LIGHTWEIGHT 3D MODELING OF URBAN BUILDINGS FROM RANGE DATA

by

WEIHONG LI

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy,

The City University of New York

2011

This manuscript has been read and accepted for the

Graduate Faculty in Computer Science in satisfaction of

the dissertation requirement for the degree of Doctor of Philosopy.

Date	Dr. George Wolberg, Chair of Examining Committee
Date	Dr. Theodore Brown, Executive Officer

Dr. Zhigang Zhu

Dr. Ioannis Stamos

Dr. Michael Reed

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

LIGHTWEIGHT 3D MODELING OF URBAN BUILDINGS FROM RANGE DATA

by

Weihong Li

Advisor: Professor George Wolberg

Laser range scanners are widely used to acquire accurate scene measurements. The massive point clouds they generate, however, present challenges to efficient modeling and visualization. State-of-the-art techniques for generating 3D models from voluminous range data is well-known to demand large computational and storage requirements. In this thesis, attention is directed to the modeling of urban buildings directly from range data. We present an efficient modeling algorithm that exploits *a priori* knowledge that buildings can be modeled from cross-sectional contours using extrusion and tapering operations. Inspired by this simple workflow, we identify key cross-sectional slices among the point cloud. These slices capture changes across the building facade along the principal axes. Standard image processing algorithms are used to remove noise, fill holes, and vectorize the projected points into planar contours. Applying extrusion and tapering operations to these contours permits us to achieve dramatic geometry compression, making the resulting models suitable for web-based applications such as

Google Earth or Microsoft Virtual Earth. This work has applications in architecture, urban design, virtual city touring, and online gaming. We present experimental results on the exterior and interior of urban building datasets to validate the proposed algorithm.

Acknowledgements

I would like to express my gratitude to my advisor, George Wolberg, for his support, invaluable suggestions and encouragement on issues that were beyond my research. His systematic thinking and many comments and ideas have undoubtedly boosted the quality of the work.

I also owe a lot to my co-advisor, Zhigang Zhu who brought me to the computer vision research area. My technical skills and research interest were gradually built when working with Professor Zhu during my early stage of the doctorate journey. My sincere thanks go to the rest of my thesis committee as well, Ioannis Stamos who gave insightful suggestions and comments on my research and provided the range datasets for the experiments and Michael Reed who spent time to serve on my thesis committee as an outside member and to supervise my dissertation defense.

I would like to give a special thank-you to Siavash Zokai. His solid background knowledge, in-depth discussion and technical supports helped me out

from those difficult time frequently. I would also like to thank all those who have contributed to this work by providing valuable input and suggestions. Gene Yu shared experiences on point cloud data processing and highlighting. Hadi Fadaifard's normal detection code made major facades detection much easier. Hao Tang suggested some literature and techniques on related work.

I would also like to express gratitude to my parents. Special thanks to my mother-in-law who took care of little Brian, my lovely son, so that I could be more focused on this research. Finally, I must express my appreciation to my wife, Pingna. She has been a source of constant support throughout the many long days spent working on this dissertation. For this, and many other things, I am profoundly grateful.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The 3D modeling of urban buildings is an area of active research with increasing attention drawn from the computer graphics and computer vision communities. A survey by Tang *et al.* offers a good overview of the current status of research and techniques used for building information modeling [1]. Current state-of-the-art algorithms include procedural modeling, 3D laser scanning, and image-based approaches. In addition, conventional modeling tools are commonly used for this purpose. The most accurate input source for modeling *existing* buildings, though, remains laser range scanners [2]. They provide high geometric detail by collecting range data from hundreds of meters away with an accuracy on the order of a few millimeters [3]. This fidelity is appropriate for

construction, architecture, cultural heritage, and forensics applications. Unfortunately, laser range scanning can produce an overwhelming amount of data, which poses great challenges to visualization software that require lightweight 3D models for interactive use. Polygonal data generated from range scans are therefore too dense for use in web-based applications such as Google Earth [4] and Microsoft Virtual Earth [5]. These applications work best with lightweight models consisting of only hundreds of polygons.

The goal of this work is to produce high-quality lightweight models of urban buildings from large-scale 3D range data in a semi-automatic way. The proposed solution is inspired by the simple paradigm embedded in procedural modeling as well as interactive tools such as Google SketchUp. The core of these methods is that a simple set of extrusion and tapering operations applied to 2D contours can grow a wide array of complex 3D urban models. We propose a reverse engineering approach to infer key cross-sectional planar contours along with a set of extrusion and tapering operations to derive lightweight models that conform to the 3D range data.

The main contribution of this work is that it provides a framework for lightweight modeling of urban buildings from heavy 3D point cloud datasets (PCD) based on a series of extrusion and tapering operations along dominated axes. The proposed framework combines the benefits of *a priori* knowledge of

urban buildings and fast 2D image processing techniques to perform 3D modeling of urban buildings and can generate models across a wide spectrum of resolutions. This offers the benefit of a cost-effective geometry compression approach for voluminous range data within the domain of urban structures. A particularly useful feature of the algorithm is that it outperforms existing approximation techniques by preserving the sharpness of the raw data, even at low resolution. It can be applied to boost web-based 3D applications, virtual city touring, and online gaming.

1.1 Related Work

In an attempt to steer clear of tedious and expensive hand-made models, procedural modeling of buildings in [6–8] has been proposed. By using an effective description language, buildings and streets of a virtual city can be generated automatically. The strength of this approach is that the description language can generate a huge number of buildings and streets quickly and beautifully. This is particularly useful for gaming and other computer graphics applications. However, since the parameters used to generate the buildings are randomly generated, the city generated with these buildings and streets is a virtual one. This approach is not useful for attempting to model an *existing* building. In order to do so, one has to manually specify the parameters of the building, which is

very cumbersome. Our goal is to automatically infer the contours and extrusion/tapering parameters of an existing building directly from dense range data.

Reconstruction of 3D models from range data has been addressed in [9–11] with applications in numerous research areas, including computer-aided design (CAD), computer vision, architectural modeling, and medical image processing. The authors in [12] use a histogram of height data to detect floors and ceilings for creating accurate floor plan models of building interiors. In [13], the authors proposed a 3D building reconstruction from a 2D floor plan image. With the help of a 2D floor plan image, both the interior and exterior of a building can be reconstructed accordingly. A survey on methods for generating 3D building models from architectural floor plans is given in [14]. However, reliance on 2D floor plans makes these approaches too limiting for most applications, including our project. In [15], known manufacturing features were used to infer the 3D structure of mechanical parts. Their method benefits from the domain knowledge that most of the mechanical parts consist of predefined structures, such as holes, bosses, and grooves. Our work is partially motivated by this idea since it also incorporates *a priori* knowledge about the construction of urban buildings for further inference. However, their method is based on predefined simple geometry structures and the assumption that the input 3D data has no holes. This hinders their approach for those applications with incomplete data.

Medical image processing techniques are usually dealing with low Signal-to-Noise Ratio (SNR) data. There has been a lot of work on the medical 3D image reconstruction as in [16–20]. The basic ideas behind these approaches are 3D reconstruction from sliced or histologic images using interpolation techniques. The statistical inference are also intensively used to infer the low SNR images. In [21], Sigworth tried to deal with low SNR image data using maximum likelihood approach. Because most of the statistical processes are computational intensity, these approaches usually are heavy-duty approaches in order to obtain accurate, high resolution models.

Multi-modal data fusion is another approach for large-scale urban environment modeling where the LIght Detection And Ranging (LIDAR) scans are widely used [22]. In [23], both air and ground data are fused, where LIDAR scans are used to create the models and the camera images are used for texture mapping. Citing the cumbersome and expensive use of laser scanners, the researchers in [24] propose an approach that relies solely on passive sensors (cameras) mounted on a moving vehicle. Dense 3D point cloud measurements are derived using their multi-view stereo module based on multiple plane sweeping directions. In an attempt to compress the voluminous data produced in the method of [24], Xiao *et al.* introduced an approach for modeling facades along a street using *a priori* knowledge about the buildings based on images [25]. They achieve geometry compression and deliver a clean approximation of the facades

by applying a combination of plane fitting and window detection. However, their automatic depth reconstruction techniques based on images may fail when modeling highly reflective mirror-like buildings.

Toshev *et al.* proposed grammar based method for detecting and parsing buildings from unorganized street-level point clouds [26] . Despite its efficiency on modeling, the results could not generate enough level of details for the buildings. For example, the windows and doors are missing in the final models. The same situation appears in [27], where only building footprints are extracted from the range data. Johnston and Zakhor developed a method to generate interior building floor plans from the exterior of the structure [28]. They use a laser scanner to measure the range of hundreds of thousands of points on interior walls of the building, exploiting the fact that the laser can go through unobstructed windows. The issue is when there are obstructed windows or many objects behind the windows, the inferred floor plans become much less accurate. Vanegas *et al.* presented a work that examines the changes in building geometry from principle directions and reconstructs the models using Manhattan-World grammars [29]. This work produces models with much higher resolutions from multiple calibrated aerial images, instead of 3D point clouds. However, this method is not able to model tapered structures which exist widely in buildings and can be handled by our approach.

The ball-pivoting algorithm (BPA) is an efficient technique for meshing 3D

point clouds to produce polygonal models [30]. The generated meshes, however, constitute heavyweight models, with the number of vertices nearly approaching the number of points in the 3D point cloud. This limits its usefulness for web-based applications. Although a BPA model can be simplified using approximation techniques such as *qslim* [31], the sharp detail of the original model is not preserved. Recently, Sareen *et al.* proposed a contour based 3D point cloud simplification approach for modeling free-form surface [32]. The authors first extracted 2D slices from equally spaced slabs of point clouds. In the second stage, a cubic B-spline curve is used to approximate each 2D slice with the number of control points predefined by users. The proposed method was applied to a facial scan data with 50k data points and can achieve 5-20% of overall size reduction ratio. This is not scalable enough for point clouds of urban buildings which usually consist of millions of data points and require much more size reduction ratio.

