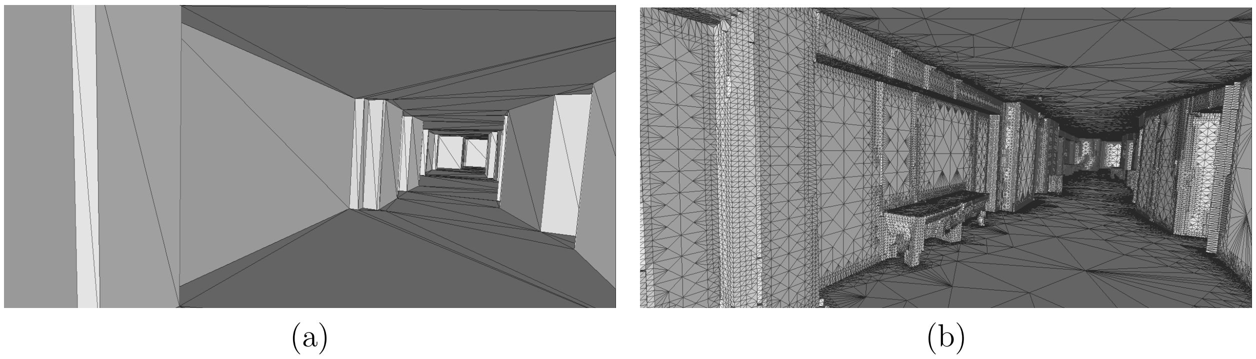


Figure 5.1: Comparison of models from (a) our 2.5D simple modeling approach with (b) the approach discussed in this chapter.

5.1 Comparisons of 2.5D and 3D Models

In this section, we compare the surface reconstruction techniques described in Chapters 3 and 4. In doing so, we analyze the size of the produced models and associated run times, compare our results to state-of-the-art methods, and discuss limitations. We also show examples of these models with texture-mapping applied [7, 13]. While this dissertation does not focus on the texture-mapping procedure, such processing is useful for determining the qualitative accuracy of the model geometry and how it aligns with the captured camera imagery. It is important to perform this comparison of methods, since later in this chapter we discuss how these different methods can be combined to improve the quality of our output models.

Our 2.5D approach produces simplified models when compared to surface reconstruction techniques that preserve fine detail with more complex output. The 3D carving method includes interior objects such as furniture. Figure 5.1 compares the models resulting from our 2.5D method described in Section 3.3 with that of the 3D building modeling technique, as described in Section 4.2, for hotel hallways [60, 93]. The two methods produce 2,944 triangles and 4.1 millions triangles, respectively.

We can also apply our reconstruction schemes to static scanning systems, even though they were originally designed to work with ambulatory acquisition systems [52]. As shown in Figure 5.2, point clouds generated from traditional static-scanning technologies can be used as inputs for our reconstruction techniques. The represented scans in Figure 5.2a are taken from the VmmlLab dataset of [73]. The original paper for this dataset details segmentation of point clouds, and not surface reconstruction, but the same group has also developed surface reconstruction approaches [61, 65]. This dataset contains 133 million points from three scan locations, representing a 20 foot $\times$ 30 foot room. We can convert these data to be used by our techniques by treating each scan location as a pose in the path of an assumed mobile system. A comparison to a state-of-the-art method is shown in Figure 5.2. Figure 5.2b depicts a model of the room using the techniques published by Mura et al. [61]. This model is represented with 12 triangles. The models shown in Figures 5.2c and 5.2d are constructed from the methods described in Chapters 4 and 3, respectively. The detailed model in Figure 5.2c is represented by 6.6 million triangles, while the simple model shown in Figure 5.2d is represented by 124 triangles. The method of Mura et al. is optimized for static

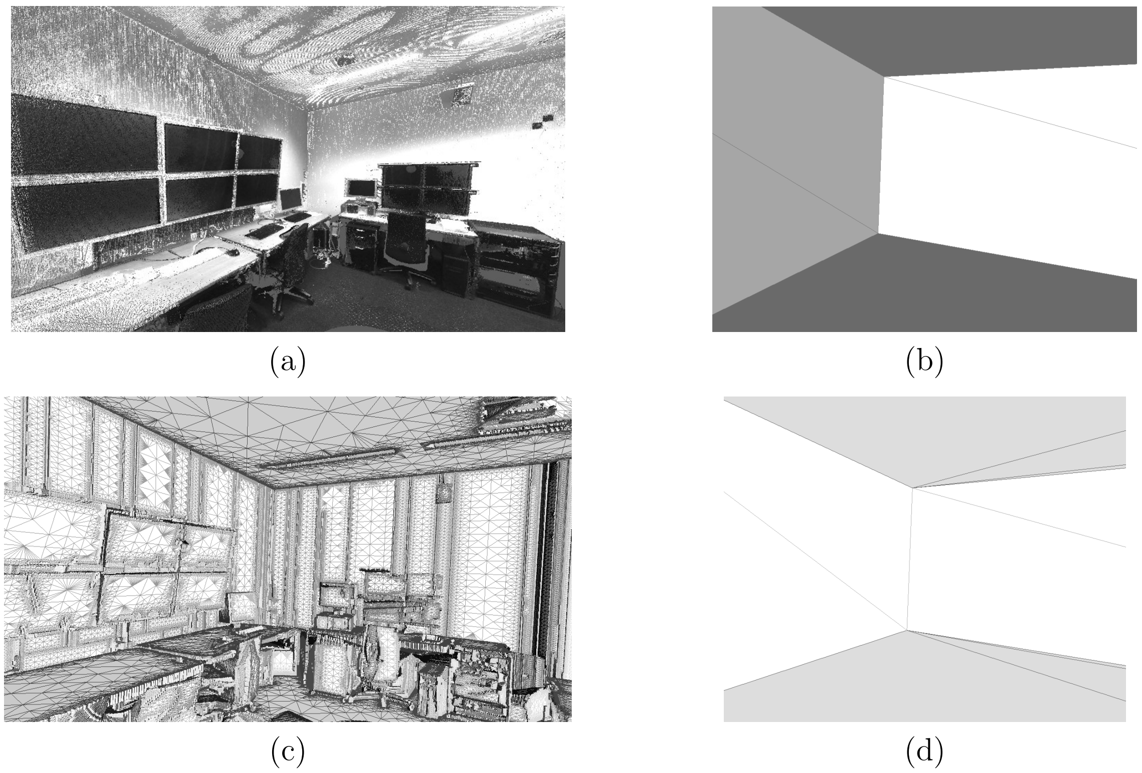


Figure 5.2: Comparison to state-of-the-art method: (a) Static point cloud scans from VmmlLab set of [73]; (b) reconstruction of dataset using method described in [61]; (c) reconstruction of dataset using method from this Chapter; (d) reconstruction of dataset using method from Section 3.3.

scans of small environments such as shown in Figure 5.2, and while it uses fewer triangles than our 2.5D method, both techniques still produce models of the same aesthetic quality and overall accuracy. The remainder of the examples shown in this section are generated from our ambulatory system. Note that the main difference between these two sources of scans is the level of mis-registration noise. Unlike ambulatory systems, which can produce mis-registration up to 27 cm [54], static scanning systems can be accurate to 0.25 cm [61].

Run-time analysis was performed on the dataset shown in Figure 5.3. The input to this dataset contains 25 million points. The code was run on a personal laptop with an Intel i7-2620M processor with 8 GB of RAM. These approaches are implemented in C++ as single-threaded programs. The voxel carving method described in Chapter 4, at 5 cm resolution, took 55 minutes of processing. The surface reconstruction of these voxels took 1 minute and 2 seconds. Existing voxel carving schemes processed similar models of 15 million points in 16 hours at the same resolution of 5 cm [5]. For a 2 cm resolution, voxel carving took 12 hours and 10 minutes and surface meshing took 9.5 minutes for this same dataset.

The same input point cloud was processed using the 2.5D modeling approach described in Chapter 3 on the same hardware. The processing step of extracting wall samples from the input point cloud took 84.3 seconds. Once these wall samples were generated, the process of generating the mesh took a total of 3.5 seconds. This step includes data i/o, floor plan

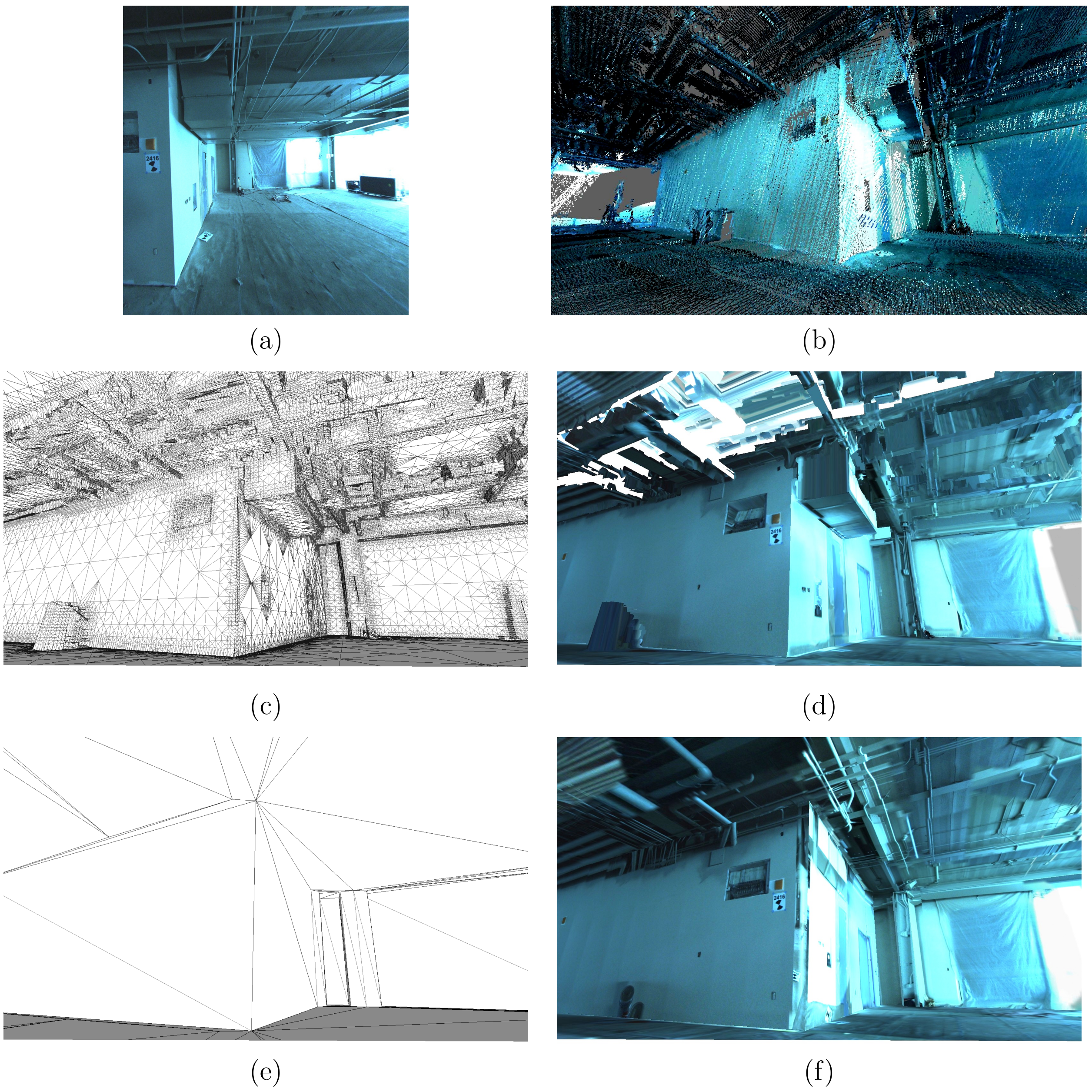


Figure 5.3: Close-up of models generated with the techniques described in this paper: (a) photograph of scanned area; (b) point cloud of scanned area; (c) surface carving model from Chapter 4; (d) surface carving with textures [13]; (e) extruded floor plan model from Chapter 3; (f) extruded floor plan with texturing [13].

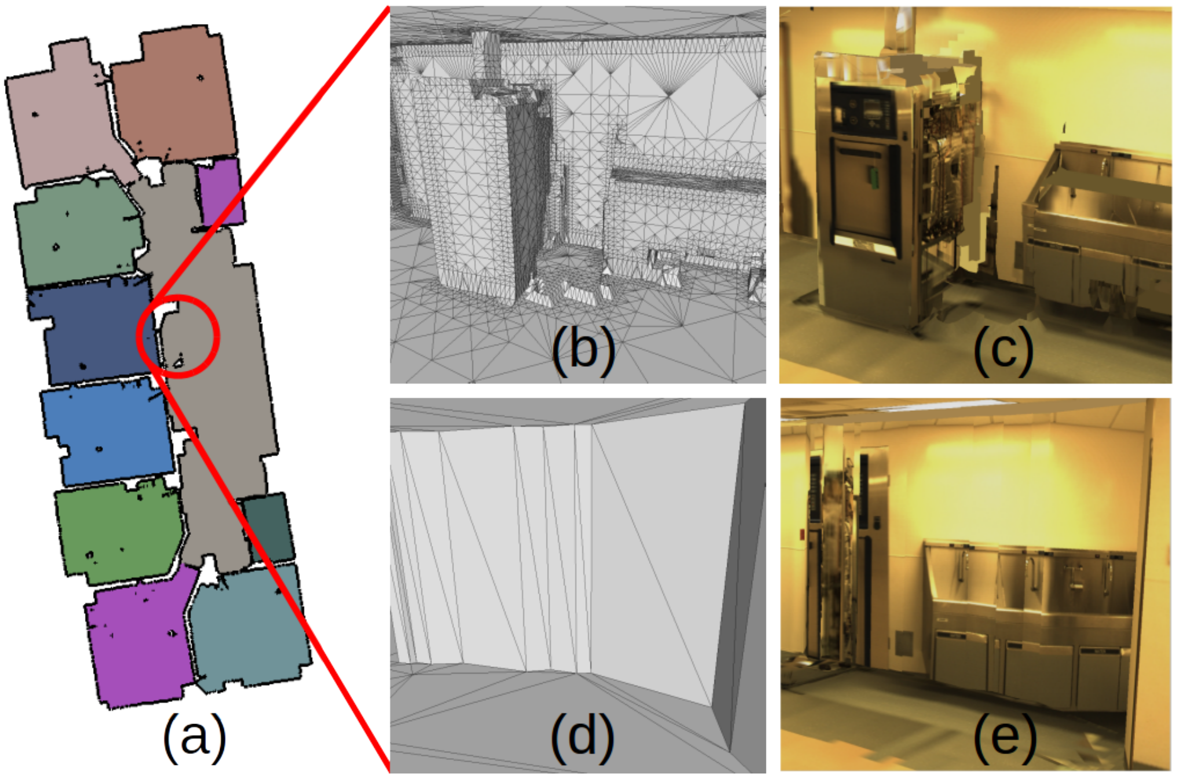


Figure 5.4: Surface reconstruction of a hospital's operating room area: (a) Floor plan of environment generated using method from Chapter 3; (b) surface carving model from Chapter 4; (c) surface carving with textures; (d) extruded floor plan model from Chapter 3; (e) extruded floor plan with textures. Note that the objects in the environment are not represented in the mesh of the extruded floor plan, so their texture is projected to the back wall of the room.

generation, and 3D extrusion of the floor plan. While our surface carving routine is efficient when compared to other similar techniques, generating a model using 2D information is orders of magnitude faster. Since the floor plan generation technique can be applied in a streaming fashion to input grid-map data, it could be able to run in real-time for compatible SLAM systems.

The texture-mapping process requires more computation time than the surface reconstruction schemes [13, 85]. Using a single-threaded implementation on the same hardware, texturing the output surface of the voxel carving method as shown in Figure 5.3 took 10 hours and 44 minutes. The texture-mapping of the surface generated from the floor plan extrusion approach took 113 minutes. The shorter time to texture-map this surface is due to the far fewer number of elements in the output mesh.

While using an extruded floor plan mesh to generate a texture-mapped model is far more computationally efficient, there are also several limitations. Specifically, removing the geometry for furniture and other objects in the environment creates a disparity between what is seen in the camera imagery and the reconstructed mesh. As such, the texture for these objects is incorrectly projected onto the wall surfaces behind the objects, as shown in

Figure 5.4. The advantage of a fully-3D meshing method is the preservation of fine detail in the geometry, while the advantage of the floor plan extrusion method is speed, simplicity, and a higher proportion of large, planar surfaces that allow for efficient texturing.

Figure 5.5 shows the results of modeling the scans collected in a hotel lobby. The input point clouds for this model consisted of 70.4 million points, which comprised 5.16 GB on disk. The generated models cover 2,317 square meters, or about 25,000 square feet. The surface carving model, as shown in Figures 5.5a and 5.5e, is represented by 2.65 million triangles. The extruded floor plan, as shown in Figures 5.5c and 5.5g, is modeled with 2,944 triangles. This reduction by a factor of a thousand means that finer details in the model such as furniture or drop ceilings are not present, but can aid in many applications, including texture-mapping.

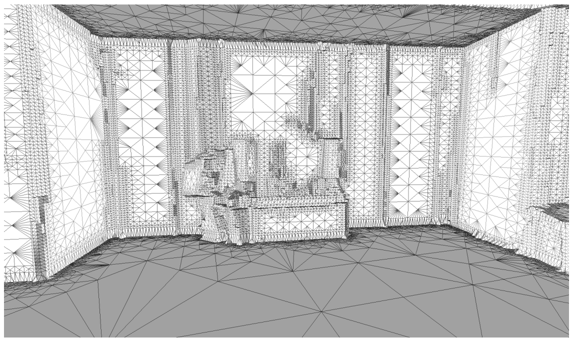
The size of the resulting texture-mapped models is much larger due to the high-resolution textures generated from camera imagery. The texture-mapping of the surface carving model, as shown in Figures 5.5b and 5.5d, is 1.45 GB on disk comprised of textures for 1,277 distinct surfaces. The texture-mapping of the extruded floor plan depicted in Figures 5.5f and 5.5h takes up 488 MB on disk with 566 surfaces. The largest surface for texturing is the floor of the main lobby area, whose texture is represented by a 14210×10555 image. A video fly-through of these models can be found online [99].

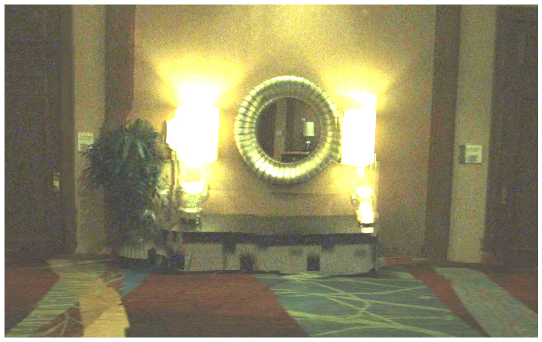
While the 2D floor plans, 2.5D extruded models, and the full 3D complex models all have their respective uses and applications, they represent the building in fundamentally different ways. These modeling methods can be used to enhance each other, using one method to improve the quality of another. In the remaining sections of this chapter, we discuss how analysis of the 3D complex model can be used to improve the results of the 2D floor plans and their corresponding extruded models. We then discuss how the opposite can also be done, where the 2D floor plans are used to improve the quality of the 3D models, including added analysis about separating the 3D geometry of furniture from the rest of the building model.