Simplification of 3D buildings is also an active research topic in laser scanning research community [33–35]. Gonzalvez *et al.* proposed an automatic approach to convert a laser scanner point cloud into a realistic 3D polygonal model [33]. They first segmented the unstructured point cloud into sections based on vertical walls which are the mainly characteristic of the shapes of the buildings. Afterwards, sections and profiles are simplified automatically based on Douglas-Peucker algorithm followed by a clustering process which groups all related sections to extract partial primitives. Finally, a growing clustering of each

partial primitive is conducted to obtain a global primitive. The main issues in their method is that a number of thresholds have to be tested by the user before obtaining good results, and the underlying building must be linear in order to apply the proposed approach.

In addition to the aforementioned research projects carried on in academia, some commercial products are developed in start-up companies. In [36], the buildings in Manhattan are modeled to enhance the virtual reality of social network. The buildings are accurately modeled via aerial LIDAR data and were associated with address and related social information. Another commercial product is EdgeWise^{TM}, a new developed product by ClearEdge^{3D} [37]. They first apply ground extraction to classify the point cloud data. And then the polygons are inferred for each classified data points and are exported to be edited in Google SketchUp. Essentially, this commercial tool only provides a good starting point for interactive editing.

1.2 Overview

We propose an efficient way to reconstruct 3D models from range data by partitioning the data into thin cross-sectional volumetric slabs. For each slab, all range data in that slab is projected onto a 2D cross-sectional contour slice. Producing this array of slices permits us to avoid costly computation directly on

3D data. A similarity measure is used to cluster the sliced images together into *keyslices*. This term is analogous to the use of "keyframes" in computer animation, which denotes important snapshots in the animation sequence from which intermediate results can be derived. In essence, each keyframe is a slice in the spatiotemporal volume of an animation. Similarly, each keyslice is a 2D image which contains a *transitional* cross-section of the building, encapsulating major contours in the facade. The model is then generated by applying basic extrusion and tapering operations from one keyslice to the next. This produces a lightweight representation consisting of only a few hundred polygons.

An overview of our approach, depicted in Fig. 1, begins with the acquisition of a dense 3D point cloud C of a building. C is then partitioned into a non-overlapping set of volumetric slabs. Each slab S is associated with one projection plane P, sitting at the base of S. The purpose of partitioning C is to establish a set of cross-sections, or contour slices. By examining the changes among these slices, we can identify the prominent slices, or *keyslices* K, as well as the necessary extrusion and tapering operations that must apply to them to generate the model. By casting this 3D modeling task into a series of 2D operations, we reduce the dimension of the problem to achieve a significant savings in computational complexity. Boundary vectorization are then carried out to these raster images K to obtain the contours T. After applying the extrusion and tapering operations on T, the final model can be generated and rendered for visualization.

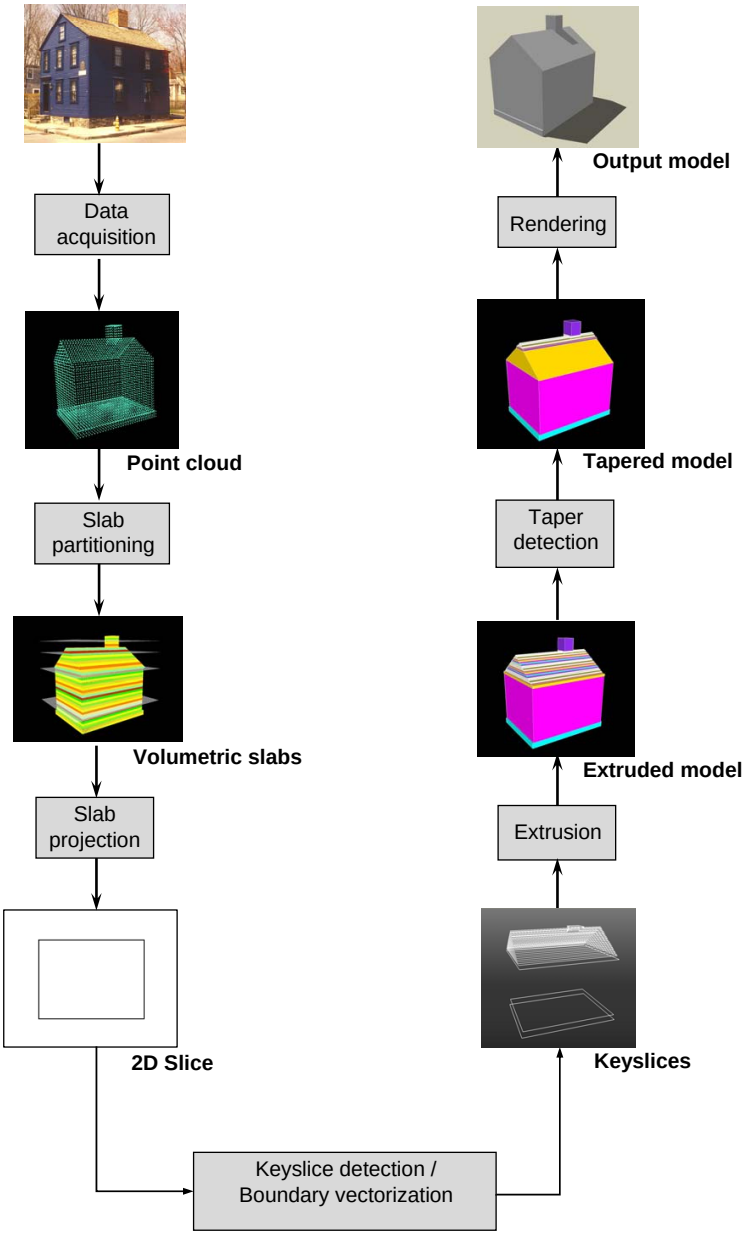


Figure 1: Overview of the proposed approach.

The modular flow diagram for our system is shown in Fig. 2. The whole system consists of three stages of computation. The first stage mainly contains preprocessing modules. In this stage, 3D synthetic datasets are generated and 3D real datasets are obtained as input to the whole system. Major plane detection is carried out to compute the normals of the major facades for extracting 2D slices which are then enhanced by noise removal and hole filling. These enhanced 2D slices are then segmented for further processing. The second stage is iteratively applied to each segment computed in the first stage and plays key role for model reconstruction. First, both distance measurement based and curvature based keyslice detection are conducted in the keyslice detection module to locate those prominent slices for the current segment. Following this, boundary vectorization is carried out on these raster keyslices to obtain the contours for reconstruction. Furthermore, linear tapered structure is computed to reduce the model size. The final stage is to model the range data based on information computed in previous stages. In model generation module, each segment is reconstructed by applying the extrusion and tapering operations on the keyslice contours. These models are then assembled in the module of model merging. Finally, if exist, windows and doors are projected to generate the whole model.

The point cloud datasets in our experiments consist of synthetic and real ones. The synthetic datasets are generated from 3D building models with no noise introduced and are used for proof of concept. These 3D building models

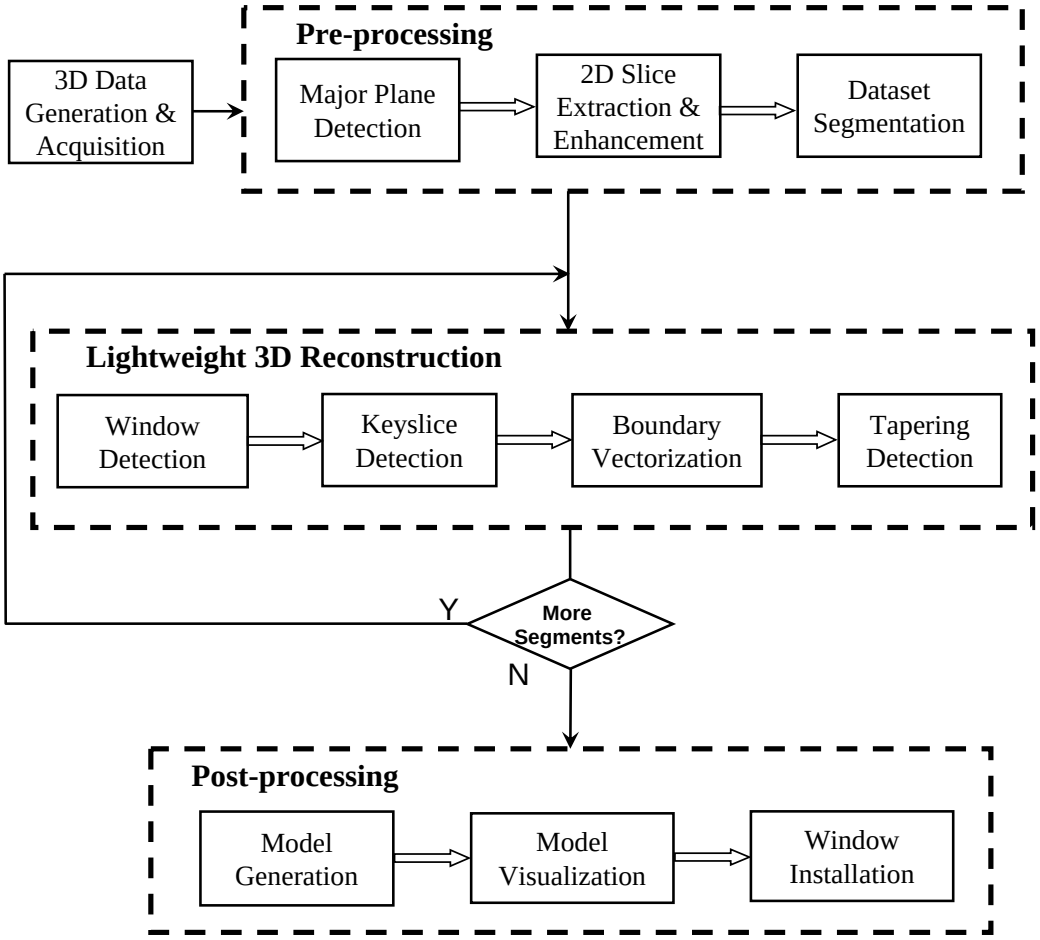


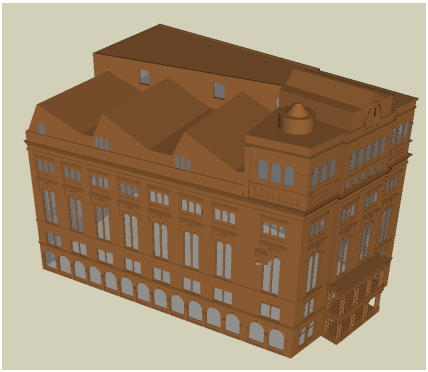
Figure 2: Flow graph for proposed approach.

which were downloaded from Google SketchUp 3D warehouse contain 3D faces and their normals. Because we have the ground truth of reconstructed models, that is, the original models, the synthetic datasets can also be used to check the correctness and measure the errors for reconstructed models. On the other hand, the real datasets are acquired from laser scanners. Due to the nature of buildings (with windows, doors) and blocking, the real datasets also contain all kind of noise and they are only used to validate the proposed approach.

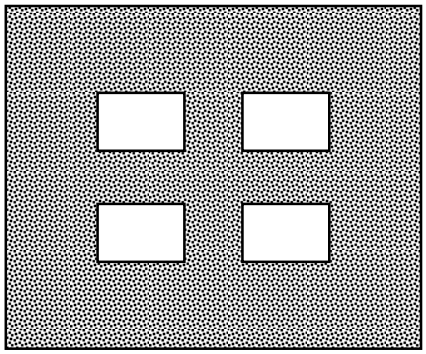
1.3 Synthetic Data Generation

For a Google SketchUp building model, it needs to be exploded to decompose any grouped faces. This ensures each individual face of the model can be accessed. We wrote a Ruby script using SketchUp APIs to dump all faces and their normals into a file as input for point cloud dataset generation. All faces laying on the same plane are marked, which is an important information for correct data generation.

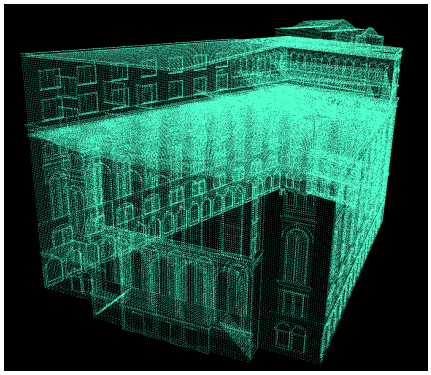
With these faces and their normals, we can carry out the synthetic 3D point cloud data generation directly. Alternatively, we can transform each 3D plane to 2D space, and then transform those sampled data generated on 2D space back to 3D space. To simplify the process, the co-planar 3D faces are first transformed onto a 2D plane. Let $\mathbf{V} = \{v_0, v_1, ...v_i, ..., v_0\}$ be the set of vertices for a 3D face. Let



(a)



(b)



(c)

Figure 3: The synthetic point cloud generation: (a) the snapshot of a 3D model; (b) the example of the sampling for embedded faces; (c) the snapshot of the synthetic 3D point cloud from (a).

F be the set of faces sitting on the same plane. Let N_v be the normal of the plane and $|N_v| = 1$. **V**,**F** and N_v are known from the SketchUp model and are input for the process. We can construct another vector on the face, say $N_p = \overline{v_0 v_1}$ and normalize the vector $|N_p| = 1$, N_p and N_v are perpendicular to each other. Let $N_s = N_p \times N_v$, that is, the cross product of the N_p and N_v. The transform matrix $M = [N_p, N_v, N_s]^T$ transforms the 3D vector onto an 2D plane of the coordinate system with the axis N_p, N_v, and N_s.

The remaining data generation process is similar to the scan line based polygon filling. A parameter s is used to control the distance between scan lines and to define the interval between two consecutive synthetic data points. For each scan line, we count the number of intersection points, n, between the scan line and the face edges. When n is an odd number, it starts to add points on scan line with the interval s. For example, assume the current scan line is $y = 8$ and $(2,8)$ is a new generated synthetic data point. With the parameter $s = 4$, the next data point is $(6,8)$. This process continues until the counter n becomes an even number. In this case, the process stops adding points unless the counter n changes back to an odd number. If the process reaches the end of the current scan line, it will start the same process for the next scan line until it finishes scanning the whole image. Some special cases need to be handled, such as the coherence of scan line and edges. Once the process is done on the 2D plane, the generated data points can be transformed back to 3D coordinate system using the inverse

matrix M^{-1}.

The previous process is actually a grid based process for synthetic data generation. Alternatively, a sampling based process on the scan line can be conducted to generate synthetic point cloud dataset. For each scan line, we first compute all valid line segments which are actually part of the polygons to be filled. And then, we conduct uniform sampling on these line segments to generate synthetic point cloud data. We have tried both methods to generate synthetic data points and found that the point cloud datasets generated from the two methods are almost identical to our model reconstruction.

An example of the sampled image is shown in Fig. 3(b). There are four inner faces (windows) sitting inside an outer face (wall), which should be leave untouched (white areas). The sampled location are indicated with dark points. The parameter s can be used to adjust the density of the sampled points. The smaller value s is, the denser point cloud data would be. In our experimental settings, $s = 1.0$ is used to generate point cloud dataset as shown in Fig. 3(c), which consists of approximately 10 million points from the 3D model shown in Fig. 3(a).

1.4 Real Data Acquisition

The real datasets are assembled from range data obtained from a Leica Cyrax 2500 laser range scanner [3], which works by sweeping an eye-safe laser beam

across the scene to collect up to one million 3D depth points per frame. All scene points that lie within 100 meters can be acquired with an accuracy of 5mm in depth. The basic algorithm that we use for registering the voluminous 3D data acquired from multiple scans of buildings has been introduced in [38]. That same algorithm is also responsible for extracting the major axes of the building in order to align it to the axes of the world coordinate system [39, 40]. This is necessary to properly infer the keyslices. Fig. 4(b) displays a properly aligned, *registered* 3D point cloud consisting of 14 scans totalling 14 million points collected from Fig. 4(a).

Due to occlusions and limited vantage points, the point cloud collected by the laser scanner contains artifacts and holes. In addition, computing directly on 3D data is time-consuming and computationally complex. To tackle these issues, we define inner and outer bounding boxes for the building to clip away unrelated scene objects. Then, we convert the 3D modeling problem into a set of 2D problems by projecting the 3D data into a series of 2D cross-sectional contour images. Noise removal, hole filling, and vectorization are all done in this 2D space.

(a)

(b)

Figure 4: The real point cloud acquisition: (a) image of building to be modeled; (b) 3D point cloud of building assembled by registering multiple scans.

Chapter 2

3D Data Preprocessing

The synthetic input data to our system is unorganized point cloud data (PCD), namely, no underlying structure information about buildings is available for modeling. Furthermore, they were not rectified and the major facades or planes of the building were not aligned with the world coordinates. For the real dataset, although the major axes of the building have been extracted during the registration stage, they might be not precise enough for computing major facade planes. We need to know the major facade planes and their normals in that they will be used to project PCD to 2D image slices, which are the starting point for the further computation and inference. Consequently, as one of the pre-processing stages, the PCD will be transformed to be aligned with world coordinates along the major planes.

2.1 Major Facades Detection

Numerous approaches have been proposed on plane and normal detection on PCD. Based on the techniques used in these approaches, they can be divided into two different categories.

1. **Sampling and statistical based:**

   The sampling and statistical methods mainly use the RANdom SAmple Consensus (RANSAC) techniques [41] to obtain small set of sampled points for computing the planes [42–44]. RANSAC is widely used in plane detection due to its conceptual simplicity, generality and reliability on noise. The RANSAC method computes planes by randomly choosing small sets from the PCD and constructing corresponding plane parameters. The resulting candidate plane is tested against all points in the data to determine how many of the points are well approximated. After a given number of trials, the plane which approximates the most points is extracted and the algorithm continues on the remaining data. However, the major issue of RANSAC is that it would find a wrong plane if the data has a complex geometry.

   In [45], the authors tried to overcome the above issue by combining RANSAC with Minimum Description Length (MDL) which is used to deal with several competing hypothesis. They first partitioned the PCD into

small rectangle blocks to ensure that there is no more than three planes in each block. After this, they applied RANSAC on each block to extract planes. The MDL is carried out to avoid detecting wrong planes due to the complex geometry of the 3D data. Finally, they applied region growing to merge neighboring planes with certain local range. The major issue about this method is that it required considerable computation resource if no further optimizations are applied.

Plane fitting is also widely used for surface extraction from noisy PCD. Mitra and Nguyen proposed and analyzed a method based on local least square fitting for estimating the normals at all sample points of a PCD [46]. Given a set of local points p_i, $1 \leq i \leq k$, they tried to search for a line $a^T x = c$, with $a^T a = 1$, such that the sum of square distance from the points p_i's to the line is minimized. Theoretical bound on the maximum angle between the estimated normal and the true normal of the underlying PCD is computed. This theoretical study provides a way to find an optimal neighborhood size to be used in the least square method.

2. **Hough space based:**

   This category is a closed-form computation based on Hough space. Vosselman and Dijkman [47] proposed 3D Hough transform (HT) to detect roof planes from PCD acquired from airborne laser altimeter. Each point

(x,y,z) in a laser dataset defines a plane $z = ax + by + c$ in the 3D parameter space spanned by the axes of the parameters a, b and c, where a and b are the slopes in $x-$ and $y-$ direction and c denotes the vertical distance of the plane to the origin. This representation can work well on detecting near horizontal planes, as those from airborne laser scanner dataset [48]. But it suffers from inferring near vertical planes due to the potential large value range of the parameters. Shah improved the representation by defining 3D plane with its normal direction $\hat{\mathbf{n}} = (n_x, n_y, n_z)$ and its perpendicular distance from the origin ρ, which is able to detect vertical planes efficiently [49]. However, it suffers the discretization problem when the distance ρ is relative large. Moreover, since this approach is working on the entire points, it also requires numerous memory resource when dealing with large-scale PCD.

In order to process large-scale dataset efficiently, we proposed a plane estimation using a local plane fitting based normal computation for each 3D point. By doing local computation, the memory resource requirement is minimized and the computational requirement is largely reduced. Although in theory, the accuracy of the normal for each point is not as precise as those computed globally, it works well in our case because the majority data points are those representing the walls/facades of the building.