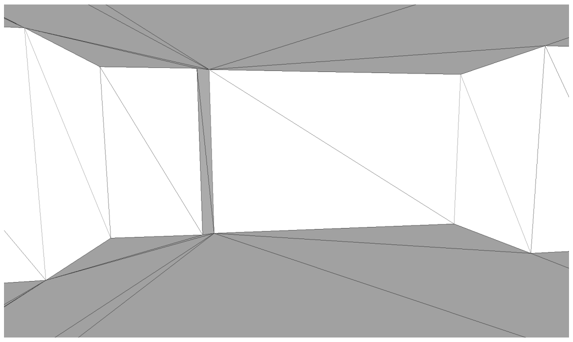
5.2 Level-Splitting with Octree Structures

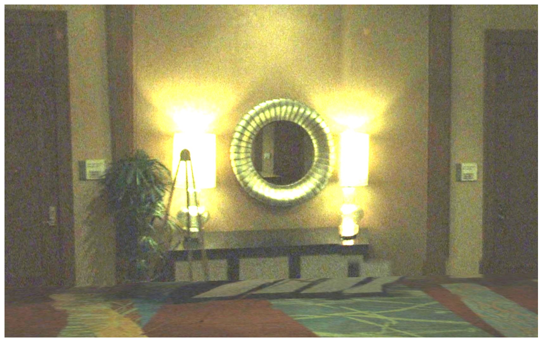
In Section 2.5, we discuss how our floor plan generation methods require the model to first be split into separate building levels before the floor plan geometry can be reconstructed. In this section, we show how this splitting can be performed more accurately by using the octree model to identify separations between levels, rather than using the raw point clouds. This method of level splitting finds horizontal surfaces in the point cloud data by performing a histogram of the point elevations and searching for peaks. An alternate method to perform level splitting on a building model is to find horizontal regions in the boundary of the octree data structure generated in Section 4.3.2.

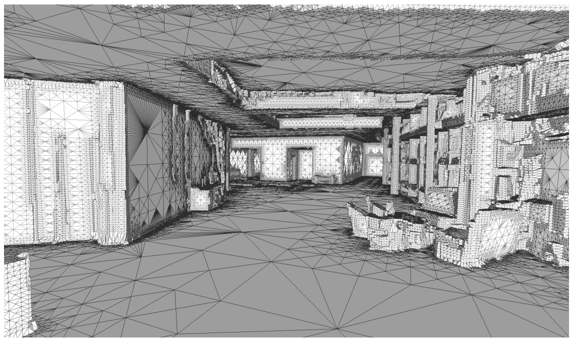
We first produce an octree data structure of the scanned environment. Next, we perform planar region fitting on the boundary faces of this octree, as described in Section 4.3.3. Next, we select all horizontally-aligned regions within the model, discarding any regions whose normals are more than 5 degrees away from vertical. We then discard any regions that do not meet a certain surface area threshold. This step allows us to only consider the dominant horizontal surfaces when finding the elevations of floors and ceilings, which

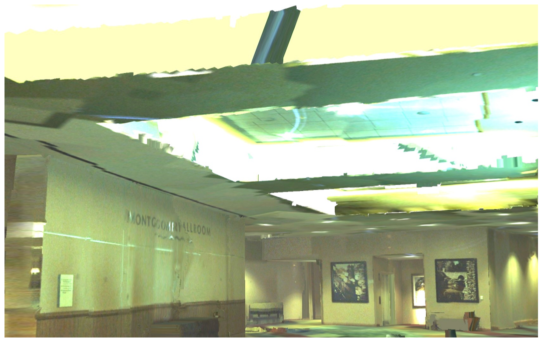

(a)

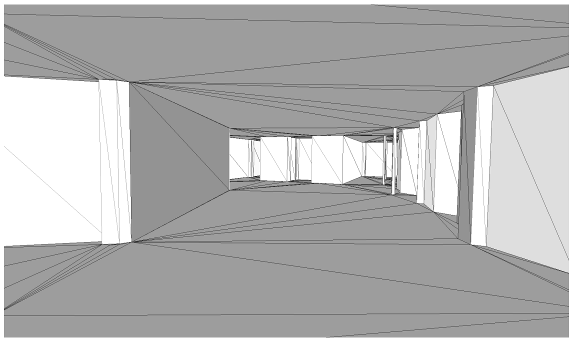

(b)


(c)


(d)


(e)


(f)


(g)

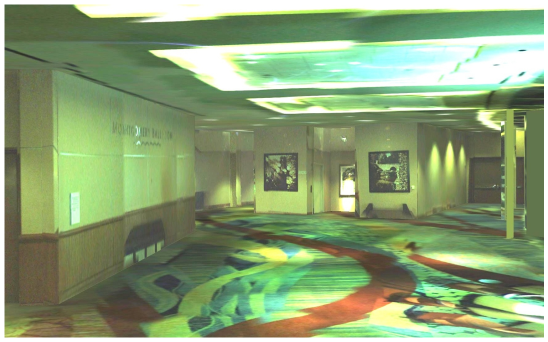

(h)

Figure 5.5: Modeling results of hotel lobby: (a) Desk area modeled with method from Chapter 4; (b) texture-mapping applied; (c) area modeled with method from Chapter 3; (d) texture-mapping applied; (e) main lobby area modeled with method from Chapter 4; (f) texture-mapping applied; (g) area modeled with method from Chapter 3; (h) texture-mapping applied.

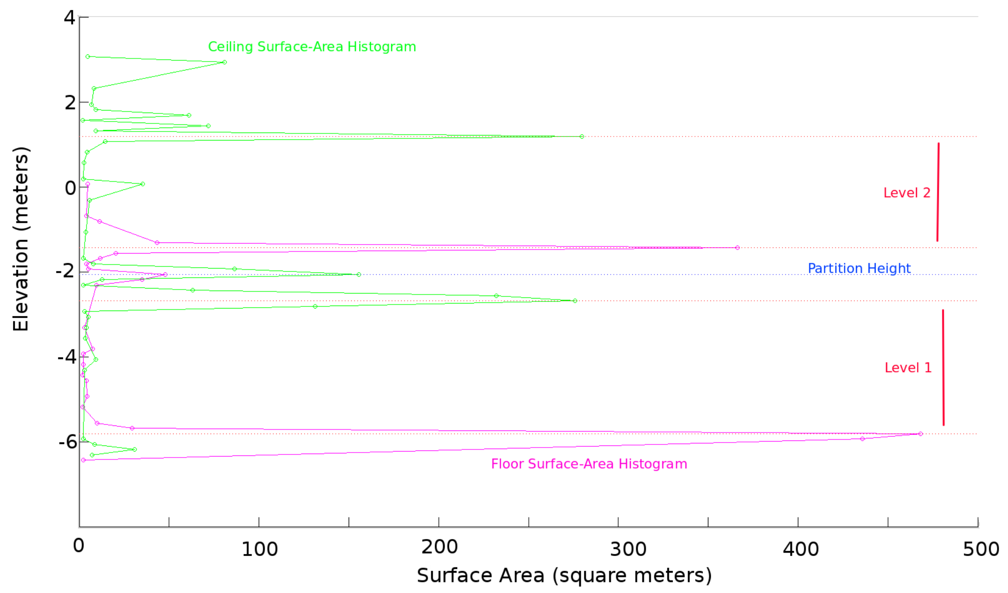


Figure 5.6: An example of using the octree data structure to perform level-splitting in a two-story office building. This graph shows a histogram of surface area of horizontal surfaces with respect to elevation. By finding peaks in floor and ceiling surfaces, we can estimate where one level ends and another begins.

provides a much cleaner histogram representation than if every point in the point cloud were used. We can also separate floors, whose normals face upward, from ceilings, whose normals face downward, more accurately than performing histogramming with a point cloud, whose components have no normal information. Additionally, a histogram of points assumes a relative uniform sampling of each surface. If the operator traversed one level significantly slower than another level, then the first level will have more points, skewing the histogram. By performing the histogram on surface area, we can assure that the geometry will be analyzed independently of how it was traversed.

Figure 5.6 shows an example of this analysis on a two-story dataset. We plot the total surface area of horizontal planar regions found at each elevation. Elevations with a local maximum in floor surface area are likely to denote the beginning of a level and those with a peak ceiling surface area are likely to denote the end of a level. We then deduce the total number of levels by finding the pair of floor and ceiling heights for each level. The model can be split by level at the elevation halfway between the ceiling of the lower level and the floor of the upper level. The visualization of the two levels of this building model are shown later in this chapter, in Section 5.4.

5.3 Generating Floor Plans from Octree Structure

Several existing methods of surface reconstruction of building environments ignore low-level detail and aim to only model the primary floors, walls, and ceilings within the environment [2,43,44,55–58,61,64–69]. In this section, we describe how we use the populated octree from Section 4.3 to generate this simplified model. We use the same technique discussed in Section 3.3 to produce 2D floorplans of the environment and extrude 2.5D models using the height information from each room [60]. Rather than applying this technique directly on wall samples generated from raw point cloud scans of the environment, we instead produce the input data for this method from our populated octree. We show that this approach yields a more accurate model that is both well-aligned with our octree and more aesthetically consistent with real-world geometry. The remainder of this section describes how we produce 2D wall samples from the octree, use these samples to generate a 2D floorplan of the environment, and create a 2.5D simplified model from the floorplan.

5.3.1 Wall Sampling From Octree Structure

To generate a floor plan that can be extruded into a simplified 2.5D model, we first need to generate a set of wall samples in the environment. Wall samples are a set of points in 2D space that are locations with high likelihood of being wall positions. This set of points is used by the 2D floor plan generation procedure as input data [60]. In Section 3.1, these wall samples are generated either from the output grid-map of a particle filter or by analyzing a 3D point cloud of the environment.

The above two approaches can erroneously misclassify some surfaces as walls. Any long vertical structure contributes to the wall samples, causing environment features such as bookshelves, doors, or other objects to be detected as walls. By finding wall position estimates from the octree, rather than directly from the point cloud, we use topology information to ensure that detected surfaces are large and planar. We show that this approach not only produces a floor plan better aligned with the complex geometry of the octree, but also one that is less affected by clutter, such as furniture, than when using the point cloud directly to generate the floor plan.

The first step of generating wall samples from the octree is to identify large planar surfaces. We cluster the boundary faces of the octree into planar regions as discussed in Section 4.3.3. Figure 5.7a shows the initial boundary regions of a model, with each initial region depicted as a separate color. These regions are iteratively merged in the process described in Section 4.2.4. The result is a single planar region for each dominant surface of the model, as shown in Figure 5.7b.

We then filter the regions based on the normal direction of the fitting plane, keeping only the surfaces that are vertically-aligned, as shown in Figure 5.7c. Any surface whose normal vector is more than 5 degrees off of horizontal is rejected. This approach makes the reasonable assumption that walls are vertical, which fits almost all building types. We also filter wall surfaces based on their surface area and vertical height. Regions that are less than 0.05 square meters or have a vertical extent less than 1 meter are not considered.

The output regions have gaps corresponding to the portion of the walls hidden behind any furniture in the model. We expand the represented geometry of each wall to include any

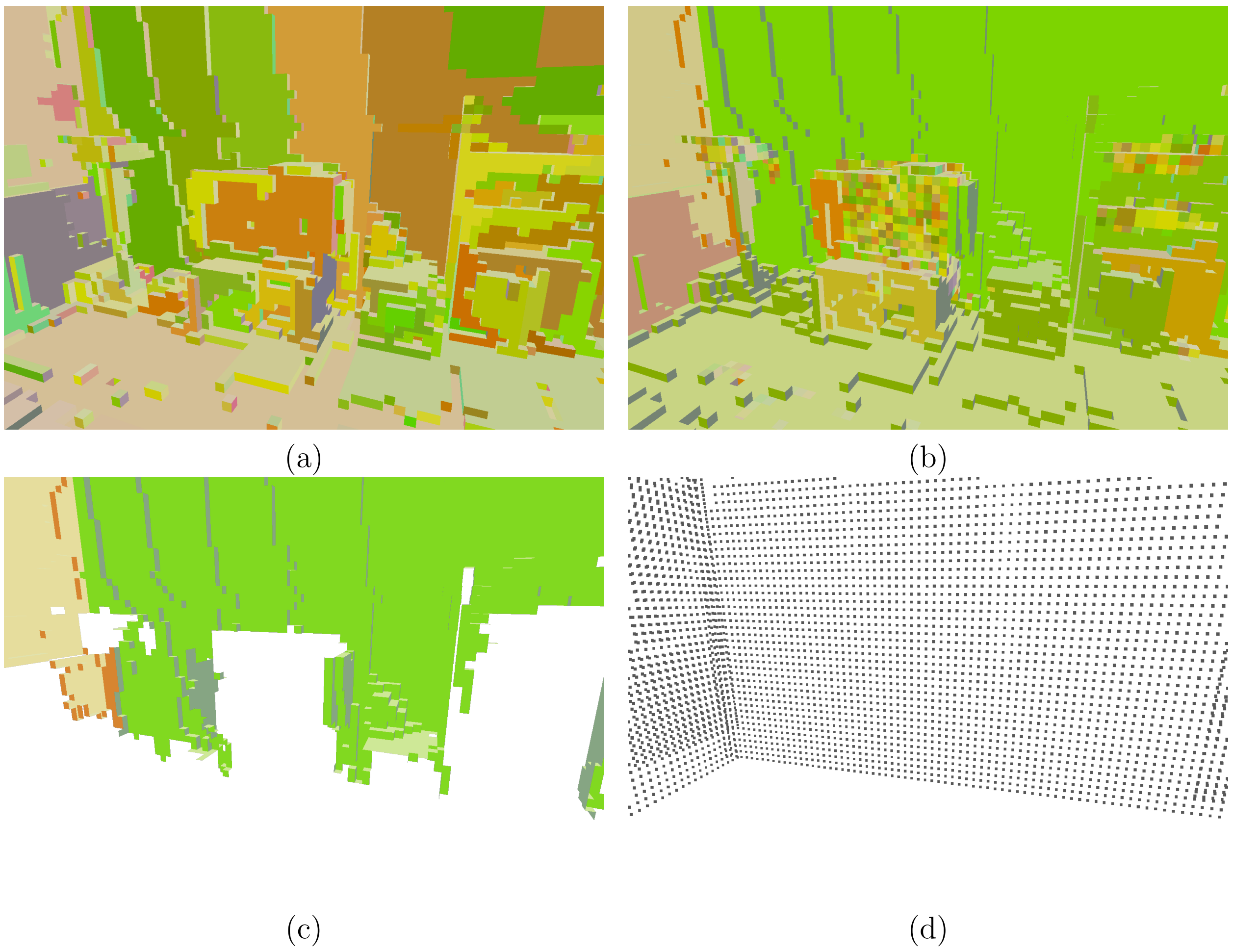


Figure 5.7: Generating wall samples from an octree: (a) initialize regions on the octnodes' boundary faces; (b) perform region growing to form large planar regions; (c) filter out wall regions; (d) generate points along planar regions to make wall samples.

exterior points that are within the 2D convex hull of each wall planar region to counteract these occlusions. The check for whether points are *exterior* is performed using the populated octree. Figure 5.7d shows a set of these points, sampled uniformly, across the wall plane.

Once we obtain these 3D wall positions, we can convert them to the 2D wall sample representation described in Section 3.1. Each column of sampled points along the 3D surface becomes a 2D wall sample in the XY-plane. The vertical extent of this column is used to estimate the height of the wall at that position. These wall samples are a cleaner representation than those extracted from point clouds directly, since any surface variations have been flattened beforehand. Since the entire surface has been interpolated behind objects in the environment, the wall representations are more robust to clutter in the environment, which leads to fewer false positives in the presence of large obstacles such as bookshelves or doors. These attributes can be seen by comparing wall samples develop with this new method as opposed to wall samples taken directly from point clouds, as shown in Figure 5.8.

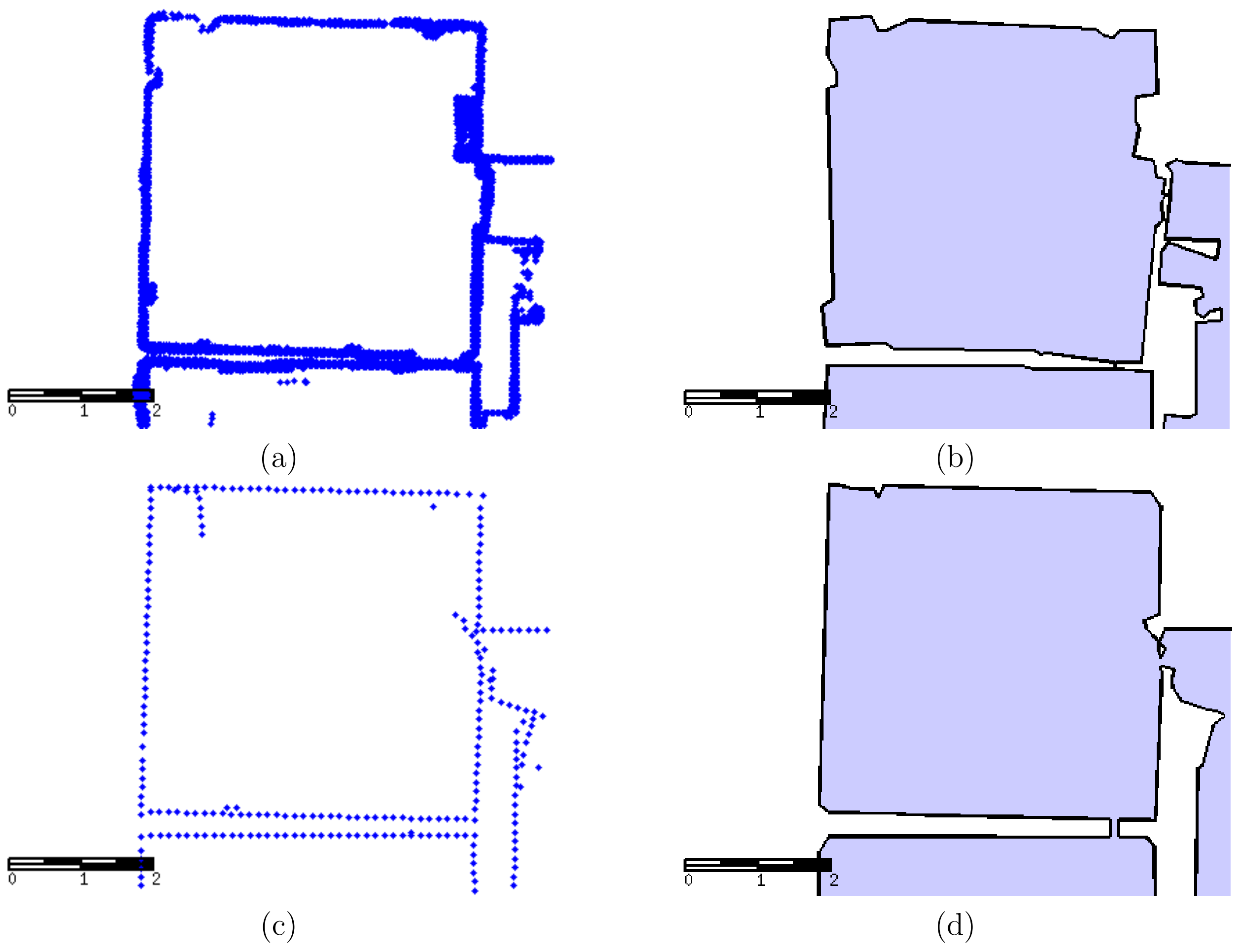


Figure 5.8: Comparison of wall samples and floorplans: (a) wall sampling generated from original point cloud; (b) corresponding floorplan; (c) wall sampling generated from octree; (d) corresponding floorplan. All units are in meters.

5.3.2 Floor Plan Generation

Once wall samples are generated via octrees for a model, they can be fed into the 2D floor plan generation method discussed in Section 3.3. This method produces a watertight 2D model that defines the scanned area. Figure 5.8 shows a comparison of floor plans generated with the octree versus floor plans generated with the raw scan points. Figures 5.8a and 5.8b show the wall samples and the floor plan generated from the raw scan points, respectively. Obstacles in the environment such as furniture are not properly removed, causing the output walls to be noisy. Figures 5.8c and 5.8d show the wall samples and floor plan generated from the octree. Since the region merging allows more sophisticated separation of large planar surfaces, the furniture in the environment is properly removed and the output model is cleaner. As an example, the upper-right corner of this model contains a large bookcase, which appears in the wall sampling in Figure 5.8a and is propagated to the floor plan in Figure 5.8b. This bookcase is correctly removed from the wall samples generated from the octree in Figure 5.8c, which means the wall is more appropriately represented in the floor plan shown in Figure 5.8d.

The approach discussed in this section uses the generated octree to produce improved wall samples, which in turn can be used to generate 2D floor plan models. When compared to floor plans generated from raw point cloud scans, our proposed approach produces not only better alignment to the complex geometry, but also a cleaner floor plan [60]. We demonstrate this contrast with Figure 5.9, which represents a scan of a 14,079 square foot office area with over 50 rooms. This data acquisition took 25 minutes and processing the octree took 84 processor hours. An example floor plan from the previous method is shown in Figure 5.9a. This floor plan has several artifacts caused by clutter and furniture in the environment. Locations (1) and (3) show rooms filled with large amounts of objects, causing holes in the floor plan. Locations (2) and (4) show rooms with a single large object that occluded part of a wall, causing a incorrect notch in the floor plan. The floor plan of the same environment generated by our proposed technique is shown in Figure 5.9b. This floor plan correctly separates the rooms of the environment and does not have the same artifacts as in the previous method. Additionally, long hallways in the building are correctly represented as one room, rather than being split into several small segments.

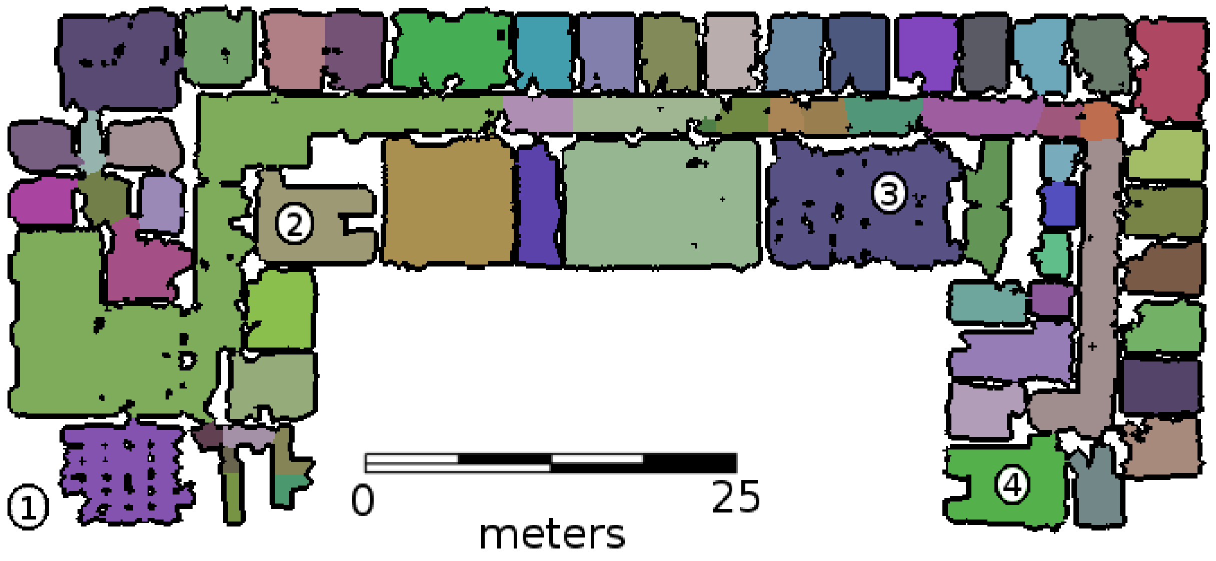
In addition to the examples shown here, more examples of improved floor plans are shown along-side 3D models in Section 5.7.

5.3.3 2.5D Extrusion

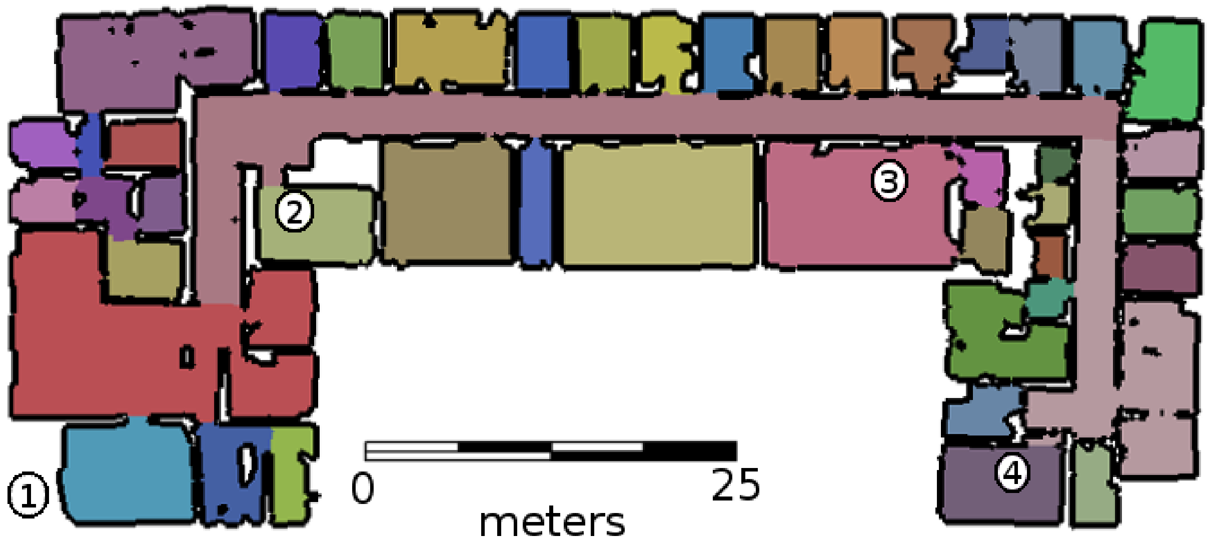
Height information is stored in the floor plan model structure, so a 2.5D model can be extruded, resulting in a simplified representation of the floors, walls, and ceilings in the environment. As originally discussed in Section 3.3.4, the floor plans are extruded into 2.5D models by assigning a single pair of floor and ceiling heights to each room in the environment. The values for these heights are taken by finding the median floor and ceiling heights across all wall samples in each room. As shown in Section 3.3.4, if the wall samples are generated directly from the point cloud, their estimates of heights are very noisy. This effect is due to the heights of wall samples originally being derived from the extent of the vertical surface. If a piece of furniture was blocking part of a wall, or a vertical surface was otherwise occluded, then the wall sample may not represent the full floor-to-ceiling height of a room.

By generating wall samples from the boundary surfaces of the octree, we are able to generate the extruded heights of the floor plans more robustly. Rather than directly examining the vertical extent of a wall surface, we instead observe the elevations of large horizontal surfaces in the environment. These surfaces are likely to be floors and ceilings for each room. We use the topology of the octree to find the closest large, horizontal surface to each candidate vertical wall surface. When extracting wall samples, we save the elevations of the nearby floors and ceilings, instead of relying on the vertical extent of the wall itself.

This process allows for higher accuracy in the heights of each room, since it is robust against clutter in the environment. An example is shown in Figure 5.10. This figure shows a kitchen environment, as seen in the reference photograph in Figure 5.10a, with example wall surfaces detected using both methods. Figure 5.10b shows the original wall surfaces, whose vertical extents are estimated by the vertical extent of the original planar region found in the octree boundary. Figure 5.10c shows the new method of computing wall surfaces, whose vertical extends are estimated by first classifying large horizontal surfaces as floors



(a)



(b)

Figure 5.9: Comparison of floor plans of large office environment: (a) floor plan generated from raw point cloud scans; (b) floor plan generated from octree proposed here. The octree floor plan has much fewer artifacts due to clutter or furniture, as shown in areas "1" through "4".

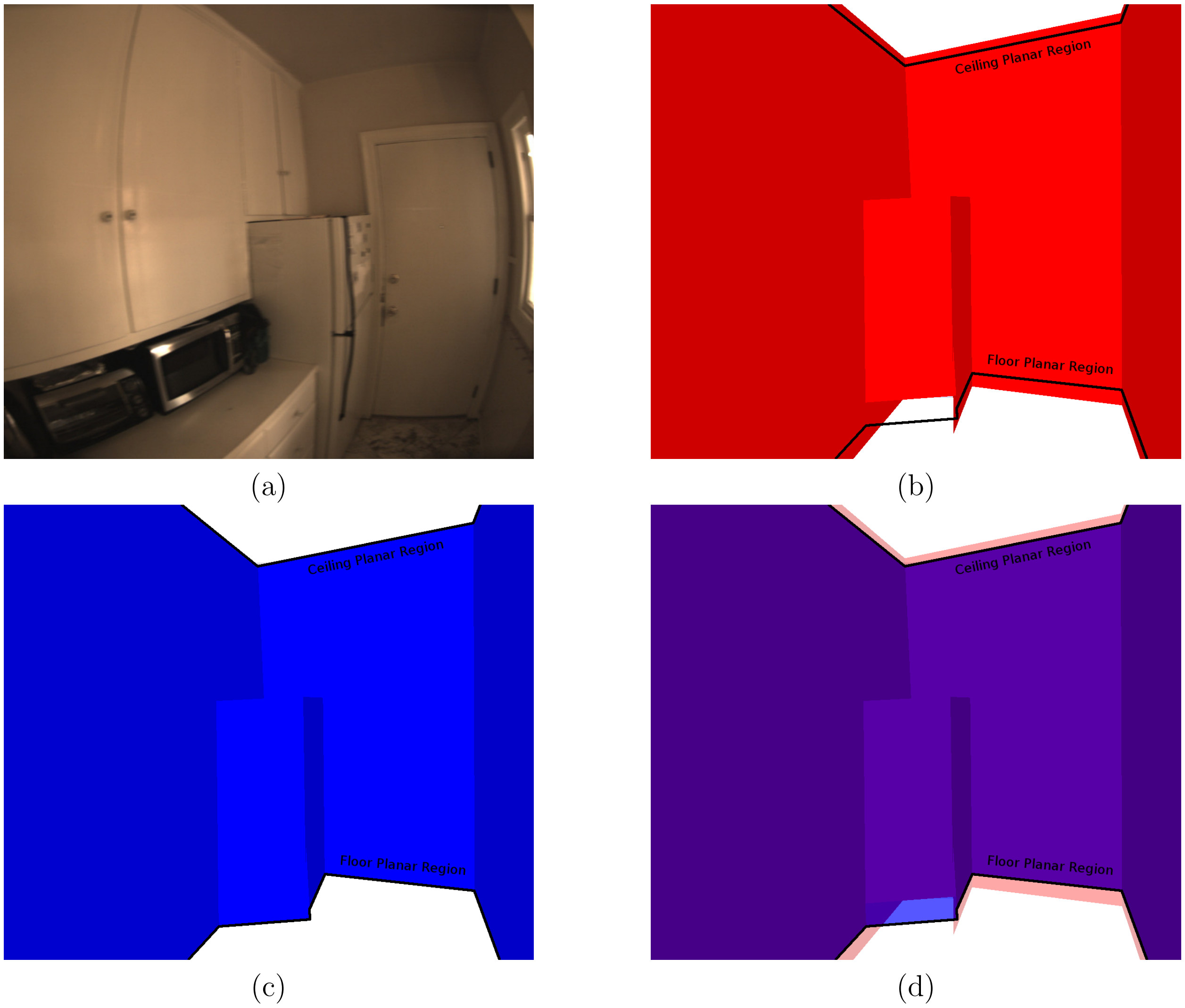


Figure 5.10: Example showing improvement in estimating wall heights for extruded floor plans: (a) reference photo of environment; (b) the original wall surfaces, whose heights are determined by the surfaces' vertical extents; (c) the modified wall surfaces, whose heights are determined by the elevation of floor and ceiling planar regions in the room; (d) overlay of the original and new wall surfaces.

and ceilings, then adjusting the wall heights by the elevations of these surfaces. Lastly, Figure 5.10d shows both sets of surfaces overlayed. As seen, the original surfaces are not consistent with the actual height of the room, whereas by aligning the heights of walls with the elevations of floors and ceilings, we can produce a more consistent floor plan extrusion that is more accurate as opposed to the original model.

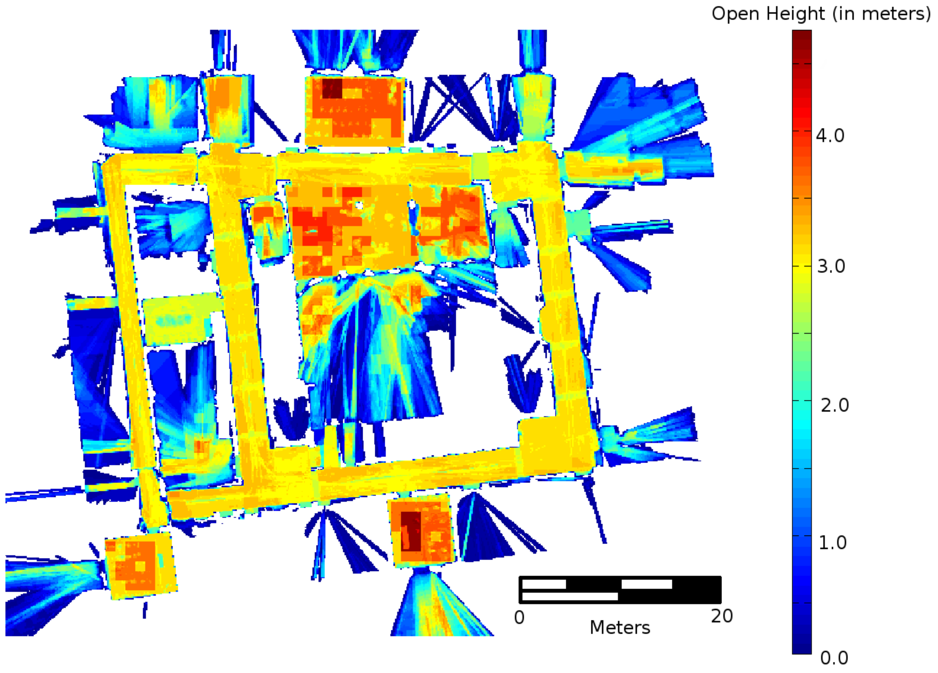
5.4 Top-Down 2D Visualization of Occupancy using Octree Structure

The octree data structure described in Section 4.3.2 represents the full 3D volumetric information about a scanned environment. These values can span across several scanned floors of the environment and are used to generate the fully-detailed 3D models shown in Chapter 4. We can also compile these data to present a 2D visualization of the modeling environment, as discussed in this section.

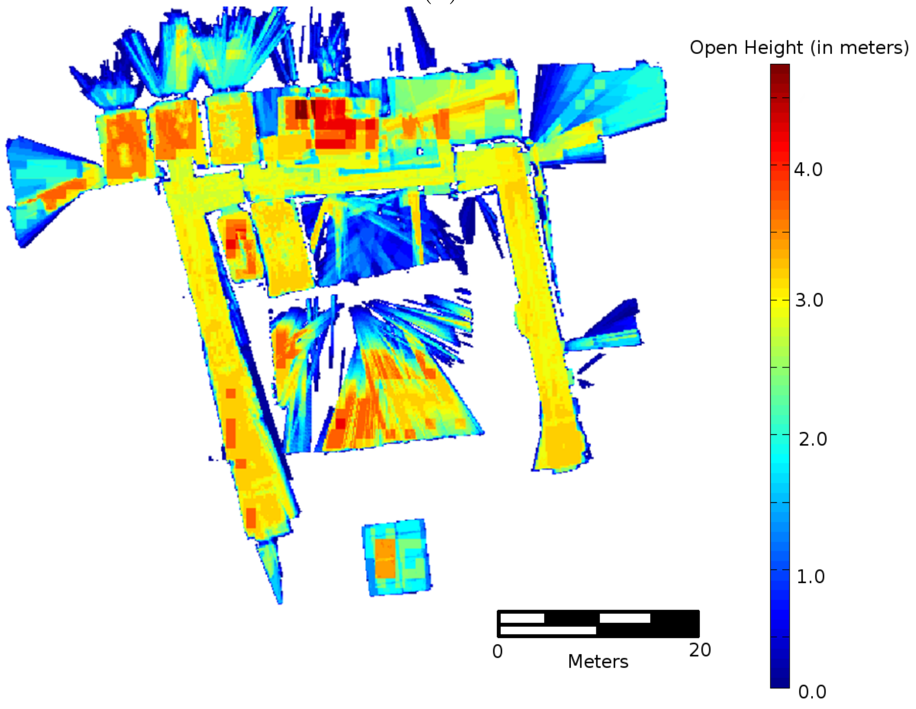
We generate a top-down, 2D histogram of the values stored in the octree. In each node, a value for the probability of interior is stored. By generating a sum for each xy-position of these values, which intersects all nodes along a vertical column, we can determine the expected "open height", in meters. This value indicates how much of the model is labeled *interior* at each xy-position, yielding a map of the 2D representation of each level.

Figure 5.11 shows an example of this top-down histogram. This figure uses the same model that was represented in Figure 5.6, which represents a two-story academic office building. Since these models show a representation of all volume that was scanned within the octree, the output includes partial scans of the exterior area of the building, which were observed through windows and doors to the outside. As we discuss later in Section 5.5, these artifacts can be removed by performing an intersection test with the generated floor plan of the environment.

We can also view a close-up representation of one room in this same area, as shown in Figure 5.12. This room is also present in the upper-left corner of Figure 5.11b. This visualization represents all the features in the room environment, including detail of the furniture within the room. This level of detail includes the curvature of the cushions and armrests on the couch, and the shape of the legs of the pool table. Note that the histogram picks up the detail of the light fixtures on the ceiling as well, which show up as three bars across the length of the room. The bookshelf on the right-side of the room shows up with less clarity than other objects in the environment. This is due to most of the bookshelf being *interior* area: open shelves facing forward. This visualization shows how analysis of the octree structure results in more accurate wall positions than wall samples taken from the point cloud scans directly.