2.1.1 Major Plane Detection

The Hough space based plane detection described in Appendix C can work efficiently if a grid parameterization is available. However, if such a grid is not available, it will be very inefficient due to the global quantization of ρ in Eq. (15). To solve this issue, we adopted a local plane fitting method to compute the normals for each 3D point. Once we obtain the normals in Euclidean representation, we can convert it to spherical coordinates representation for quantization. Hough transform can then be used to identify the largest N voted normals as the normals of the major planes. Usually, N is a predefined value by users and represents the number of facades a building consists of.

We used moving least squares (MLS) for deriving a smooth plane from a set of neighboring data points in space for normal computation. A survey of plane detection on point set was done by Cheng *et al.* [50]. Given n points in $\mathbf{R}^d$, i.e., $\{\mathbf{x}_i \in \mathbf{R}^d | i \in [1...n]\}$, we want to obtain a fitting function $f(\mathbf{x})$ that approximates the given scalar values f_i at points $\mathbf{x}_i$. The idea is to start with a weighted least squares (WLS) formulation $f'(\mathbf{x})$ for an arbitrary fixed point in $\mathbf{R}^d$, and then move this point over the entire parameter domain, where a WLS fit is evaluated for each point individually. The fitting function $f(\mathbf{x})$ is obtained from a set of local

approximation functions $f'(\mathbf{x})$:

$$f(\mathbf{x}) = \underset{f' \in \Pi_m^d}{\arg\min} \sum_{i=1}^{n} w_i(\|\mathbf{x}-\mathbf{x}_i\|)\|f'(\mathbf{x}_i)-f_i\| \tag{1}$$

where f' is taken from Π_m^d, the space of polynomials of total degree m in d spatial dimensions, and can be written as

$$f'(\mathbf{x}) = \mathbf{b}(\mathbf{x})^T \mathbf{c}(\mathbf{x}) \tag{2}$$

where $\mathbf{b}(\mathbf{x}) = [b_1(\mathbf{x}),...,b_k(\mathbf{x})]^T$ is the polynomial basis vector and $\mathbf{c} = [c_1(\mathbf{x}),...,c_k(\mathbf{x})]^T$ is the vector of unknown coefficients, which we want to minimize. In general, the number k of elements in $\mathbf{b}(\mathbf{x})$ is given by $k = (d+m)!/m!d!$ [51]. The weighting function w_i is monotonic decreasing as the distance from $\mathbf{x}$ increases. There are many options for choosing the weighting function, such as a Gaussian

$$w_i(d) = e^{-d^2/h^2} \tag{3}$$

where h is a spacing parameter which can be used to smooth out small features in the data.

Solving the minimization problem by taking partial derivatives with respect

to the unknown coefficients $c_1, ..., c_k$, we obtain,

$$\mathbf{c}(\mathbf{x}) = \left[\sum_{i=1}^{n} w_i(\|\mathbf{x}-\mathbf{x}_i\|)\mathbf{b}(\mathbf{x}_i)\mathbf{b}(\mathbf{x}_i)^T\right]^{-1} \sum_{i=1}^{n} w_i(\|\mathbf{x}-\mathbf{x}_i\|)\mathbf{b}(\mathbf{x}_i) f_i \tag{4}$$

For our 3D plane fitting, we have $m = 1$, $d = 3$ and $\mathbf{b}(\mathbf{x}) = [1, x, y, z]^T$. The normal is given by the eigenvector of the square matrix $\sum_{i=1}^{n} w_i(\|\mathbf{x}-\mathbf{x}_i\|)\mathbf{b}(\mathbf{x}_i)\mathbf{b}(\mathbf{x}_i)^T$ that corresponds to the smallest eigenvalue, which can be computed using the singular value decomposition (SVD).

In practice, we use kd-tree [52] to speed up the query of neighbor points for a given point $P(x_p, y_p, z_p)$ with distance less than a radium d. The construction of a kd-tree takes $O(NlogN)$ time, where N is the number of 3D points. Querying an axis-parallel range in a balanced kd-tree takes $O(N^{1-1/k})$ time, where k the dimension of the kd-tree. The complexity of MLS is of $O(m)$, where m is the number of neighbor points for P [53]. Therefore, the overall major plane detection is of complexity $O(NlogN)$.

2.1.2 Point Cloud Data Rectification

Once the major planes are detected, we can rectify the original data to generate 2D slices for each major plane. Assuming u, v, and n represent the three

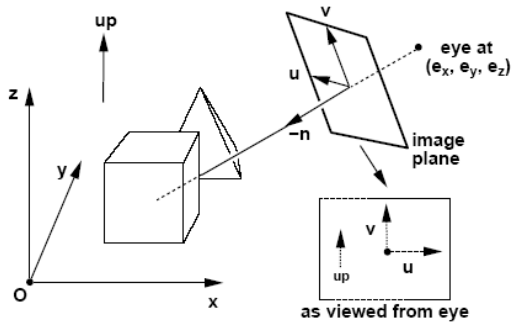


Figure 5: The transformation matrix.

major axes, we can rectify the data using the following transformation matrix **M**:

$$\mathbf{M} = \begin{pmatrix} u_x & u_y & u_z & -e_x \\ v_x & v_y & v_z & -e_y \\ n_x & n_y & n_z & -e_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

This is illustrated in Fig. 5. (x,y,z) is the world coordinate system. The rectified coordinate system is represented using (u,v,n). The vector $[-e_x,-e_y,-e_z]^T$ is the translation of the view point from the world origin. In our case, u is the bottom-up vector, and n is the normal of a major plane which is perpendicular to u. v is the cross product of u and n. After applying **M**, the original point cloud data is rectified with respect to each major plane and ready for 2D slice extraction.

2.2 2D Slice Extraction and Enhancement

We consider the point cloud data as a large array of 3D points to be sliced into horizontal volumetric slabs. All 3D points within each slab are projected onto a horizontal projection plane, or slice, at the base of the slab. Fig. 6 shows the 3D point cloud in Fig. 4(b) partitioned into 50 slabs. The 3D points in each slab are projected onto a projection plane to form cross-sectional contour slices. Fig. 7 depicts four such slices, associated with the four displayed projection planes of Fig. 6.

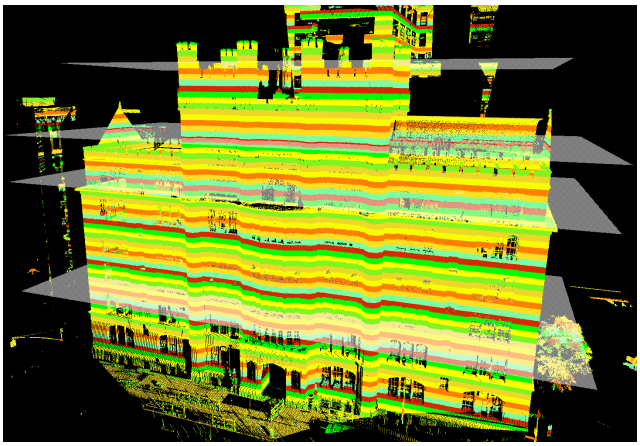


Figure 6: The 3D point cloud of Fig. 4(b) partitioned into uniform volumetric slabs. The 3D points in each slab are projected onto a projection plane to form cross-sectional slices. Four such planes are shown.

The height of each slab is δ. If δ is held constant, each slice is generated

from equi-spaced slab intervals. If δ is allowed to vary, then we may choose to allow for large values in parts of the structure that are similar, and low values in regions that contain finer detail. To avoid working on 3D data directly, we choose a relatively small constant value for δ to generate 2D cross-sectional image slices.

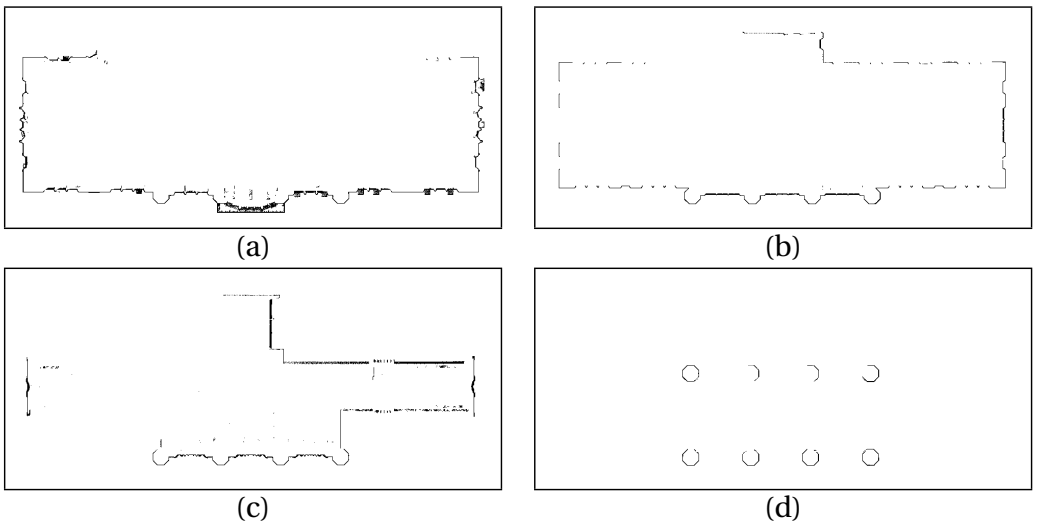


Figure 7: The set of slices corresponding to the four projection planes in Fig. 6.

Without loss of generality, the $y-$axis is used to represent the bottom-up vertical direction. Over each slab in height range $[H_{lo}, H_{hi})$, we project the 3D data $\mathbf{P}(x,y,z)$, for $H_{lo} \leq y < H_{hi}$, onto a 2D image slice. The projection is normalized in the range $[0, W]$, where W is the image width:

$$\begin{pmatrix} x^{2D} \\ y^{2D} \end{pmatrix} = \frac{W}{X_{max} - X_{min}} \cdot \begin{pmatrix} x_i^{3D} - X_{min} \\ z_i^{3D} - Z_{min} \end{pmatrix} \tag{5}$$

Note that $[X_{min}, X_{max}]$ and $[Z_{min}, Z_{max}]$ pairs define the 3D bounding box, which can be obtained through user input and can be used to clip away noise.