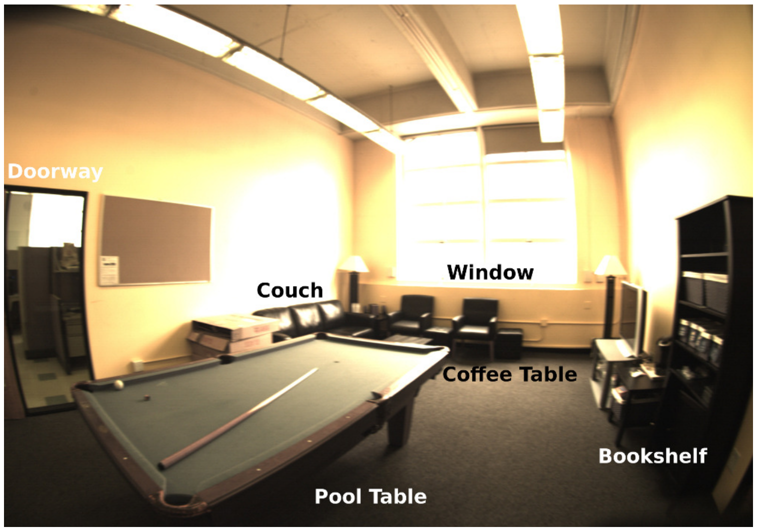


(a)

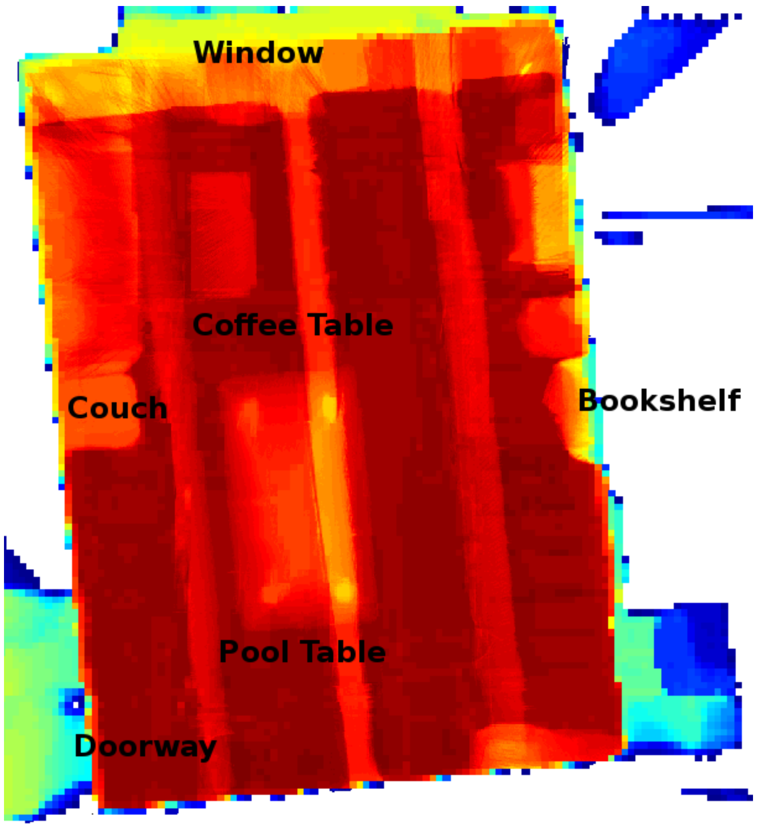


(b)

Figure 5.11: An example top-down histogram of a two-story model, generated from a volumetric octree structure: (a) the first level; (b) the second level. These plots are colored by the amount of open height at each location, so that red indicates a large open space and blue indicates a very shallow volume.



(a)



(b)

Figure 5.12: A high-resolution top-down histogram of the octree model of a single room: (a) reference photograph of room; (b) histogram visualization of model, with features labeled.

5.5 Removing Exterior Scanning Artifacts

For the remainder of this chapter, we expand on the techniques described in Chapter 4. These techniques are used to generate fully-detailed, complex 3D models of the observed environment. We exploit the 2.5D extruded floor plan geometry to improve the quality and capabilities of these dense models. By combining both modeling techniques, we can improve the accuracy of the 3D complex models and allow for further analysis of the scanned area. In this section, we identify areas of the 3D models that are scanning artifacts using volumetric analysis with the 2D floor plans. We then are able to remove these artifacts from the output 3D models.

Room labeling within a model provides an effective mechanism to prevent misrepresentation of poorly scanned areas. The mobile scanning system does not necessarily traverse every room and may only take superficial scans of room geometry while passing by a room's

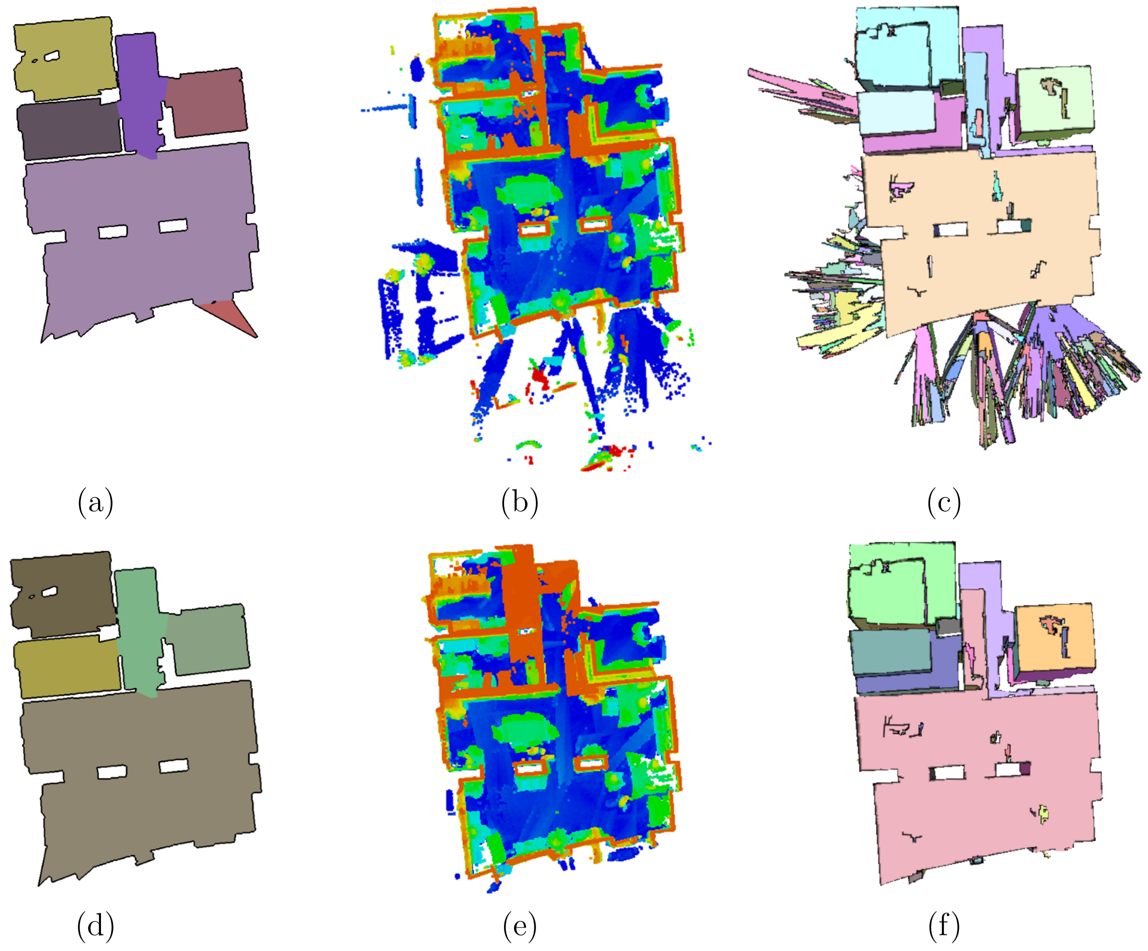


Figure 5.13: Using room labels to trim and improve models. Top-down view of: (a) Floor plan before room trimming; (b) point cloud before trimming, colored by height with ceiling points removed for viewing; (c) top-down view of surface carving before trimming; (d) floor plan after trimming; (e) point cloud after trimming; (f) surface carving after trimming.

open doorway. When a room is not entered, the model is unlikely to capture sufficient geometry and therefore it is necessary to remove this poorly scanned area from the model. An example of this process can be seen by comparing Figures 5.13a and 5.13d. If none of the triangles for a room within the floor plan are intersected by the scanner's path, we can infer the room is never entered. The room's triangles are then relabeled from interior to exterior, removing it from the floor plan. Figures 5.13a and 5.13d show an example floor plan before and after such trimming occurs, respectively. Note the removal of a sharp extrusion in the bottom-right corner, which was a partial scan through a window.

Similar refinement can be used to improve the 3D carving method described in Chapter 4. Due to the nature of voxel carving, laser scans that pass through a building window or open doorway can capture geometry outside of the desired scanned area. Since the outside volume is only observed from a few angles, the resulting carving produces undesirable artifacts, as shown in Figures 5.13b and 5.13c. Using the 2.5D extruded floor plan, we can automatically remove scans from the input point cloud that fall outside our desired area. Figures 5.13b and 5.13e show the corresponding point clouds before and after the result of this trimming, colored by height with the ceiling points removed. Any scans that pass through windows or doorways are removed. As a result, the surface carving of these point clouds discussed in Section 4.2, as shown in Figures 5.13c and 5.13f respectively, can be improved by reducing these undesirable artifacts. Similarly, the improved surface reconstruction scheme using an octree structure, as discussed in Section 4.3 can also be improved by preventing any nodes labeled *interior* by the carving but are well outside the floor plan geometry from being considered when exporting the mesh.

It is undesirable to simply remove any points that are outside of the floor plan area, since there may be geometry for interior walls or other features that are not labeled as *interior* in the floor plan that we still want to keep. Rather, we make the assumption that any points that are at least some distance d meters from any *interior* area of the floor plan are likely to be outside the scan environment. Figure 5.14 shows an example of this process. In Figure 5.14a, a top-down view of the 3D complex mesh is shown for a model of a residential apartment. The corresponding floor plan of the same area is shown in Figure 5.14b, in purple. We also show the extent of the "expanded" floor plan area, in red, which is intersected with the octree volume to determine the nodes to be ignored. Any octree nodes that are outside of the red area of the expanded floor plan are not considered as boundaries for meshing. The result of this modification is shown in Figure 5.14c, where the artifacts in the model caused by scanning exterior area through doors and windows are removed, leaving only the building geometry.

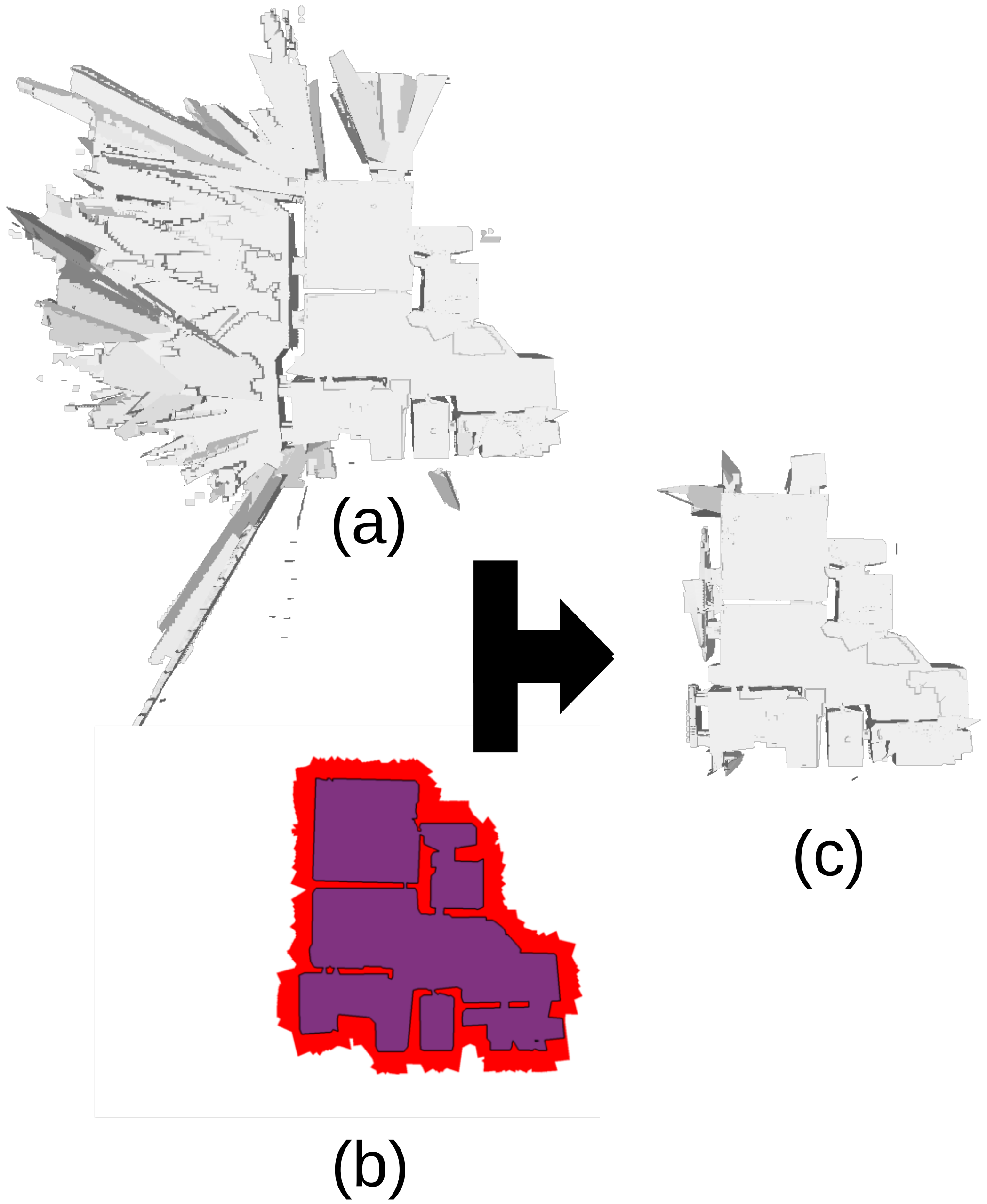


Figure 5.14: Example of using floor plan geometry to remove scan artifacts from 3D octree carving: (a) a top-down view of the original octree carving; (b) the corresponding floor plan geometry (in purple) with a 1 meter buffer of nearby area (in red); (c) the resulting octree carving after only considering interior area within the nearby area of the floor plan geometry.

5.6 Segmenting Furniture and Room Geometry

In this section, we propose a method to combine the octree-based 3D carving method described in Section 4.3 and the 2.5D extruded floor plan method described in Section 3.3 to perform object segmentation within the scanned environment. In both modeling techniques, each point in space has some probability of being *interior* or *exterior*. We define *interior* space to be empty or open area that range scans can pass through. We define *exterior* space to be solid material in the environment, including furniture and building structure.

Once we obtain a fully populated octree containing interior/exterior values, there is enough information to generate a surface reconstruction of the environment. Such volumetric representations of building environments can be fed into existing techniques to generate a mesh of the scanned area [5, 75, 93]. However, these meshes are limited in that they use the same method for meshing all parts of the environment. In this section, we discuss how we segment this representation to separate the volume of objects, such as furniture, from the rest of the building geometry and use different meshing techniques on each of these parts separately.

Our primary goal is to use this volumetric information to form two watertight meshes of the environment. The first mesh only represents the building geometry, including floors, walls, ceilings, windows, and doors. The second mesh represents the objects in the environment such as furniture, light fixtures, or other items. The block diagram of the method used for this segmentation is shown in Figure 5.15. We first generate two representations of the same environment. The populated octree represents a complex model of the volume, as shown by the red graphic in the upper-left of Figure 5.15. We then generate a simplified 2.5D model of the same volume, as shown by the blue graphic in the lower-left of Figure 5.15. As described in Chapter 3, this mesh does not represent any interior objects, but is aligned with the major surfaces of the building.

We then perform a set difference between these two volumetric models, keeping the volume that is labeled *exterior* by the octree and *interior* by the extruded floor plan. This subset of the volume represents the objects in the environment, and is shown as the green graphic in the upper-center of Figure 5.15. Similarly, we can denote the union of the *interior* space of both models to be the building geometry, as shown in the lower-center of Figure 5.15. Once the model is segmented into object and building geometry, we can generate meshes for each type. The object geometry is refined to enhance detail and is meshed uniformly to preserve its fine structure. The building geometry is split into planar surfaces and each surface is triangulated efficiently, preserving the sharp corners between floors, walls, and ceilings. We perform this planar meshing on the carved representation of the building geometry, rather than simply using the 2.5D extruded mesh, because it preserves building features such as windows and door-frames. These two meshes are combined to form the whole environment, shown by the graphic at the right-side of Figure 5.15. This approach has the added benefit of modeling hidden surfaces, such as the backs of furniture or areas of walls occluded by objects.

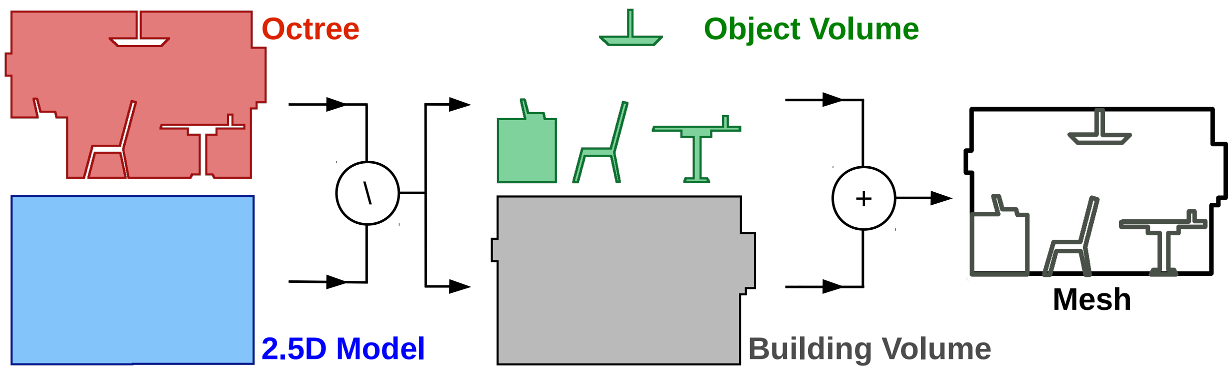


Figure 5.15: The scanned volume is meshed using two approaches that are combined to separate room geometry and object geometry. The complex geometry from the octree (upper left, in red) and the simple geometry from the 2.5D model (lower left, in blue) are both volumetric techniques. They are combined to extract the object volume (upper center, in green) and the building volume (lower center, in grey). These volumes are meshed separately and exported (right, in black).

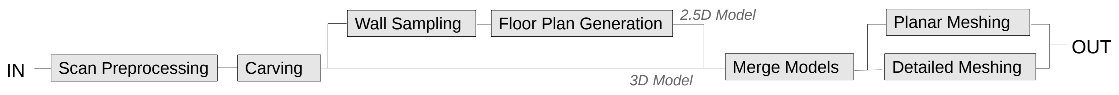


Figure 5.16: System flowchart of our approach. Scan Preprocessing is described in Section 4.1, Carving is detailed in Section 4.3, Wall Sampling and Floor Plan Generation are discussed in Section 5.3, Merge Models is delineated in Section 5.6.1, and Planar and Detailed Meshing are documented in Section 4.3.3.

5.6.1 Aligning Floor Plan Geometry to Octree Structure

In this section, we combine many ideas that are introduced throughout this dissertation in order to combine full 3D models and 2.5D extruded floor plans into a single unified model. The overview of data flow through this process is outlined in Figure 5.16. In Section 4.1, we discuss how we probabilistically model the input scans to produce volumetric estimates of *interior*/*exterior* occupancy from each scan point, which is labeled as "Scan Preprocessing" in Figure 5.16. In Section 4.3, we describe how these scans are efficiently combined to generate a unified occupancy estimate for the entire scan volume, as shown by the "Carving" block in Figure 5.16. These occupancy estimates are stored in an octree, representing a complex model of the environment. In Section 5.3, we detail how the octree is used to produce a simplified building model by first generating a 2D floor plan and then extruding the floor plan into a 2.5D mesh, shown in the "Wall Sampling" and "Floor Plan Generation" blocks of Figure 5.16.

In this section, we discuss how the complex model in the octree and the simplified model in the 2.5D extruded floor plan are merged to segment the volume delineating objects such as furniture in the environment, as shown by the "Merge Models" block in Figure 5.16.

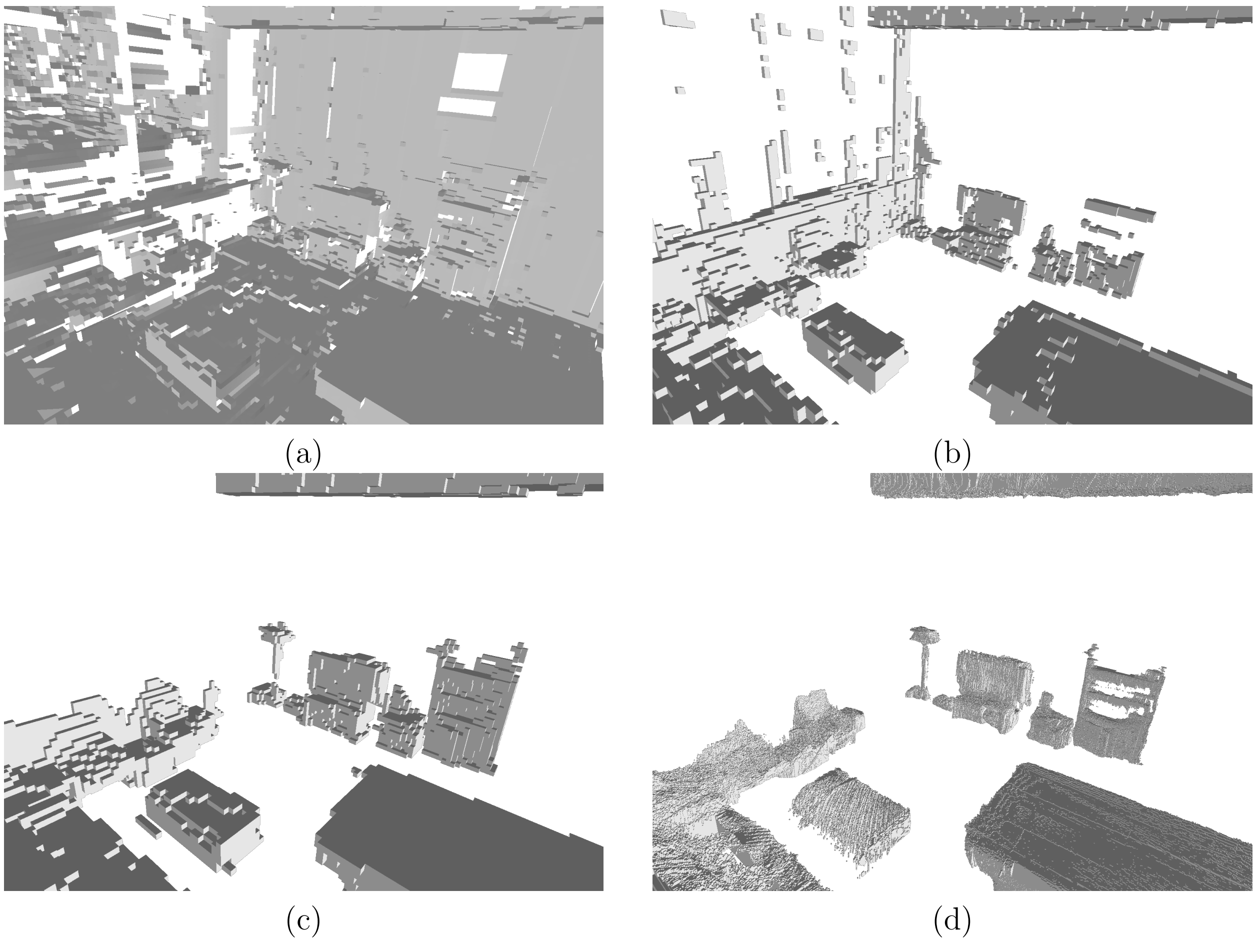


Figure 5.17: Example of aligning floor plan to segment objects: (a) original octree nodes, at max resolution of 6.25 cm; (b) segmented objects using unaligned floor plan; (c) segmented objects using aligned floor plan; (d) the segmented objects are recarved to a resolution of 0.8 cm.

This processes produces two separate watertight models: one for the building elements itself and one for the objects and furniture inside the building. These two types of models are meshed using the two separate techniques discussed in Section 4.3.3: dense uniform meshing on the octree structure and planar meshing by fitting large planar regions to the octree boundary. Each of these surface reconstruction techniques are geared to efficiently characterized their respective parts of the building. As shown in Figure 5.16, these output meshes are constructed separately, then combined into a unified exported mesh of the full building environment.

Both the extruded floor plan and the original octree are volumetric models of the environment, so we can classify the overlapping volumes into three categories. First, locations that are *exterior* in the octree yet *interior* in the extruded floor plan are objects or furniture in the environment. Locations labeled *exterior* by both models are considered part of the building structure. Lastly, all locations labeled *interior* by the octree are considered open space interior to the building, regardless of the extruded floor plan's labeling. Volume intersected by the boundary of the 2.5D floor plan is considered *exterior*, since these represent the primary building surfaces and not objects within the building. Using this segmentation, we can now consider the objects in the building separately from the 2.5D building structure. For instance, in Figure 5.17a, we see the original octree leaf nodes of a scanned environment. By performing a set difference of the octree volume from the volume of the 2.5D model of the environment, we can extract the furniture and other objects. Figure 5.17b shows the segmentation using an unaligned floor plan and Figure 5.17c shows the result with a fully aligned floor plan. The unaligned floor plan was generated directly from the raw point cloud of the scans whereas the aligned floor plan was generated with our method described in Section 5.3. With a properly segmented representation of the room's objects, we can recarve the nodes of the octree containing object geometry, since these locations tend to have finer detail than the rest of the model. Figure 5.17d shows an example of this recarving, which has been refined from the original resolution of 6.25 centimeters to a new resolution of less than a centimeter.

The effect of misalignment can be seen by comparing Figures 5.17b and 5.17c. In Figure 5.17b, the octree was segmented using a floor plan generated directly from the point cloud and not the octree. As such, parts of the back wall and window are mislabeled as objects and kept in the output. In Figure 5.17c, the octree was segmented using a floor plan generated from the octree itself as described in this section. The back wall is no longer mislabeled and only the actual furniture in the environment are segmented as objects.

Once we have fully merged these models, each node of the octree is labeled as either object geometry or room geometry. In addition to refining the resolution of object nodes, we can use this labeling to adjust how we generate a mesh for each portion of the environment.

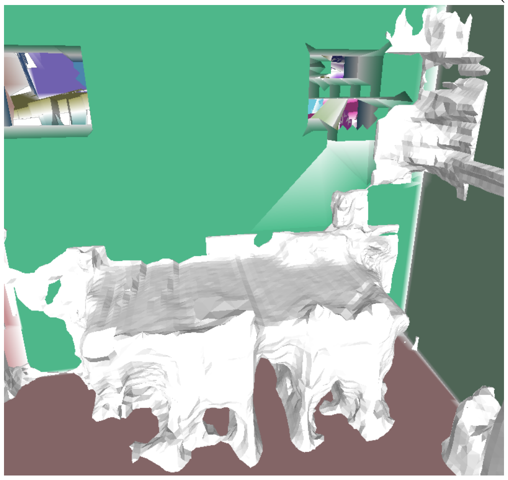
5.6.2 Meshing Building and Object Volumes

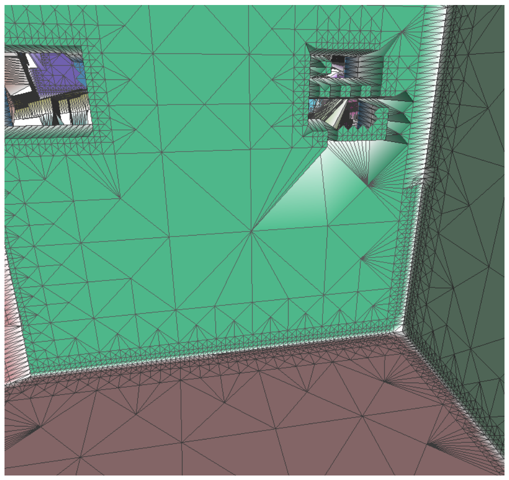
After segmenting the octree geometry into objects and rooms, we can mesh each separately. Objects such as furniture, light fixtures, doors, etc. tend to have higher detail than the room-level geometry. The room-level geometry tends to be composed of large, planar surfaces. We use a dense meshing technique to represent the object geometry, which preserves detail and curves in the geometry. For the room-level geometry, we identify planar regions and mesh each plane with large triangles. Figure 5.18 shows an example of applying both of these methods to a given model. Figure 5.18a shows a photograph of the scanned area, a kitchen table, and Figure 5.18b shows the final output of all meshing approaches combined.

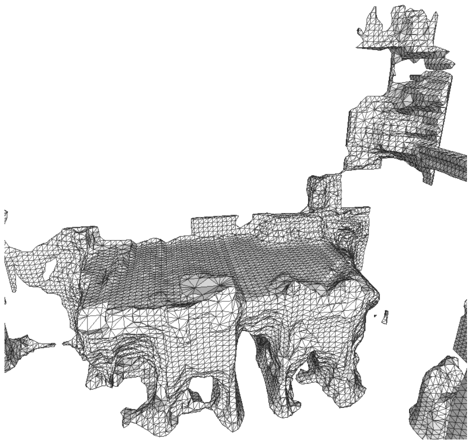
To mesh building geometry, we first partition the boundary faces of the octree into planar regions, as discussed in Section 4.3.3. We perform planar meshing on the octree elements to represent building features rather than simply using the 2.5D extruded mesh generated from the floor plan because the latter does not capture sufficient detail of the environment. Features that do not follow the 2.5D assumption, such as protruded windows or door-frames, can not be captured by the extruded floor plan mesh. As shown in Figures 5.18c and 5.19d, the planar mesh of the building surface still provide geometry for features such as window recesses.

We opt to use dense meshing on the objects and furniture found in the environment. This dense meshing technique is fully described in Section 4.3.3, and based off of dual contouring [18]. The advantage to this method is that it preserves all detail from the octree when generating the output surface, and takes full advantage of the probabilistic nature of

(a)


(b)


(c)


(d)

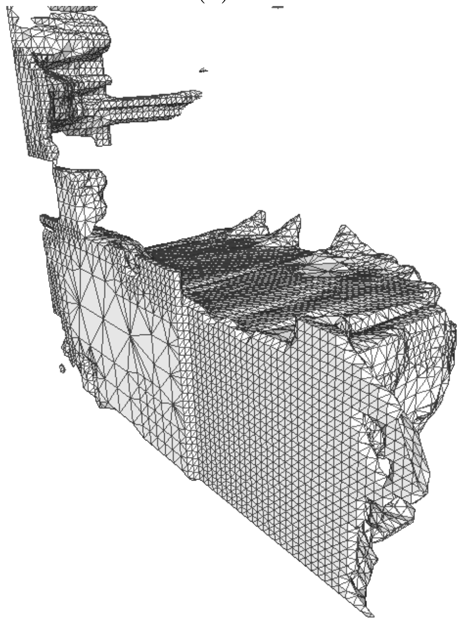

(e)

Figure 5.18: Example meshing output of residential area: (a) photo of area; (b) all reconstruction geometry; (c) geometry of room surfaces only, colored by planar region; (d) geometry of objects only; (e) geometry of objects from behind, showing watertightness.

the octree, allowing for sub-voxel accuracy in the mesh. Figure 5.18d shows the object mesh in isolation for an example model.

The disadvantage of this method is the output mesh size. Even though there is some adaptive meshing, due to the adaptive nature of the octree, by far most of the output is of uniform element size, resulting in a large number of elements. We make the assumption that objects and furniture in the building represent only a fraction of the total surface area observed in the model, which enables us to generate dense meshes for these objects of fine detail, while still leaving the remainder of the model to be meshed more economically.

An important aspect of meshing these two segments separately is to ensure watertightness of building and object models. The surfaces of walls hidden behind any occluding objects are still meshed, even though they are never directly scanned. This effect can be seen in Figure 5.18c. Similarly, the hidden surfaces of objects are also fully meshed. Figure 5.18e shows the rear surface of the table, which was filled in based on the wall location.

5.7 Results

Our goal is to generate models of large scanned environments and still preserve fine detail of objects in those environments. In this section, we discuss the advantages of our method both with qualitative examples and quantitative results. All models shown were generated on an Intel Xeon 3.10 GHz processor.

We aim to improve on existing methods by combining two fundamentally different surface reconstruction techniques for building environments. We combine the extruded floor plan model and the 3D octree carving model to segment volumetric representations of the interior objects in the environment, such as furniture or light fixtures, from the permanent surfaces of the building such as floors, walls, and ceilings. These steps allow us to generate accurate, watertight models of objects in the building distinct from the building model itself, as demonstrated in Figure 5.19. We first produce a volumetric representation of the entire space, stored in an octree, as shown in Figure 5.19b. We use this representation to produce a rich model of the environment, as shown in Figure 5.19c. The objects of the environment, shown in white, can be separated from the building structure, as shown in Figure 5.19d. We use different meshing techniques for the objects and the building itself to ensure the best representation of each type of surface. The result is a rich model of the environment that represents a whole building based on level, room, or individual objects. The objects are meshed densely with 63,500 triangles and the room is meshed in a planar fashion with 185,000 triangles. This gives a net savings of 143,000 triangles than if the whole model is meshed densely.

In Figure 5.20, we show results for a scan of several rooms in a hospital operating area. This model contains four rooms, covering a total of 1,937 square feet, and was scanned using our hardware system in 2 minutes 47 seconds. Processing this model took a total of 6 hours and 8 minutes. The room shown in Figure 5.20a is a hybrid operating room, which contains several medical scanners affixed to the ceiling. As shown in Figure 5.20b, our approach segments the geometry of the scanners from the rest of the building and generates a mesh for the entire environment. Figure 5.20c shows the geometry of the operating table and medical equipment only. This object would be difficult to model with techniques that

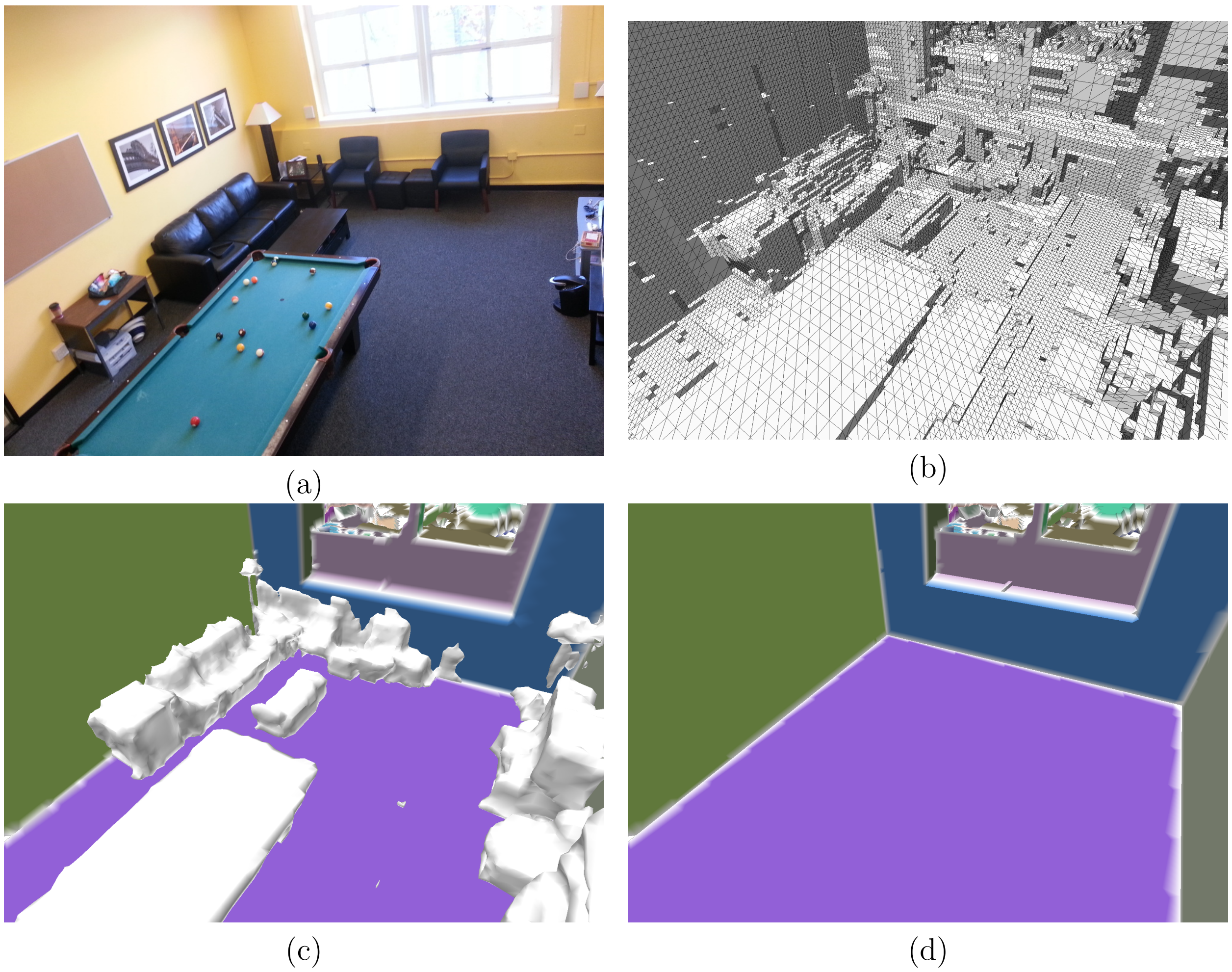


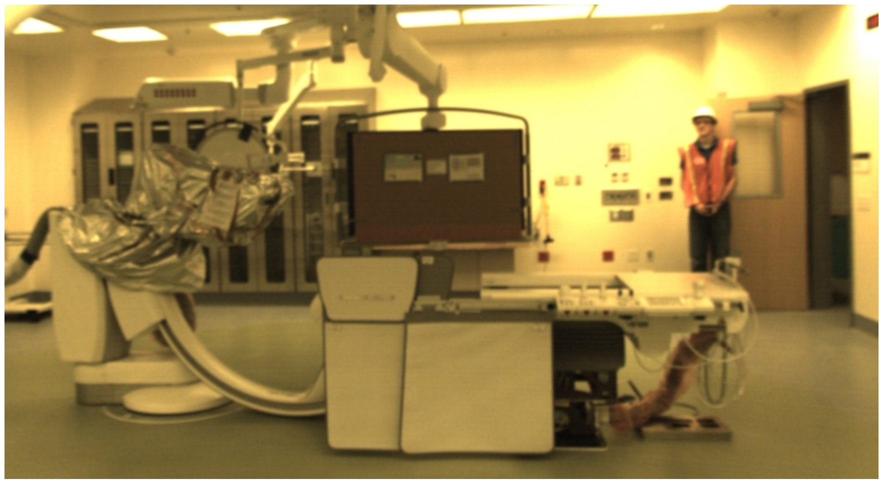
Figure 5.19: An area modeled by our technique: (a) a photo of the room; (b) the volumetric boundary of room; (c) final mesh with room and objects modeled; (d) final mesh of room only, colored by planar region.

semantically classify geometry to form a mesh, since it is unlikely that shape libraries would have many examples of such an usual device. Since our technique does not need to classify the shape, we can still generate an accurate representation of its geometry.