A simple way to implement this 2D slice projection is as follows: we first compute the total number of slices N, i.e., $N = (Y_{max} - Y_{min})/\delta$. For each slab S_i, $1 \leq i \leq N$, the height range is $[Y_{min} + \delta * (i-1), Y_{min} + \delta * i]$. Once the height range is computed, we check each 3D data point $\mathbf{P}(x,y,z)$ whether y is inside this range or not. If so, one can use Eq. (5) to compute the pixel coordinates of the image for $\mathbf{P}$. However, the number of 3D point cloud checking would be $N * C$, where C is the total number of 3D point cloud data. For a large scale dataset and a reasonable δ, this process is usually slow due to excessive checking and computation.

To improve the performance, we can reduce the number of checking to be only C by making use of extra memory space. Basically, for each slab S_i, we create a linked-list which will be used to store all points belong to this slab. For each 3D data point $\mathbf{P}(x,y,z)$, we compute its corresponding slab index i based on y value using the equation, $i = (y - Y_{min})/\delta$. Then $\mathbf{P}$ is *stored* in the linked-list associated with the slab with the index i. After going through all the 3D point cloud data once, the linked-list for each slab is filled with corresponding points. With the linked-list, it is trivial to generate the sliced image for each slab. This new implementation largely improved the projection process by reducing the redundant computation with extra memory space. Fig. 7(a)-(d) show some examples

of the 2D slices, where noise and incomplete data are observed.

2.2.1 Slice Enhancement with Hole Filling

The slices we extract above often have holes (i.e., missing data) due to occlusion or other visibility issues. Most urban buildings have symmetry structures that we can exploit to fill these holes. Symmetry computation on 3D data is an active research topic and has potentially numerous applications, such as those in [54–57]. However, symmetry computation on 3D data directly is expensive. Fortunately, we only conduct this computation on 2D image slices. Furthermore, since the 3D data has been already rectified and projected onto 2D slices, we only need to consider 2D translation for symmetry computation. Let $P(x,y)$ be a point on the original image I. Let $P'(x',y')$ be the reflected point of P with respect to a symmetry line L. The symmetry computation equation for L is as follows:

$$L = \underset{x,y}{\arg\min} \sum d_{x,y}(P', I) \tag{6}$$

where the $d_{x,y}(P', I)$ is the distance between the self-reflected point P' and its nearest data point in image I. The reflected point P' of the original point P is computed with respect to a line along either the $x-$axis or $y-$axis. Therefore, the symmetry line L is obtained as the line with minimum summation error over the reflected data points. Fig. 8(a) depicts the original input sliced image with holes.

The result after hole filling using symmetry computation is shown in Fig. 8(b).

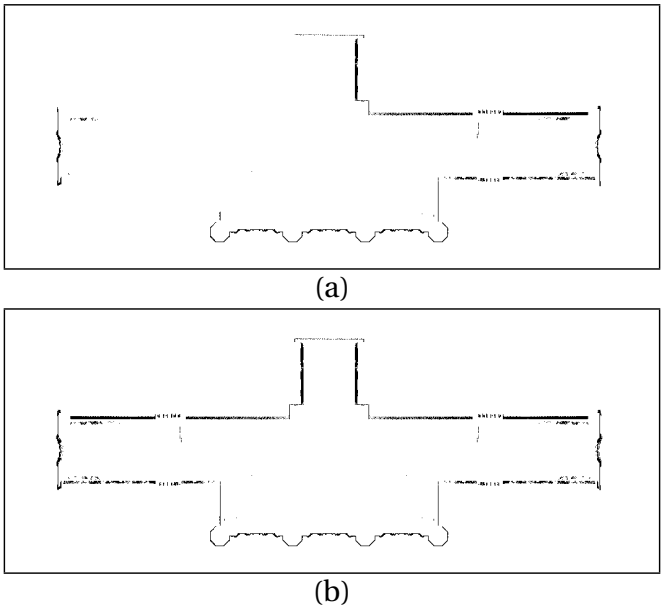


Figure 8: Symmetry-based hole filling: (a) original 2D slice image; (b) enhanced image after hole filling.

2.3 Dataset Segmentation

Modeling a building PCD as a whole structure simultaneously is complicated due to the natural complexity of buildings. To simplify this problem, We utilize the divide and conquer strategy to segment the whole PCD of a building into simpler ones. Following this, we reconstruct each segmented part based on

extrusion or tapering operations. Once each segment is processed and modeled, the whole PCD can be combined and merged into the final model.

3D point cloud data segmentation has been an active research topic for years, and Hoover *et al.* compared a variety of methods for finding planar segments in range images [58]. Essentially, segmentation is the process of labeling each data point so that the points belonging to the same structure are grouped together. Based on the techniques used for segmentation, the proposed algorithms can be roughly categorized into two groups.

1. **Edge based method:**

   Edge based methods first apply edge detector to extract the boundary edges of different regions. The edges are detected based on the local surface properties, such as surface normals, gradients, principal curvatures, or higher order derivatives. These edges include jump edges, crease edges and smooth edges [59]. Jump edges are defined as discontinuities in depth values, which are observed when an object is occluded by another one. Crease edges are characterized by discontinuities in surface normals, which are formed where two regions meet. Smooth edges are those with continuous surface normals but discontinuous curvatures. The second stage of these methods is to link these detected edges to form closed regions for segmentation [60, 61].

   The main weakness of these methods is that they cannot guarantee closed

boundaries for the complicated situations where some edge points may not be correctly detected. To tackle this issue, edge detection based on scan line approximation are proposed as in [62–64]. Scan-line based methods first project the range data into binary edge map along a given direction or scan line. For example, a scan line on 3D plane produces a 3D straight line. After projection, scan line segments are merged together based on some similarity measure in a region-growing fashion. Finally, some edge grouping techniques, such as minimum spanning tree, are applied to obtain closed contours for segmentation. The scan line method is mainly designed to extract planar surfaces.

Heath *et al.* compared some well-known edge detectors, such as Canny, Nalwa-Binford, Sarkar-Boyer and Sobel [65]. There are several criteria they used for comparison: For each edge detector, is it possible to choose a single set of optimal parameters for all the images without significantly affecting the performance? Does an edge detector produce edges of the same quality for all images or does the edge quality vary with input images? They found that the optimal parameter settings of an edge detector are strongly dependent on the input images, and that the relative performance of the edge detectors varied statistically significantly across the input images. This indicates the performance of the dataset segmentation might be affected by the parameter settings of its edge detector and there

is no optimal detector that can be used for any input data.

2. **Region based method:**

   The majority work on range image segmentation relied on region-based techniques [66–68]. These methods are relatively less sensitive to the noise in the data, and usually perform better when compared to edge-detection methods. First, seed regions which can be planar or non-planar surface patches are computed. Least squares adjustment and Hough transform are robust methods to detect planar seeds. The seed regions are then gradually growing to larger region by grouping points around them based on the given similarity measure, such as slope, curvature or surface normal.

   Region based approaches are essentially focusing on local features and their consistencies in the data. For example, Hernandez *et al.* extracted features from connected components based on Top-Hat of hole filling algorithm of range images [69]. Although these techniques bring acceptable results, they suffered from some common issues, such as the selection of the initial seed regions and the determination of the number of clusters. Some global features based on graph techniques are proposed to improve segmentation results. Graph cut in [70, 71] and minimum spanning tree in [72] are widely used graph based algorithms. The observation is that data points in the same segment are much more closely connected to each other compared with those points in other segments. The segmentation is

achieved by graph partitioning algorithms to find the optimized cuts that minimize the similarity between segments while maximizing the similarity within segments at the same time. Segmentation can be performed as recursive partitioning or direct multi-way partitioning as in [73].

Computational cost for region based methods is usually much higher compared with edge based methods. It is not really an issue when small range images are considered. However, this high-demanding of the computational resources prevent them from being used on large-scale point cloud datasets.

These proposed approaches on dataset segmentation are either not applicable to the building PCDs or computational expensive. We proposed an efficient segmentation approach for building PCD based on the observation that different parts of a building are usually separated by walls, ledgers etc. These *"separators"* provide clues for dataset segmentation. When projected onto 2D images, these separators have a common characteristic, that is a relative large *"dark"* region representing a salient feature in binary images. For example, the roof and the body are divided by a ledger as shown in Fig. 9(c). Fig. 9(b) and Fig. 9(d) shows different parts of the roof are separated by walls.

The dataset segmentation is carried out as follows. First, for a given point cloud dataset, we compute separators from all major directions obtained earlier, including bottom-up direction and directions perpendicular to major planes.

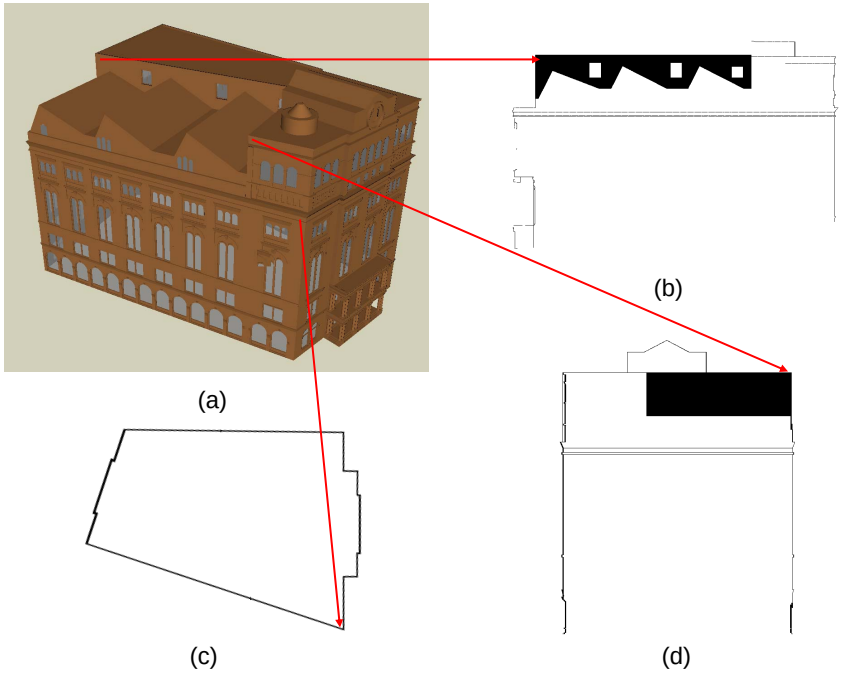


Figure 9: The dataset segmentation: (a) the original cooper union model; (b) the projected wall image from one face; (c) the projected ledger image; (d) the projected wall image from another face.