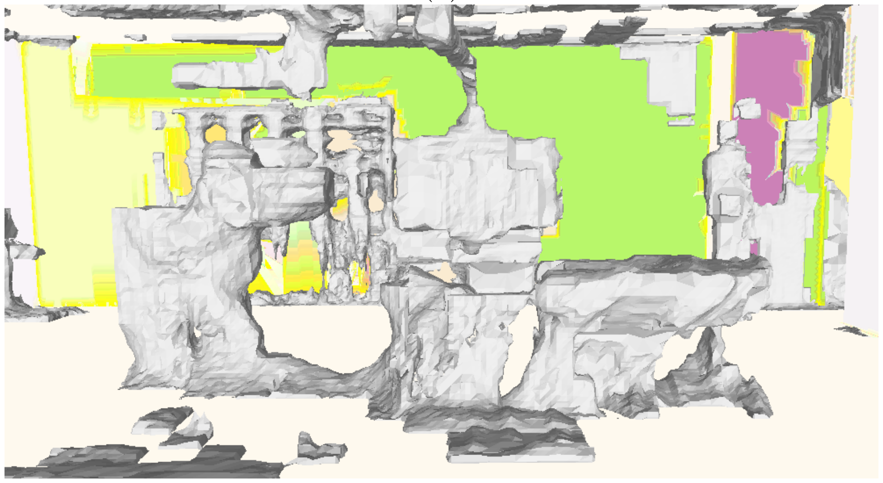
We scanned a large office environment to show the scalability of our approach, shown in Figures 5.21, 5.22, and 5.23. This model contains a complete scan of 41 rooms as well as the surrounding hallways, totaling 13,048 square feet. The data acquisition was completed in 17 minutes and processed in 48 processor-hours. The building geometry is represented by 4 million triangles and the objects are meshed with a total of 13 million triangles. Even though the building surfaces compose most of the environment, they represent the minority of triangles since those surfaces can be meshed efficiently as planar regions. By comparison, if the entire model had been meshed using dual contouring, then it would have been composed of 21.5 million triangles.

Figure 5.21 shows the automatically generated floor plan of the environment. We show examples of the output 3D model from multiple locations, marked as “1” and “2” on the floor plan. Figures 5.22a-5.22d show the reception desk located at point "1" in the floor plan shown in Figure 5.21. The desk, computers, and sink in this area are correctly segmented as objects and the primary building surfaces are meshed in a planar fashion. The building surfaces can be seen in Figure 5.22c and the objects in Figure 5.22d. We also show an example resulting mesh from inside one of the rooms, in Figures 5.23f-5.23i. These figures represent location "2", as marked on Figure 5.21. Note that the door frame, desks, and other objects in this room are correctly segmented from the model. Most of the wall fixtures have shapes that are non-planar, so they are more accurately modeled using a dense meshing scheme as discussed in Section 4.3.3 rather than the corresponding planar meshing scheme. The primary building surfaces are still meshed in a planar fashion, resulting in a model that preserves the sharp edges and flat surfaces of the floors, walls, and ceilings.

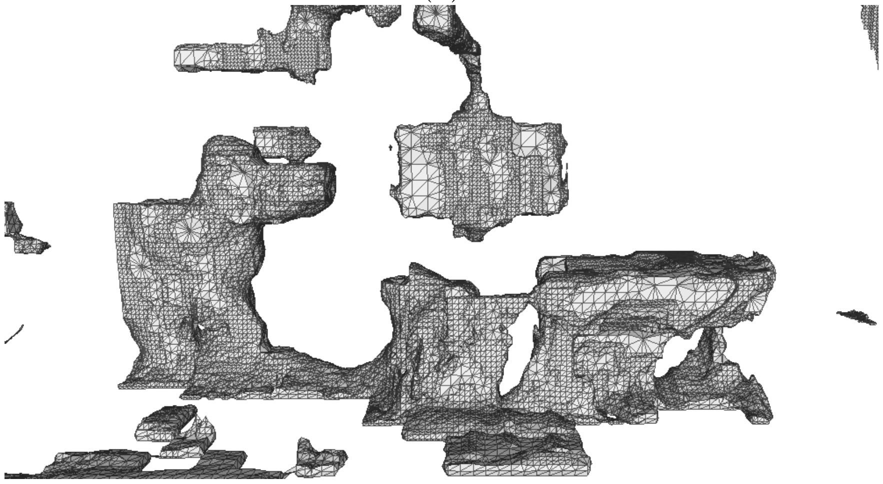
The approach proposed in this chapter does have some limitations. Since the object segmentation procedure described in Section 5.6 uses volumetric intersections with an extruded 2.5D model based on a floor plan, it relies on assumptions this 2.5D model makes about the scanned environment. This approach assumes each room has fixed floor and ceiling heights. If a room's ceiling is not horizontal, then it is approximated with a horizontal surface. Figure 5.24 shows a few additional limitations. This figure shows a set of boxes on top of a raised platform next to a bookcase. The raised platform is identified as a separate object, even though it is part of the building structure, since it is at a different elevation than the rest of the floor in this room. Additionally, this figure shows a floor-to-ceiling bookcase. Since the position of fitted walls is found by looking for vertical surfaces, it is difficult to accurately gauge the depth of the shelves, especially when they are filled. As a result, the wall is not positioned correctly, and only part of the bookcase is segmented as an object. It is important to note, however, that all these limitations are highly localized. Other parts of the model, such as the set of boxes or the coat-rack, are still meshed correctly in the presence of these issues.



(a)



(b)



(c)

Figure 5.20: Example scan of equipment in hospital's hybrid operating room: (a) picture of scanned area; (b) model of area; (c) object model triangulation of operating table and equipment.

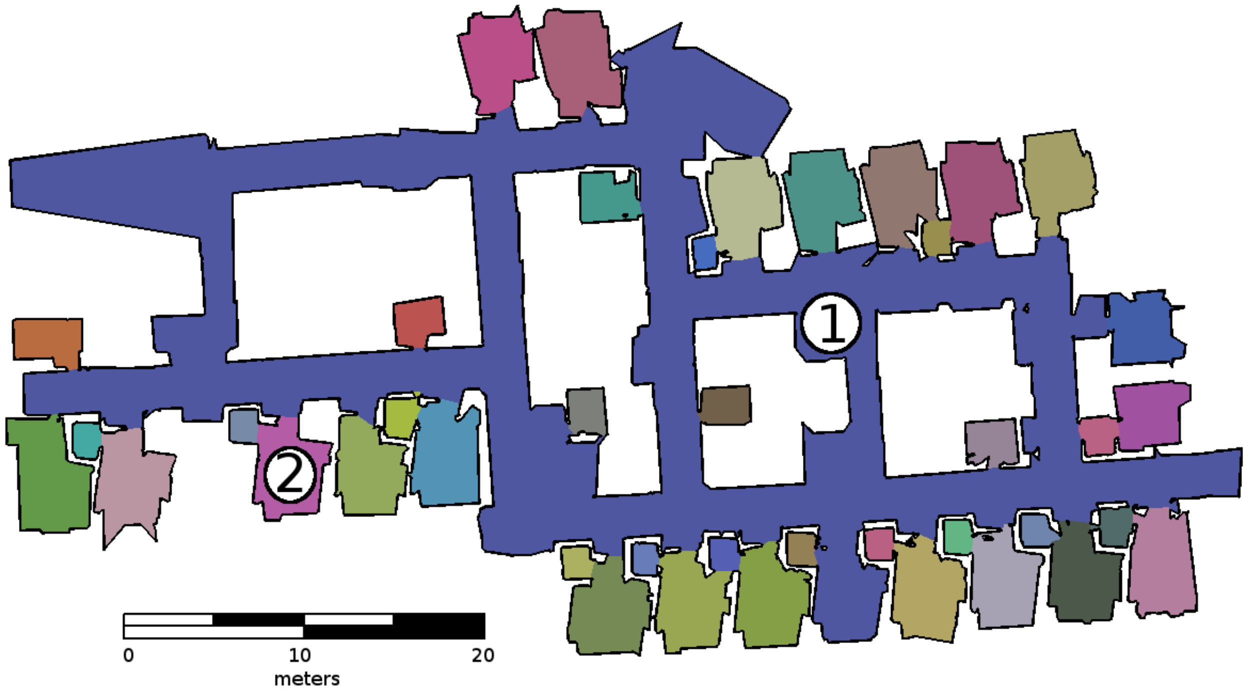
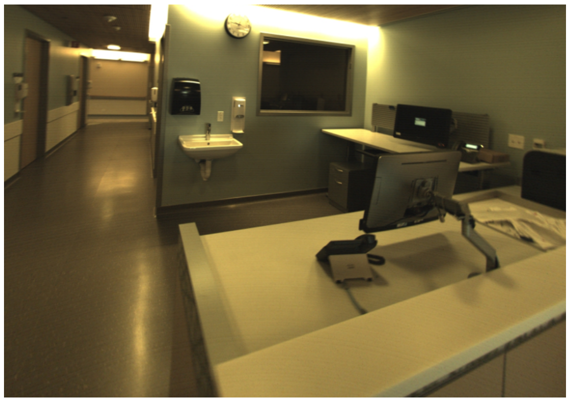
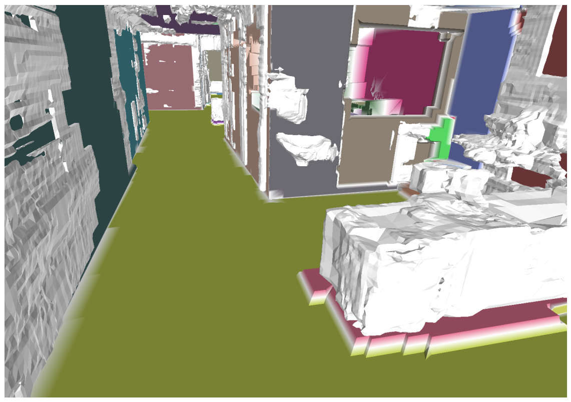


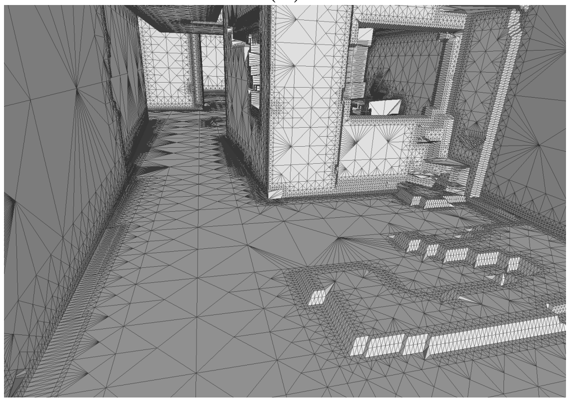
Figure 5.21: Example model of large office environment, showing the generated floor plan of scanned environment, colored by room.



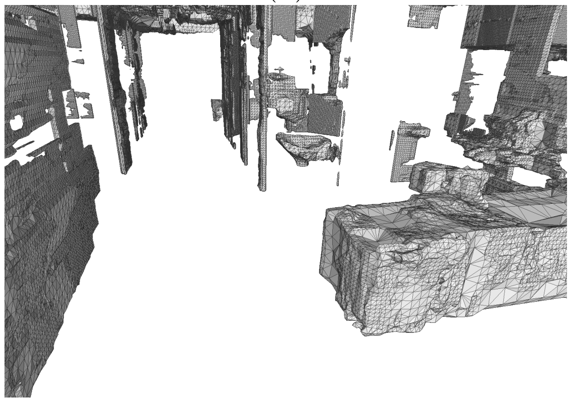
(a)



(b)

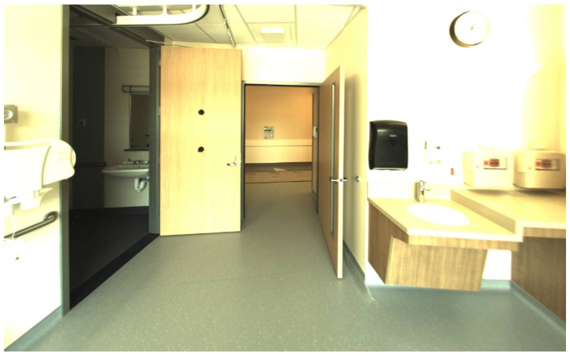


(c)

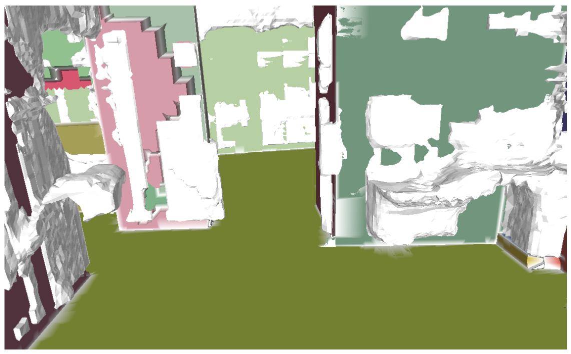


(d)

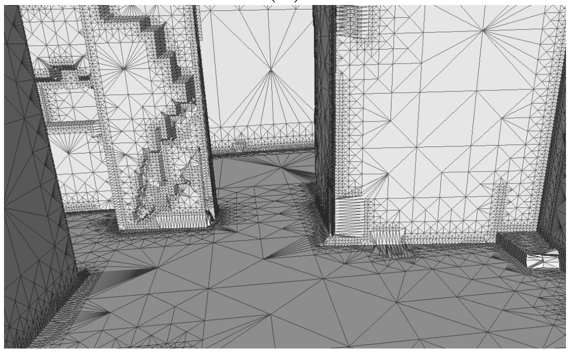
Figure 5.22: Example model of large office environment, showing close-up of location "1" from Figure 5.21: (a) Photo of area "1" in model; (b) full mesh of area "1"; (c) triangulation of room geometry in area "1"; (d) triangulation of object geometry in area "1".



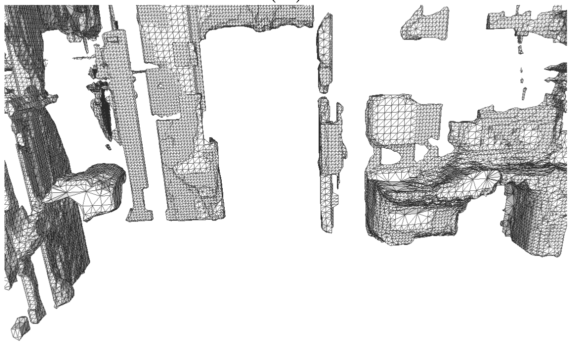
(a)



(b)

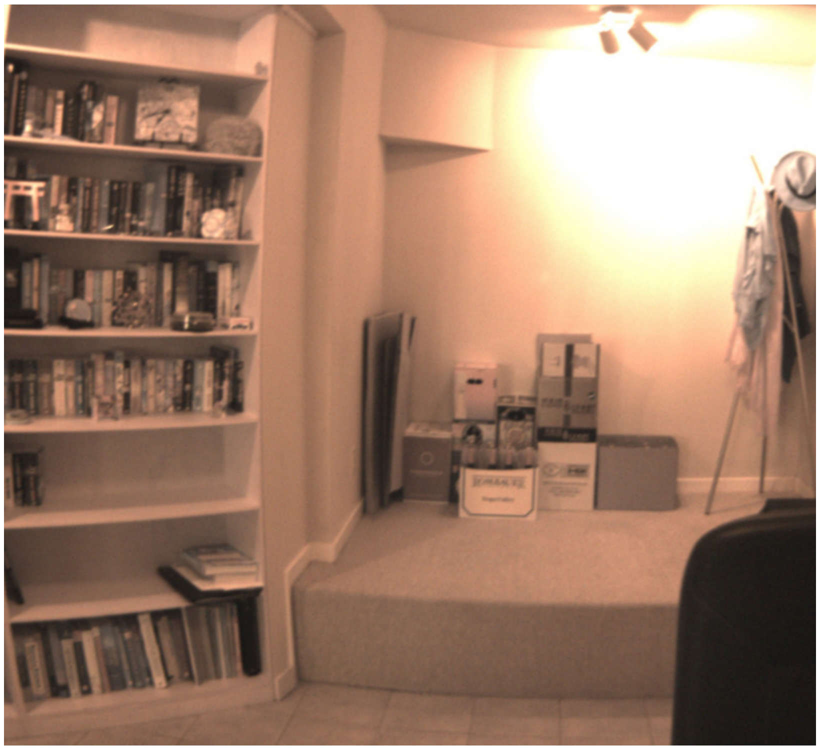


(c)

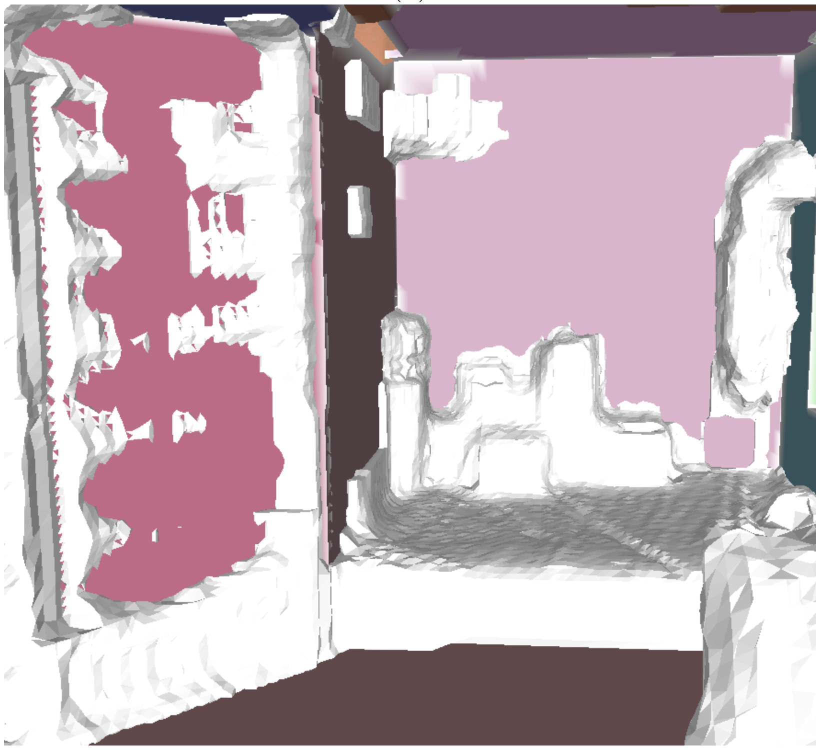


(d)

Figure 5.23: Example model of large office environment, showing close-up of location "2" from Figure 5.21: (a) photo of area "2" in model; (b) full mesh of area "2"; (c) triangulation of room geometry in area "2"; (d) triangulation of object geometry in area "2".