The next step is to split the original 3D dataset into segments based on the computed separators together with direction information. We will elaborate these steps in the following sections.

2.3.1 Separator Detection

We used a kernel based connected components (CC) method for computation. For each identified slice image I_i, the separator detector checks each data point P_i using a 5x5 kernel K centered at P_i. Let S_p be a set of points that have been visited by the separator detector. S_p is initialized to be empty. The point P_i is checked against S_p to determine whether it has been visited previously. If not, P_i is added into S_p and the data points $N = \{P_j | P_j \neq P_i, P_j \in K\}$ covered by the kernel K are recorded. If there are enough data points found in the neighbor, say $|N| > 12$, namely half of the kernel K, P_i is considered as qualified point for the connected component and is added to C. The same computation is applied to all new recorded data point in N, and qualified points are added into C. When there is no more new data point needs to be checked, the detector stops and a new connected component, C, is detected.

The next step is to compute the rectangle boundary of C, $\{x_{min}, x_{max}, y_{min}, y_{max}\}$. To check whether C represents a qualified separator, the following test is

conducted,

$$\begin{cases} |x_{max} - x_{min}| > T_{size} \\ |y_{max} - y_{min}| > T_{size} \end{cases}$$

where T_{size} is a threshold representing the minimum size of the separator, usually a value of 16 is good enough to rule out all non-separators. The above testing implies that a real separator CC should contain a big chunk of data and its width and height should be at least T_{size}. If a connected component C satisfies the above conditions, the slice index i together with the bounding information $x_{min}, x_{max}, y_{min}, y_{max}$, are logged down for further process.

For real dataset, separators might be detected in adjacent or neighboring sliced images as shown in Fig. 10 (a) - (c). This is usually caused by imperfect major plane detection or point cloud data registration introduced in earlier stages. To cope with this case automatically, we integrate all the neighboring sliced images with separators detected into a new image I as shown in Fig. 10(d). And the same separator detection algorithm is applied on I to obtain the bounding information. The index for this integrated image is set to be the mean of the indices of those neighboring images. After this automatic computation, we present the results to users and allow them to adjust the results manually. Essentially, users can adjust the index of the integrated image, which could avoid any potential errors for segmentation.

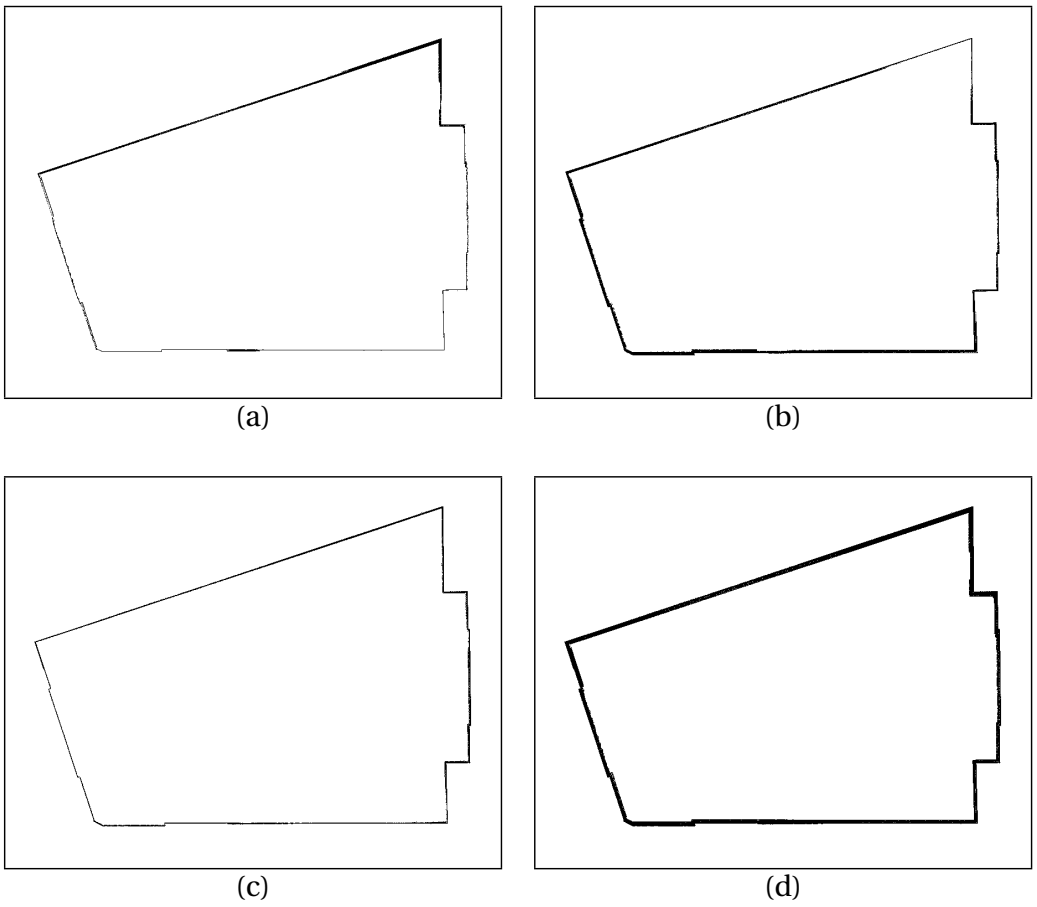


Figure 10: The adjacent slices with separators detected: (a) the first slice; (b) the second consecutive slice; (c) the third consecutive slice; (d) the merged slice from (a) to (c).

2.3.2 Segmentation Based on Separators

Once we have computed the separators based on the 2D sliced images extracted from all major directions, we can divide the original complex building structures into smaller and simpler modules for modeling. Based on the type of the input data we are about to segment, two approaches could be adopted for this partitioning process. First, the segmentation can be carried out on the original 3D point cloud dataset. That is, a series of smaller 3D point cloud datasets will be generated for each segment. However, for this approach, we have to transform each smaller segment and generate the projected 2D images for each major facades. This would impose redundant projection and the noise removal and hole filling would have to be applied again. Alternatively, we can apply the segmentation directly on the 2D sliced images generated previously, and therefore avoid those redundant transformation and projection computation required for the first approach.

To conduct the segmentation for the 2D sliced images, we need to transform the separators computed from all major directions into the common coordinate (with y-axis as the bottom-up direction). Because the separators detected in bottom-up direction are already in the common coordinate, no transformation is needed for these separators and the partitioning is straight-forward: we first sort the indices I of the bottom-up separators in an ascending order. The 2D slices projected along bottom-up directions are grouped based on their indices

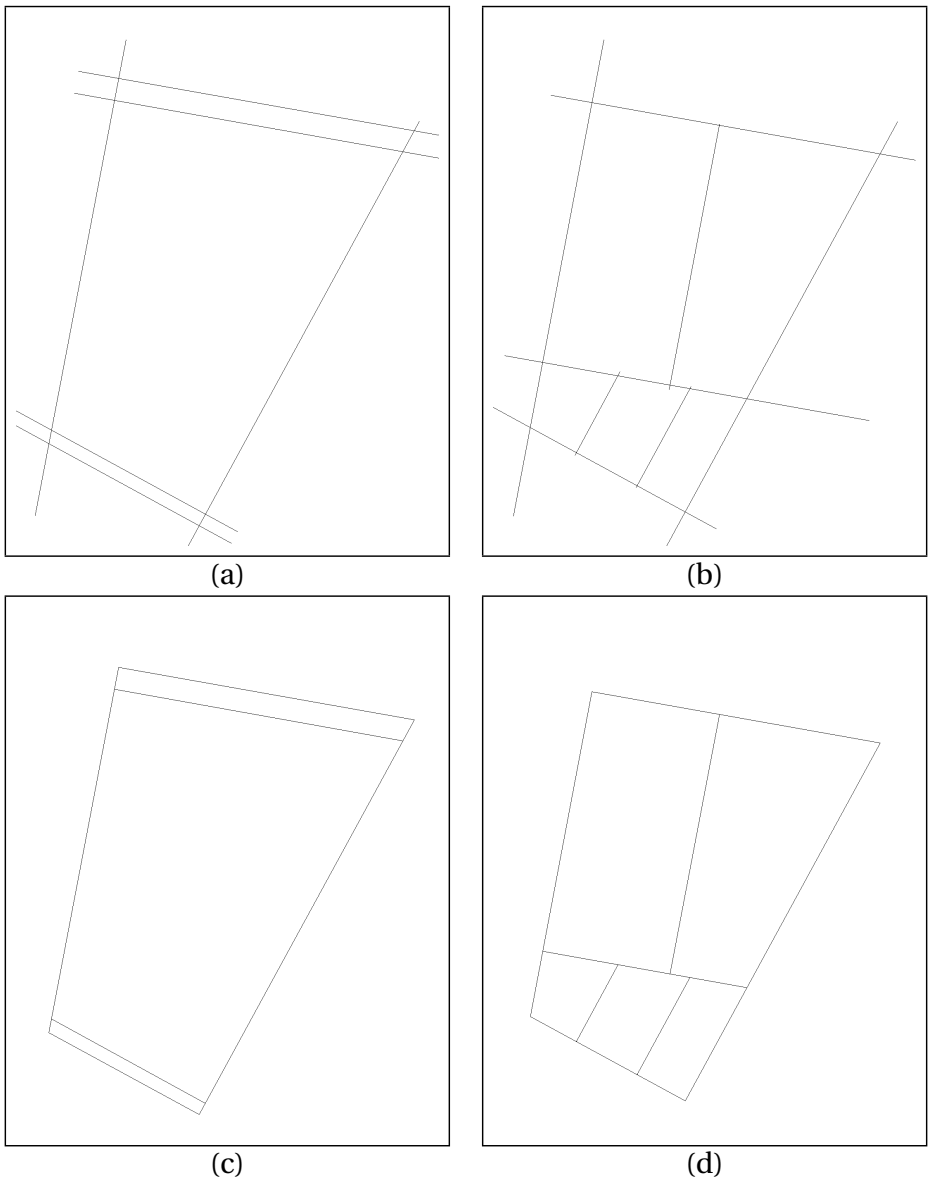


Figure 11: Segmentation region computation: (a) the transformed image with line segment of the separators for body; (b) the transformed image with line segment of the separators for roof; (c) the processed segment image regions for body; (d) the processed segment image regions for roof.