(a)



(b)

Figure 5.24: Example mesh of bookcase and boxes: (a) photograph of scanned area; (b) generated mesh, showing building geometry in color and object geometry in white.

Chapter 6

Applications of Building Models

In this dissertation we have proposed three main categories of models: 2D floor plans, 2.5D extruded models that produce highly simplified geometry, and fully complex 3D models. In this chapter, we discuss applications for each of these techniques that go beyond visualization of the building geometry. Each of the discussed methods uses the generated mesh of the building environment to perform some analysis or computation. The complexity of the mesh plays an important role in how effective these computations can be, and it is often important to have all categories of models for each analyzed environment.

In Section 6.1, we discuss how these models can be used to aid building energy simulations, which help for energy audits for indoor environments. Understanding the geometry of the building environment at many resolutions facilitates detection of important features in the building environment. In Section 6.2, we discuss how building models can be used to facilitate indoor navigation. By producing a fully scan of a building environment, we can generate a database of fingerprints, uniquely identifying each location in the environment. Then, at a later time, a user can identify their location in the environment by performing a look-up in this database.

6.1 Building Energy Simulation

Building Information Models (BIMs) have seen considerable use in the Construction, Engineering, and Architecture communities [41]. It is often important to able to compare the architectural blue-prints for a building to the as-built layout of that building. Such comparisons allow for architects to ensure the structure is being built to specification, or to account for any deviations that may take place. Additionally, renovations to a building environment can make the original architectural plans out-of-date for such concerns. An example of such a comparison is shown in Figure 6.1. An architect's designs are shown as a 3D as-designed model, with the scanned as-built 3D model overlayed in green. This model was generated using the carving method described in Chapter 4. Since the scanned model was able to successfully capture the pipes shown in this building, a construction crew can ensure that their placement of this equipment is in the correct location.

Similarly, recent trends have been to perform as-built surveys of building environments when performing energy audits. The large-scale of design for a building can often play

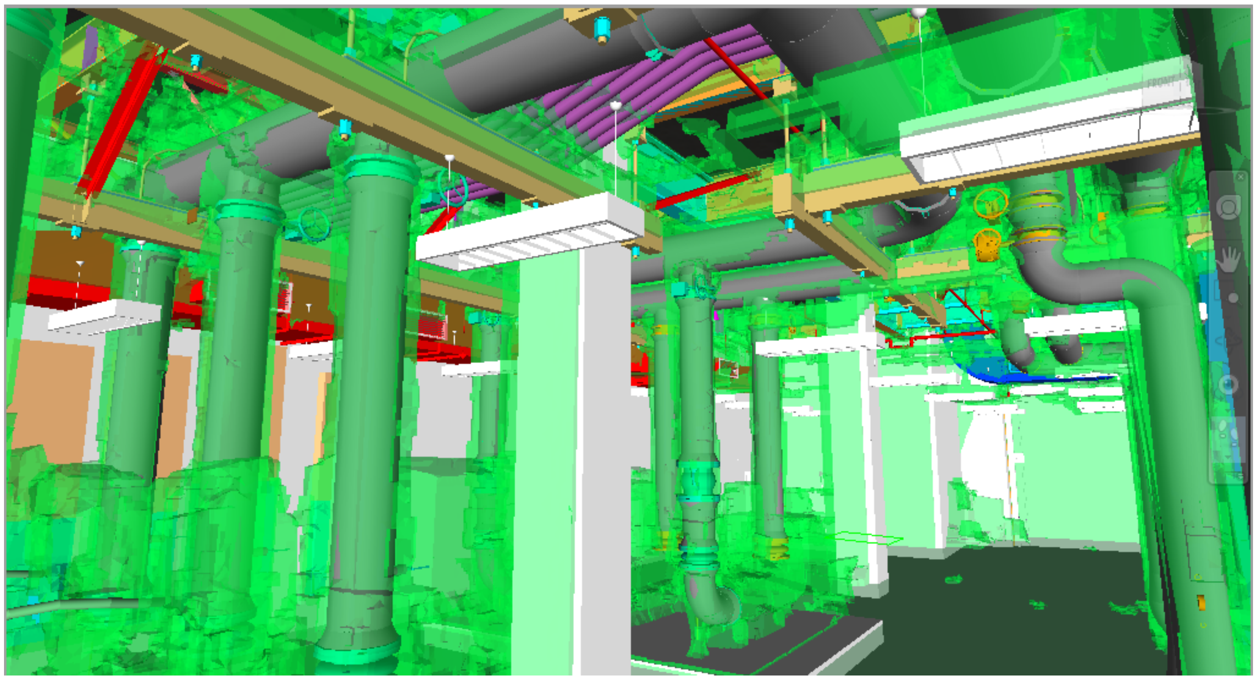


Figure 6.1: An example of overlaying scan data to an architect's original building designs. The scanned as-built 3D model is shown in green, overlayed onto the as-designed 3D Building Information Model (BIM).

important roles in its energy efficiency, but such analysis also includes many building features that would not be on the original architectural plans, such as the types of bulbs used for lighting or the window material used. As-built analysis is required to produce the most accurate energy audit for a building environment [100].

Recently, tools have been developed to perform an automated simulation of the thermal flow of a building environment using imported building geometry [6]. While these tools were intended to require human users to manually generate the geometry, our modeling system can be employed to produce models of the quality required to perform thermal simulations. Since these simulations are performed across the entire building environment, highly simplified geometry is required. Models of buildings as discussed in Chapter 4 contain millions of triangles, and are intractably large for such simulation engines. This need for simplified 3D geometry of the building environment is part of the motivation for our work on extruded floor plans discussed in Section 3.3.

Part of the analysis required to produce energy simulations with out automatically generated models is to detect building features relevant to energy usage. Such features include windows, light fixtures, equipment plug-loads, and occupancy. Even though the simplified 2.5D extruded floor plan model is used to form the geometry for these simulations, all scan components are necessary to perform the processing and analysis to detect these features. Figure 6.2 shows an example model used for energy simulation, generated by our backpack scanning system. The windows shown in this model are automatically generated using the floor plan model from Section 3.3 and the colored point clouds from Section 2.4 [101]. Plugload detection is performed by classifying objects via depth-maps produced using the 3D

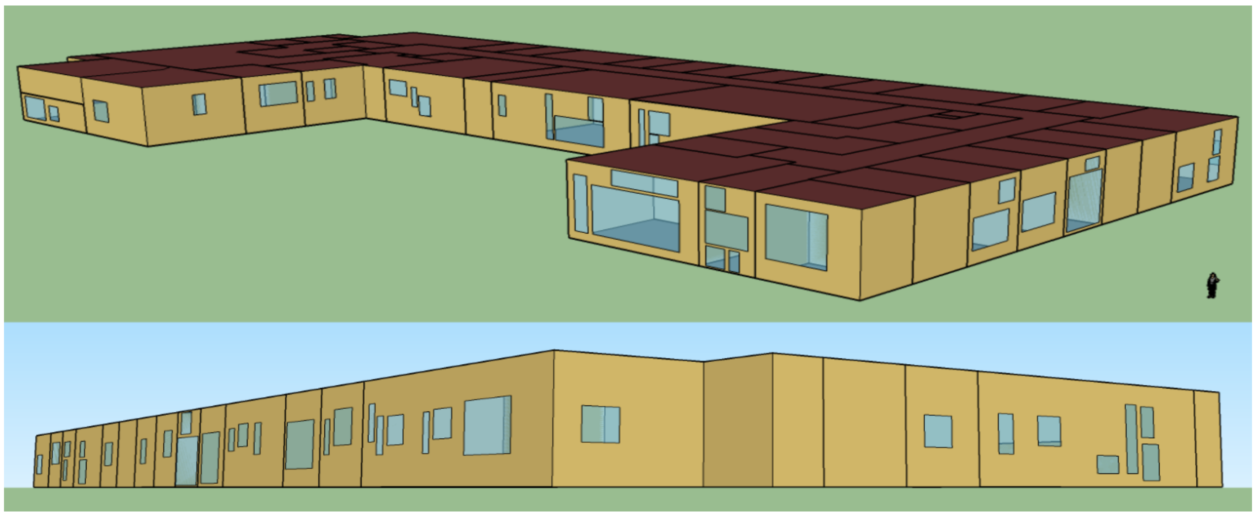


Figure 6.2: Example building model used for energy audit simulation. This model features automatically-detected windows based on the input scans from our backpack system.

complex models discussed Section 4.2. Detected computers are used to estimate the occupancy counts, which are split into zones based on the partitioned rooms in the environment as shown in floor plan generation in Section 3.3.

The result is a full building simulation based on automatically generated models of the building environment, using scans from our backpack-mounted system. Such simulations can be expanded even further using the additional sensors on the scanning system. Infrared cameras can be used to detect faulty HVAC equipment or leaks in the insulation [84], barometers can be used to map pressure variance from room-to-room within the building, and dataset timestamps can be used to determine local weather conditions during the model acquisition.

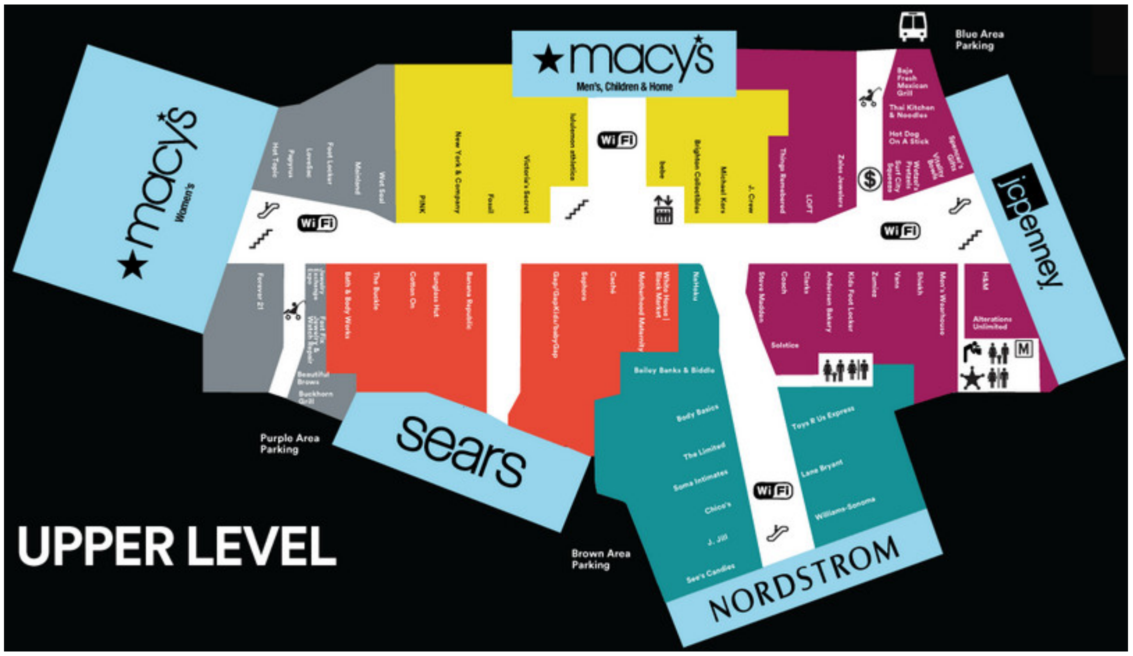
6.2 Indoor Navigation

Another application for building models is for indoor navigation. Since indoor environments are often GPS-denied, it is difficult for a user to identify their position and orientation while indoors. A popular technique is to perform a building scan using a modeling system, such as our backpack system. Once the building model is processed, then a user can traverse the building environment at a later date, using much simpler equipment such as a cell-phone. This technology gives navigation ability to users in populated indoor environments without the need for modifications to the building environment or requiring carrying bulky scanning equipment.

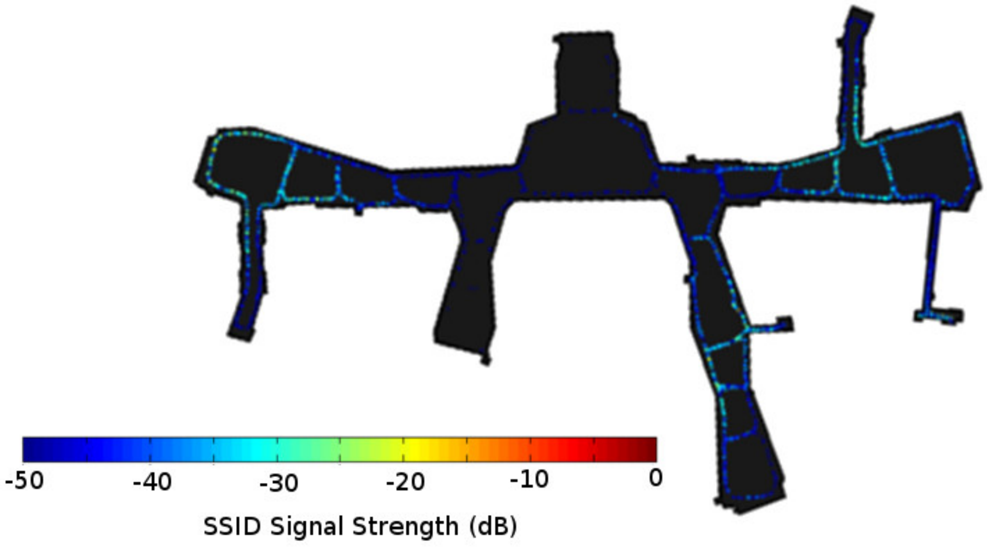
Several technologies have been utilized to produce a unique fingerprint for each location in the building environment. A popular technique is to produce fingerprints from camera imagery [3]. This method records image features taken at each location during the data acquisition. Later, a user can return the that location in the environment and take a new picture. By comparing image features, a match can be found and a location and orientation returned to the user. A detailed 3D model of the environment, as discussed in Section 4.2, must be produced to generate normal-maps and depth-maps for each camera image taken in the original dataset. These geometry attributes are used to compute the difference between the pose of the original database image and the new query image, allowing for accurate positioning in the environment.

A more advanced technique combines imagery localization with readings from Wi-Fi antennas [77]. This method tracks a users continuous movement by first performing a rough localization using fingerprinting of observed Wi-Fi signals, then tracks their local movements using imagery. This method requires an accurate floor plan of the environment to perform such tracking, as generated by Section 3.3, to ensure that estimates of the user position does not vary outside of the valid interior area of the building.

Individual rooms in a building are likely to have a unique set of signal strengths for all observed Wi-Fi SSIDs, which enables fingerprinting of the environment. However, these Wi-Fi signals also allow for rich analysis of building structures. By observing and mapping Wi-Fi signal strength, we can determine the likely location of wireless access points. Figure 6.3 shows an example of this process. Figure 6.3a shows the directory listing of the Stoneridge Mall, in Pleasanton, CA [102], which includes the rough locations of their Wi-Fi hotspots. By performing a scan of the public areas of the environment, we can visualize a floor plan of this area, as shown in Figure 6.3b. For each position traversed during scanning, we can collect the signal strength of the mall's public wireless network. As observed in this figure, the maximum strength comes from areas in the mall that match the hotspot locations listed, showing that we can track wireless access points. These signals are broadcast at frequencies that do not easily transmit through thick building material, such as brick or concrete, allowing some analysis of structural materials based on areas with low signal strength.



(a)



(b)

Figure 6.3: Example of how Wi-Fi signals can be analyzed by our scanning system: (a) the listed directory of a shopping mall [102]; (b) observed Wi-Fi signal strength. Observed Wi-Fi hotspots match those listed in the mall directory.

Chapter 7

Conclusions and Future Work

In this dissertation, we have presented a variety of robust methods of surface reconstruction designed for indoor building environments. Our methods can take input scans with high noise typically observed in mobile ambulatory acquisition systems. We showed how our backpack-mounted hardware system provides a rapid and efficient mechanism for generating models of large building environments with minimal acquisition time.

As indoor environments are often GPS-denied, our system computes localization of the path the human operator traversed. The localization approach for the backpack scanning system allows for scalable acquisitions of tens of thousands of square feet with only 10–15 centimeters of positional error globally [53,54]. The modeling techniques we describe in this dissertation are formulated to be robust to this level of noise in the input localization, and can provide accurate geometry of the building environment using our backpack system.

We discuss three modes of indoor building modeling: 2D floor plans, simplified 3D models, and fully-detailed complex 3D models. Each of these modeling approaches produces a watertight representation of the scanned interior environment and is useful for many varied applications in construction, architecture, engineering analysis, virtual/augmented reality, and indoor navigation. We show how we have advanced our techniques for each of these model types to allow scalable acquisition of large building environments. We also show the application of each of these techniques on a wide array of real-world datasets, encompassing building types such as academic buildings, office environments, hospitals, residential areas, shopping malls, construction sites, and hotels.

Additionally, we show that these techniques are not only useful independently, but can be combined to further improve accuracy and functionality. Full 3D analysis can improve the accuracy of the geometry generated for 2D floor plans or enable new methods of 2D visualization. Floor plans can be employed to improve the aesthetics and accuracy of fully 3D models, as well as enable segmentation of the watertight geometry for furniture and other objects in the building environment. Such segmentation can help improve the level of detail for how these objects are represented in the exported model. This segmentation might also be used in the future for more accurate classification of these objects.

Once objects are separated from the building geometry, the building itself can be modeled more accurately, producing estimates for positions of surfaces that would otherwise be occluded by clutter in the environment. Our approach can partition building geometry into levels, rooms, and individual objects. This degree of segmentation is an important step to automatically generating richly defined Building Information Models that represent all aspects of the environment.

The methods detailed in this dissertation can be expanded to improve results even further. First, the incorporation of sensors in addition to LiDAR can add increased detail to model processing. By tracking visual features in camera imagery, for example, we could ensure that such features are aligned and well-modeled in the exported meshes. Since our carving process uses probabilistic line-of-sight computation to identify volume as interior or exterior, triangulated feature points from camera imagery could be incorporated seamlessly into the geometry processing pipeline. Such features could be aligned with the laser scan information by detecting corners in the scans. The advantage of such camera-laser alignment would be improved texture-mapping processes. Since we are able to separate furniture and building geometry in a volumetric and watertight fashion, texture-mapping could be adapted to allow for in-painting on the parts of building surfaces that were unobserved due to occlusions by furniture. Similarly, the backs of furniture and objects could be in-painted using the texture features on the front sides. Separately, our segmented furniture models could be used as the input for object recognition algorithms, allowing for semantic understanding in large environments in a scalable fashion. The classification process would not need to analyze every location, but just the geometry segmented as objects. Such a scheme would be useful for comparing the as-built automatically generated model of a building to the original architectural plans. Discrepancies could be identified and the architectural plans could be updated based on the observed results.

These techniques are applicable to a variety of existing research topics, including automated energy auditing and indoor navigation. By producing richer models of building environments that go beyond simplified representations of observed geometry, we can provide avenues for simulation and analysis that will improve how we use these spaces and how they are constructed in the future.

Bibliography

[1] F. Bosche, “Automated recognition of 3d cad model objects in laser scans and calculation of as-built dimensions for dimensional compliance control in construction,” *Advanced Engineering Informatics*, vol. 24, pp. 107–118, 2010.

[2] X. Xiong, A. Adan, B. Akinci, and D. Huber, “Automatic creation of semantically rich 3d building models from laser scanner data,” *Automation in Construction*, vol. 31, pp. 325–337, 2013.

[3] J. Z. Liang, N. Corso, E. Turner, and A. Zakhor, “Reduced-complexity data acquisition system for image based localization in indoor environments,” *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, October 2013.

[4] Y. Bok, Y. Jeong, and D.-G. Choi, “Capturing village-level heritages with a hand-held camera-laser fusion sensor,” *International Journal of Computer Vision*, vol. 94, no. 1, pp. 36–53, October 2010.

[5] C. Holenstein, R. Zlot, and M. Bosse, “Watertight surface reconstruction of caves from 3d laser data,” *IEEE/RSJ International Conference on Intelligent Robots and Systems*, September 2011.

[6] B. B. Crawley, L. K. Kawrie, C. O. Pedersen, and F. C. Winkelmann, “Energyplus: Energy simulation program,” *American Society of Heating, Refrigerating, and Air-Conditioning Engineers (ASHRAE) Annual Conference*, vol. 42, no. 4, pp. 49–56, April 2000.

[7] E. Turner, P. Cheng, and A. Zakhor, “Fast, automated, scalable generation of textured 3d models of indoor environments,” *Journal of Special Topics in Signal Processing*, vol. 9, no. 3, pp. 1–13, 2014.

[8] N. Amenta, S. Choi, and R. K. Kolluri, “The power crust,” *Proceedings of the Sixth Symposium on Solid Modeling*, pp. 249–260, 2001.

[9] J. A. Bærentzen, “Octree-based volume sculpting,” *Vis98-IEEE Visualization*, 1998.

[10] H. Hoppe, “Progressive meshes,” *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pp. 99–108, 1996.

[11] M. Kazhdan, M. Bolitho, and H. Hoppe, “Poisson surface reconstruction,” *Eurographics Symposium on Geometry Processing*, 2006.

[12] R. Kolluri, J. R. Shewchuk, and J. F. O'Brien, "Spectral surface reconstruction from noisy point clouds," *Symposium on Goemtry Processing*, pp. 11–21, July 2004.

[13] P. Cheng, M. Anderson, S. He, and A. Zakhor, "Texture mapping 3d planar models of indoor environments with noisy camera poses," *SPIE Electronic Imaging Conference, Computational Imaging XII*, no. 9020, February 2014.

[14] B. Mederos, N. Amenta, L. Velho, and L. H. de Figueiredo, "Surface reconstruction from noisy point clouds," *Eurographics Symposium on Geometry Processing*, 2005.

[15] K. Denker, B. Lehner, and G. Umlauf, "Real-time triangulation of point streams," *Engineering with Computers*, vol. 27, pp. 67–80, 2011.

[16] M. Isenburg, Y. Liu, J. Shewchuk, and J. Snoeyink, "Streaming computation of delaunay triangulations," *ACM Transactions on Graphics (TOG)*, pp. 1049–1056, July 2006.

[17] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 163–169, July 1987.

[18] T. Ju, F. Losasso, S. Schaefer, and J. Warren, "Dual contouring of hermite data," *ACM Transactions on Graphics (TOG)*, 2002.

[19] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pp. 303–312, 1996.

[20] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," *ACM Transactions on Graphics (TOG)*, pp. 71–78, 1992.

[21] M. Bolitho, M. Kazhdan, R. Burns, and H. Hoppe, "Multilevel streaming for out-of-core surface reconstruction," *Symposium on Geometry Processing (SGP)*, pp. 69–78, 2007.

[22] ——, "Parallel poisson surface reconstruction," *International Symposium on Advances in Visual Computing (ISVC)*, pp. 678–689, 2009.

[23] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," *ACM Transactions on Graphics (TOG)*, pp. 209–216, 1997.

[24] F. Labelle and J. R. Shewchuk, "Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles," *ACM Transactions on Graphics*, vol. 26, no. 3, p. 57, July 2007.

[25] Y. Zhang and C. Bajaj, "Adaptive and quality quadrilateral/hexahedral meshing from volumetric data," *Computer Methods in Applied Mechanics and Engineering*, vol. 195, pp. 942–960, 2006.

[26] A. Sharf, D. A. Alcantara, T. Lewiner, C. Greif, A. Sheffer, N. Amenta, and D. Cohen-Or, “Space-time surface reconstruction using incompressible flow,” *ACM Transactions on Graphics*, vol. 27, no. 110, p. 110, December 2008.

[27] G. Windreich, N. Kiryati, and G. Lohmann, “Voxel-based surface area estimation: From theory to practice,” *Pattern Recognition*, vol. 26, pp. 2531–2541, 2003.

[28] Y.-K. Yang, J. Lee, S.-K. Kim, and C.-H. Kim, “Adaptive space carving with texture mapping,” *International Conference on Computational Science and Its Applications (ICCSA)*, vol. 3482, pp. 1129–1138, 2005.

[29] K. Zhou, M. Gong, and B. Guo, “Data-parallel octrees for surface reconstruction,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 5, pp. 669–681, May 2011.

[30] A. Zakhor and C. Frueh, “Automatic 3d modeling of cities with multimodal air and ground sensors,” *Multimodal Surveillance, Sensors, Algorithms and Systems*, pp. 339–362, 2007.

[31] J. Chen and B. Chen, “Architectural modeling from sparsely scanned range data,” *Springer*, pp. 223–236, November 2007.

[32] H. Karimi, A. Khattak, and J. Hummer, “Evaluation of mobile mapping systems for roadway data collection,” *Journal of Computational Civil Engineering*, vol. 14, pp. 168–173, 2000.

[33] R. Li, “Mobile mapping: An emerging technology for spatial data acquisition,” *Photogrammic Engineering and Remote Sensing*, vol. 63, pp. 1085–1092, 1997.

[34] P. Tang, D. Huber, B. Akinci, R. Lipman, and A. Lytle, “Automatic reconstruction of as-built building information from laser-scanned point clouds: A review of related techniques,” *Elsevier*, pp. 829–843, June 2010.

[35] A. Chauve, P. Labatut, and J. Pons, “Robust piecewise-planar 3d reconstruction and completion from large-scale unstructured point data,” *Computer Vision and Pattern Recognition (CVPR)*, 2010.

[36] N. Kawai, A. Zakhor, T. Sato, and N. Yokoya, “Surface completion of shape and texture based on energy minimization,” *International Conference on Image Processing (ICIP)*, September 2011.

[37] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, “The ball-pivoting algorithm for surface reconstruction,” *IEEE Transactions on Visualization and Computer Graphics*, December 1999.

[38] A. Mhatre and P. Kumar, “Projective clustering and its application to surface reconstrution,” *Proceedings of the twenty-second annual symposium on computational geometry (SCG)*, June 2006.

[39] E. Turner and A. Zakhor, “Sharp geometry reconstruction of building facades using range data,” *International Conference on Image Processing*, September 2012.

[40] A. Nealen, “An as-short-as-possible introduction to the least squares, weighted least squares and moving least squares methods for scattered data approximation and interpolation,” May 2004.

[41] I. Autodesk, “Building information modeling,” *White Paper*, 2003.

[42] S. Ochmann, R. Vock, R. Wessel, M. Tamke, and R. Klein, “Automatic generation of structural building descriptions from 3d point cloud scans,” *International Conference on Computer Graphics Theory and Applications*, no. 9, January 2014.

[43] A. Adan and D. Huber, “3d reconstruction of interior wall surfaces under occlusion and clutter,” *International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission*, pp. 275–281, 2011.

[44] S. A. A. Shukor, K. W. Young, and E. J. Rushforth, “3d modeling of indoor surfaces with occlusion and clutter,” *International Conference on Mechatronics*, pp. 282–287, April 2011.

[45] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. The MIT Press, 2005.

[46] S. Shen, N. Michael, and V. Kumar, “Autonomous multi-floor indoor navigation with a computationally constrained mav,” *IEEE International Conference on Robotics and Automation*, pp. 20–25, 2011.

[47] A. Bachrach, S. Prentice, R. He, P. Hentry, A. Huang, M. Krainin, D. Maturana, D. Fox, and N. Roy, “Estimation, planning, and mapping for autonomous flight using an rgb-d camera in gps-denied environments,” *The Internatiional Journal of Robotics Research*, no. 31, pp. 1320–1343, 2012.

[48] E. Brunskill, T. Kollar, and N. Roy, “Topological mapping using spectral clustering and classification,” *International Conference on Intelligent Robots and Systems*, pp. 2491–3496, October 2007.

[49] M. Smith, I. Posner, and P. Newman, “Adaptive compression for 3d laser data,” *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 914–935, June 2011.

[50] M. F. Fallon, H. Johannsson, J. Brookshire, S. Teller, and J. J. Leonard, “Sensor fusion for flexible human-portable building-scale mapping,” *Intelligence Robots and Systems*, pp. 4405–4412, October 2012.

[51] T. Liu, M. Carlberg, G. Chen, J. Chen, J. Kua, and A. Zakhor, “Indoor localization and visualization using a human-operated backpack system,” in *Indoor Positioning and Indoor Navigation (IPIN), 2010 International Conference on*. IEEE, 2010, pp. 1–10.

[52] G. Chen, J. Kua, S. Shum, N. Naikal, M. Carlberg, and A. Zakhor, "Indoor localization algorithms for a human-operated backpack system," *3D Data Processing, Visualization, and Transmission*, May 2010.

[53] J. Kua, N. Corso, and A. Zakhor, "Automatic loop closure detection using multiple cameras for 3d indoor localization," *IS&T/SPIE Electronic Imaging*, pp. 82 960V–82 960V, January 2012.

[54] N. Corso and A. Zakhor, "Indoor localization algorithms for an ambulatory human operated 3d mobile mapping system," *Remote Sensing*, vol. 5, no. 12, pp. 6611–6646, October 2013.

[55] B. Okorn, X. Xiong, B. Akinci, and D. Huber, "Toward automated modeling of floor plans," *3D Processing, Video, and Transmission (3DPVT)*, 2009.

[56] C. Weiss and A. Zell, "Automatic generation of indoor vr models by a mobile robot with a laser range finder and a color camera," *Autonome Mobile Systeme 2005*, pp. 107–113, 2006.

[57] A. Nuchter, H. Surmann, and J. Hertzberg, "Automatic model refinement for 3d reconstruction with mobile robots," *3-D Digital Imaging and Modeling*, pp. 294–401, October 2003.

[58] S. El-Hakim, L. Gonzo, F. Voltolini, S. Girardi, A. Rizzi, F. Remondino, and E. Whiting, "Detailed 3d modeling of castles," *International Journal of Architectural Computing*, vol. 5, no. 2, pp. 200–220, June 2007.

[59] E. Turner and A. Zakhor, "Watertight as-built architectural floor plans generated from laser range data," *3D image Processing, Video, and Transmission (3DimPVT)*, October 2012.

[60] ——, "Floor plan generation and room labeling of indoor environments from laser range data," *International Conference on Computer Graphics Theory and Applications*, no. 9, January 2014.

[61] C. Mura, O. Mattausch, A. J. Villanueva, E. Gobbetti, and R. Pajarola, "Robust reconstruction of interior building structures with multiple rooms under clutter and occlusions," *Proceedings IEEE Conference on Computer-Aided Design and Computer Graphics*, pp. 52–59, 2013.

[62] S. Or, K. Wong, Y. Yu, and M. M. Chang, "Highly automatic approach to architectural floorplan image understanding and model generation," *Pattern Recognition*, pp. 25–32, November 2005.

[63] R. Lewis and C. Séquin, "Generation of 3d building models from 2d architectural plans," *Computer-Aided Design*, vol. 30, no. 10, pp. 765–779, 1998.

[64] V. Sanchez and A. Zakhor, "Planar 3d modeling of building interiors from point cloud data," *International Conference on Image Processing (ICIP)*, September 2012.

[65] C. Mura, O. Mattausch, A. J. Villanueva, E. Gobbetti, and R. Pajarola, "Automatic room detection and reconstruction in cluttered indoor environments with complex room layouts," *Computers and Graphics*, vol. 44, pp. 20–32, November 2014.

[66] R. Cabral and Y. Furukawa, "Piecewise planar and compact floorplan reconstruction from images," *Computer Vision and Pattern Recognition (CVPR)*, pp. 628–635, 2014.

[67] F. Lafarge and P. Alliez, "Surface reconstruction through point set structuring," *Eurographics*, vol. 32, no. 2, pp. 225–234, 2013.

[68] J. Xiao and Y. Furukawa, "Reconstructing the world's museums," *EECV 2012 Lectures in Computer Science*, vol. 7572, pp. 668–681, 2012.

[69] S. Oesau, F. Lagarge, and P. Alliez, "Indoor scene reconstruction using primitive driven space partitioning and graph-cut," *Eurographics Workshop on Urban Data Modelling and Visualization*, 2013.

[70] L. Nan, K. Xie, and A. Sharf, "A search-classify approach for cluttered indoor scene understanding," *ACM Transactions on Graphics - Proceedings of ACM SIGGRAPH Asia*, vol. 31, no. 137, November 2012.

[71] Y. M. Kim, N. J. Mitra, D.-M. Yan, and L. Guibas, "Acquiring 3d indoor environments with variability and repetition," *ACM Transactions on Graphics*, vol. 31, no. 6, November 2012.

[72] A. Karpathy, S. Miller, and L. Fei-Fei, "Object discovery in 3d scenes via shape analysis," *IEEE International Conference on Robotics and Automation*, pp. 2088–2095, May 2013.

[73] O. Mattausch, D. Panozzo, C. Mura, O. Sorkine-Hornung, and R. Pajarola, "Object detection and classification from large-scale cluttered indoor scans," *Computer Graphics Forum*, vol. 33, no. 2, pp. 11–21, 2014.

[74] R. Newcombe, A. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," *Mixed and Augmented Reality (ISMAR)*, pp. 127–136, 2011.

[75] M. Kaess, M. Fallon, H. Johannsson, and J. J. Leonard, "Kintinuous: Spatially extended kinectfusion," *CSAIL Technical Reports*, July 2012.

[76] "Meshlab," http://meshlab.sourceforge.net/, accessed: May 12, 2015.

[77] P. Levchev, M. Krishnan, C. Yu, J. Menke, and A. Zakhor, "Simultaneous fingerprinting and mapping for multimodal image and wifi indoor positioning," *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, October 2014.

[78] F. Pomerleau, A. Breitenmoser, M. Liu, F. Colas, and R. Siegwart, "Noise characterization of depth sensors for surface inspections," *2nd International Conference on Applied Robotics for the Power Industry*, pp. 16–21, September 2012.

[79] U. Wong, A. Morris, colin Lea, J. Lee, C. Whittaker, B. Garney, and R. Whittaker, "Comparative evaluation of range sensing technologies for underground void modeling," *International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[80] M. J. Caruso, "Applications of magnetic sensors for low cost compass systems," *Position Location and Navigation Symposium*, pp. 177–184, March 2000.

[81] P. Guo, H. Qiu, Y. Yang, and Z. Ren, "The soft iron and hard iron calibration method using extended kalman filter for attitude and heading reference system," *Position Location and Navigation Symposium*, pp. 1167–1174, May 2008.

[82] "Magnetic declination," http://www.magnetic-declination.com/, accessed: March 11, 2015.

[83] K. Shoemake, "Animating rotation with quaternion curves," *ACM Transactions on Graphics (TOG)*, vol. 19, no. 3, pp. 245–254, July 1985.

[84] O. Oreifej, J. Cramer, and A. Zakhor, "Automatic generation of 3d thermal maps of building interiors," *American Society of Heating, Refrigerating, and Air-Conditioning Engineers (ASHRAE) Annual Conference*, June 2014.

[85] P. Cheng, M. Anderson, S. He, and A. Zakhor, "Texture mapping 3d planar models of indoor environments with noisy camera poses," *SPIE Electronic Imaging Conference*, February 2013.

[86] I. T. Jolliffe, *Principal Components Analysis, Second Edition.* Springer, 1986.

[87] D. Hahnel, W. Gurgard, D. Fox, and S. Thrun, "An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements," *International Conference on Intelligent Robots and Systems*, vol. 1, pp. 206–211, October 2003.

[88] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, February 2007.

[89] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, June 1981.

[90] J. Doyle and R. L. Rivest, "Linear expected time of a simple union-find algorithm," *Information Processing Letters*, vol. 5, pp. 146–148, November 1976.

[91] T. A. Funkhouser, C. H. Sequin, and S. J. Teller, "Management of large amounts of data in interactive building walkthroughs," *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, pp. 11–21, 1992.

[92] *Americans with Disabilities Act*, U.S. Architectural and Transportation Barriers Compliance Board, 1331 F Street N.W. Suite 1000 Washington D.C. 20004-1111, July 1990, aNSI A117.1-1980.

[93] E. Turner and A. Zakhor, "Watertight planar surface meshing of indoor point-clouds with voxel carving," *International Conference on 3D Vision*, June 2013.

[94] G. Grisetti, C. Stachniss, and W. Burgard, "Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling," *Proceedings of IEEE International Conference of Robotics and Automation*, pp. 2443–2448, April 2005.

[95] M. A. Shelley, "Monocular Visual Inertial Odometry on a Mobile Device," Master's thesis, Technischen Universitat Munchen, 2014.

[96] J. G. R. III, A. J. B. Trevor, C. Nieto-Granda, A. Cunningham, M. Paluri, N. Michael, F. Dellaert, H. I. Christensen, and V. Kumar, "Effects of sensory precision on mobile robot localization and mapping," *Experimental Robotics*, vol. 79, pp. 433–446, 2014.

[97] C. Hernandez, G. Vogiatzis, and R. Cipolla, "Probabilistic visibility for multi-view stereo," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, November 2007.

[98] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: an efficient probablistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[99] E. Turner, "Video of generated models," http://www-video.eecs.berkeley.edu/research/indoor/video/jstsp2014.wmv.

[100] "Baumann consulting - energy modeling, building and systems simulation," http://www.eb-engineers.com/index.php?option=com_content&task=view&id=36&Itemid=46, accessed: March 27, 2015.

[101] R. Zhang and A. Zakhor, "Automatic identification of window regions on indoor point clouds using lasers and cameras," *IEEE Winter Conference on Applications of Computer Vision*, March 2014.

[102] "Mall map of stoneridge shopping center," http://www.simon.com/mall/stoneridge-shopping-center/map, accessed: March 27, 2015.