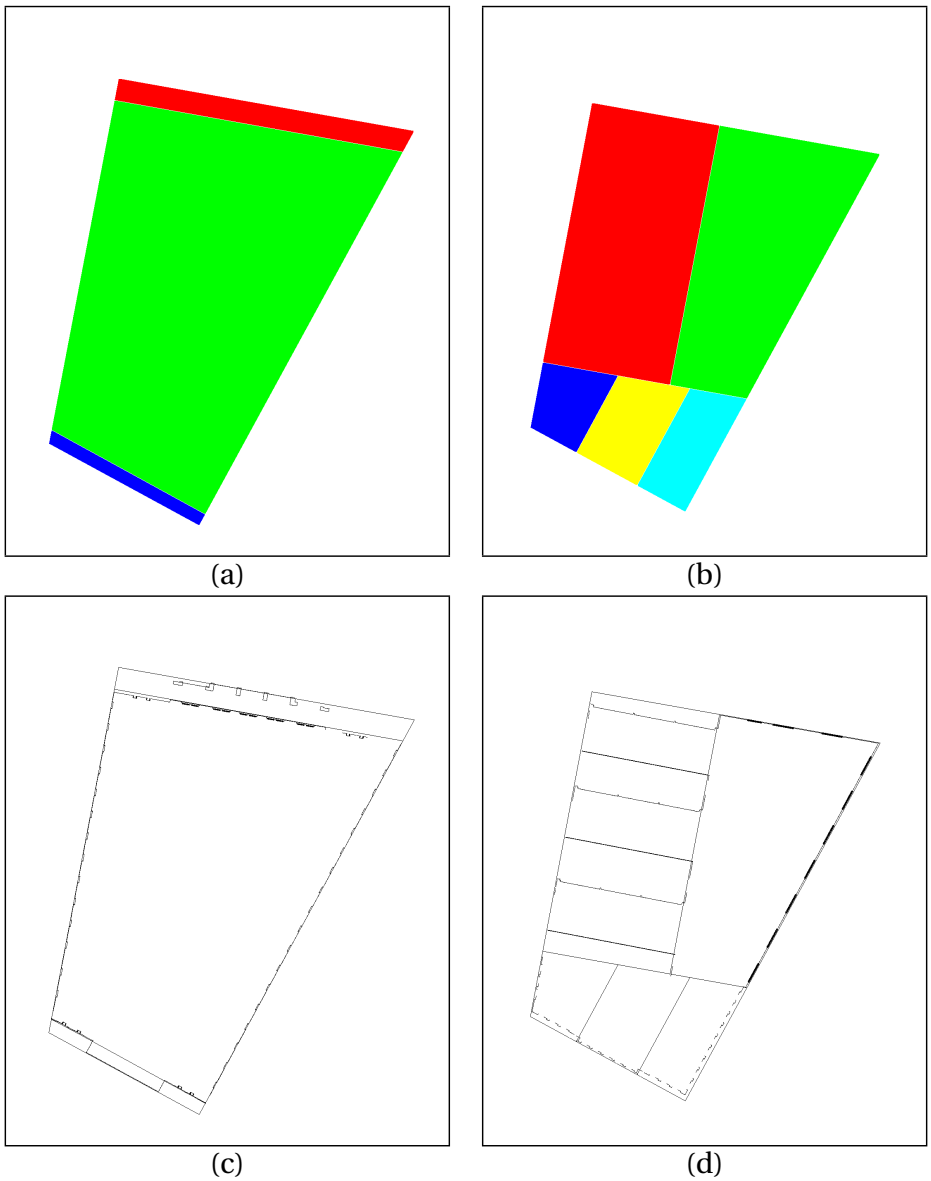


Figure 12: Segmentation region computation: (a) the highlighted regions for Fig. 11(c); (b) the highlighted regions for Fig. 11(d); (c) the superimposed image of Fig. 11(c) with a sliced image from body section; (d) the superimposed image of Fig. 11(d) with a sliced image from roof section.

against the sorted indices I. For example, the group with index range of $[I_i, I_{i+1})$, contains all 2D slices with indices greater than or equal to I_i and less than I_{i+1}. For the example of Fig. 3(c), The only detected separator (the ledger) as shown in Fig. 9(c) divides the whole dataset into top (roof structure) and bottom (body structure) parts.

The process becomes harder when we start the segmentation on the 2D slices in the same group of height section. First of all, all the separators detected in different directions need to be brought to the common coordinate. Each separator is transformed into a line segment in the bottom-up projected image with the end points representing the bounding positions. The roof and body projected images are shown in Fig. 11(a) and Fig. 11(b) respectively. To compute the boundary for each segment, the intersection points of each line segment with other line segments need to be computed. Given a line segment $L_i = P_0P_1$, represented by two end points P_0, P_1, we can compute the intersection points of L_i with all other line segments. If the computed intersection point P_i is falling outside of the image or is far away from either P_0 or P_1, P_i is regarded as an invalid intersection point and is skipped. When all the intersection points are checked, we can obtain two special ones, P_0' and P_1', which are the closest points to P_0 and P_1 respectively. P_0' and P_1' represent the starting and ending points of a wall or a facade of the underlying building. After intersection points are computed for all line segments, the segmentation image I_s can be obtained. Fig. 11(c) and

Fig. 11(d) illustrate the computed segmentation images for both body and roof sections.

With the segmentation images I_s, we can compute each segment region for 2D slices projected in bottom-up direction. To do this, we first build up a look-up table T which maps a pixel point (x,y) in I_s to a region, as shown in Fig. 12(a) and Fig. 12(b) for both body and roof. Different regions are marked in different colors and are assigned a unique region id. For a given point $P(x,y)$ in each sliced images, the region id for P can be obtained from the look-up table T. For example, for the roof sliced images, all red pixels are mapped to the same region id in Fig. 12(b). The same situation is applied to green, blue, and other pixels. After going through each point, a 2D sliced image from bottom-up direction can be divided into segments. The 2D sliced images for body and roof with segmentation regions superimposed are shown in Fig. 12(c) and Fig. 12(d), respectively. For those 2D slices projected from normals of other major planes, we carry out similar strategy for segmentation as that of bottom-up direction.

To demonstrate the segmentation results on 3D dataset, a similar process as 2D sliced image segmentation can be conducted. For each 3D data point $\mathbf{P}(x,y,z)$, we first compute its group in bottom-up direction by comparing $\mathbf{P}$'s y value (bottom-up direction) and those heights from range indices. Once the group is computed, with the segmentation image I_s, we can further compute the region id for $\mathbf{P}$ by obtaining its 2D projection point $P(x,y)$ in the image I_s.

The region id for P can be obtained from the look-up table T. For the example in Fig. 3(c), there are totally 5 regions are identified for data points of roof and 3 regions are located for body. Hence the original point cloud of the building is segmented into 8 segments as shown in Fig. 13, where different segments are labeled with different colors.

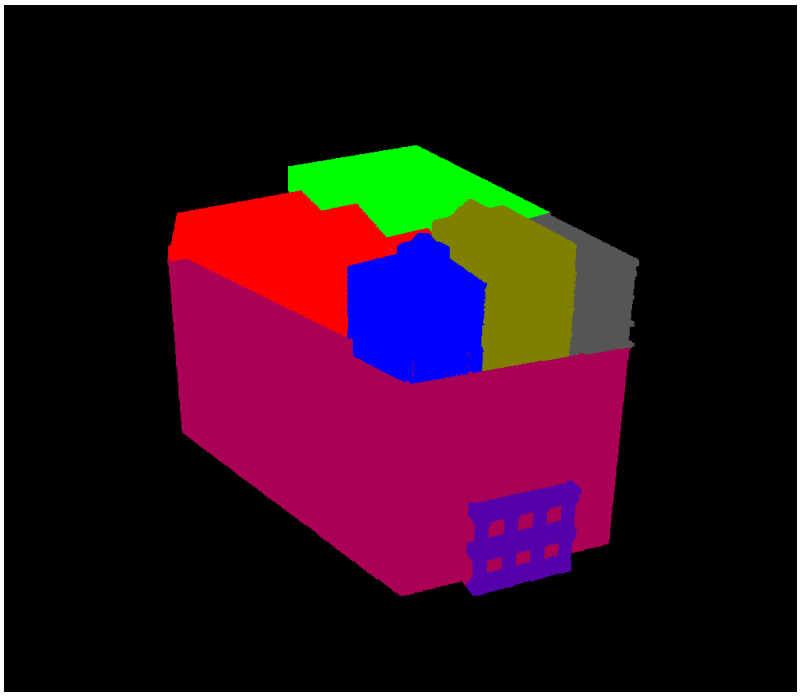


Figure 13: The segmentation result for the point cloud in Fig. 3(c).

Chapter 3

Keyslice Detection

Our 3D modeling algorithm is based on *a priori* knowledge that urban buildings can be created through a series of extrusion and tapering operations on the salient cross-sections contained in the keyslices. The key step for successful modeling is identifying these salient cross sections upon which the extrusion and tapering operations will apply.

Buildings generally are equipped with windows and doors which should not be considered as salient feature for keyslices. Therefore, data points corresponding to windows and doors should be discarded to avoid possible side-effects during the keyslice computation.

3.1 Window Detection

Windows and doors are important features for buildings to be modeled. Moreover, accurate computation of the extrusion structures depends on these information. Without knowing the marked location as window part, extra keyslices may be computed and hence lead to excessive extrusion operations. Image-based window detection has been widely conducted in [74, 75]. Essentially, the 2D window regions are extracted by exploiting the properties of building structures, such as shape and symmetry. The estimation of the depth for the extracted 2D windows is computed by using matches for the linear features within the extracted window in two or more ground views. However, the estimation is not reliable due to perspective projection. Some window detection methods on 3D data have also been proposed in [34, 76]. A constrained surface fitting based algorithm is proposed to fit parametric models of doors on point cloud in [77]. This method assumes that the data have been segmented and requires relatively high density data around the window area. Pu and Vosselman use a triangulation-based method to detect the boundaries of sparse regions within a building facade and then fit rectangles to the resulting region to compute windows [78]. However, this method needs to improve its accuracy when background noise exists.

The information we want to compute for windows and doors includes both the location and the depth. The depth is computed based on the observation that

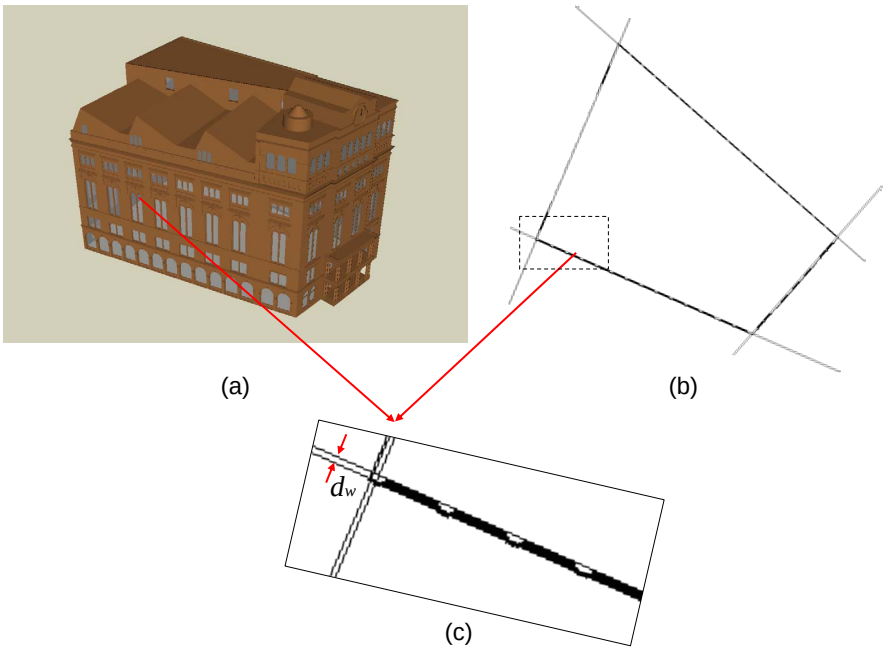


Figure 14: The depth computation of the windows/doors: (a) the window on the original model; (b) the projected slice extracted from bottom-up direction; (c) the close-up view of the parallel lines and the distance d_w.

two parallel lines can be detected when viewing from the bottom-up direction as shown in Fig. 14(c) . For two parallel lines $L_1 : y = mx + c_1$ and $L_2 : y = mx + c_2$, the distance d_w is computed with the following equation,

$$d_w = \frac{|c_1 - c_2|}{\sqrt{1+m^2}}$$

Once the distance d_w is computed, we can obtain the windows/doors image by projecting the data points between L_1 and L_2 onto a slice image as shown in Fig. 15(a). There are multiple window structures in the same projected slice image. We can use the same strategy as separator detection described in Sec. 2.3.1, that is, kernel based connected component method, to isolate each window for processing. Fig. 15(b) shows a window extracted as a connected component and BPA algorithm was applied to obtain the contour in red. After walking through the image shown in Fig. 15(a) for all the connected components and applying BPA on them, the final results show all the detected window contours as depicted in Fig. 15(c). These contours will be used to compute window mask images and project windows and doors onto the final model.

The above window detection is for synthetic dataset since clean result can be obtained. For real dataset, due to noise and missing data, the computed window images are usually not able to be processed using the above method and manual efforts are needed to assist the computation. Fig. 16(a)-(c) show the consecutive

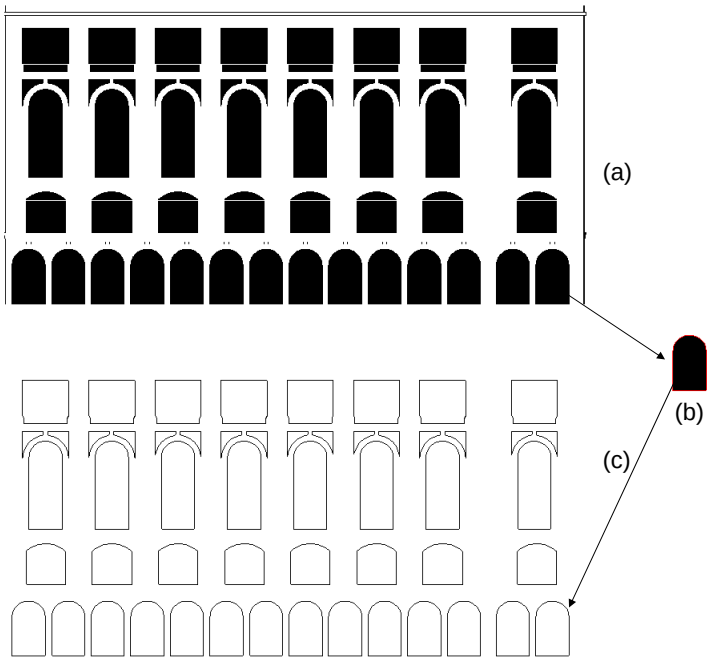


Figure 15: The window information: (a)the projected windows slice; (b) the most bottom-right window extracted from (a) with BPA contour in red; (c) the contours of windows computed from (a).

projected images with window structures. Fig. 16(d) shows the integrated image from slice (a) to (c). As we can see, some data are missing around the window section. We have to enhance the image by manually marking the locations of these window structures as shown in Fig. 16(e). With the assistance of user interaction, we can obtain the final window structure as shown in Fig. 16(f).

3.1.1 Windows Mask Image Generation

Windows and doors detection have been discussed in Sec. 3.1. If there is no windows or doors structures detected in the current PCD, this step is skipped. However, when windows or doors are detected, the data points corresponding to these structures in cross-section images should not be taken into account for keyslices computation. This is to ensure that no keyslices are generated due to curve or any geometry shapes introduced by windows or doors.

To generate the mask images, the information about windows and doors, including width, height, depth are needed, which has been computed previously. Let $\{x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}\}$ be the bounding box w_{box} of a window w, where x, y, z are corresponding to width, height and depth directions. The w_{box} is transformed to bottom-up direction, with x, z defining the rectangle location of the window, and with y defining the height range for w.

After computing the mask information for all windows, the mask images can be generated by fusing all the window information. Fig. 17 shows couple of mask

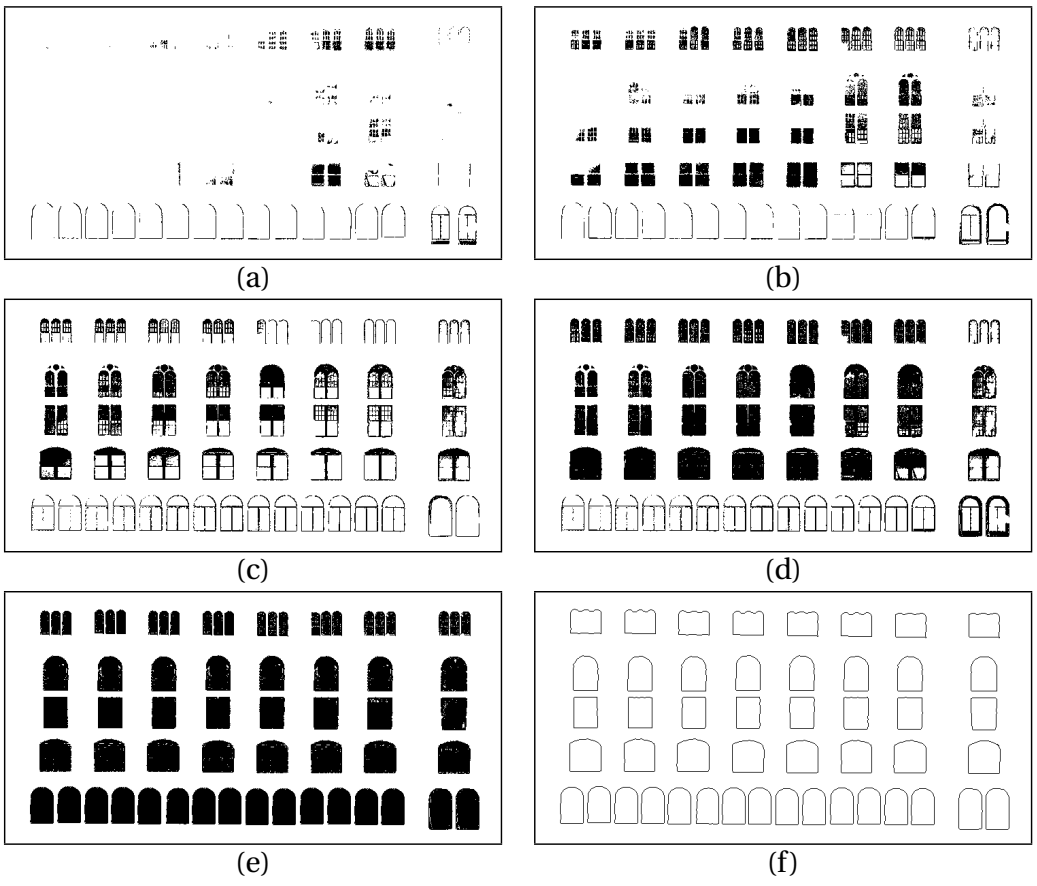


Figure 16: Window detection for real dataset: (a) - (c) are consecutive slices; (d) integrated slice from (a) to (c); (e) manually enhanced image for (d); (f) window structure computation from (e).

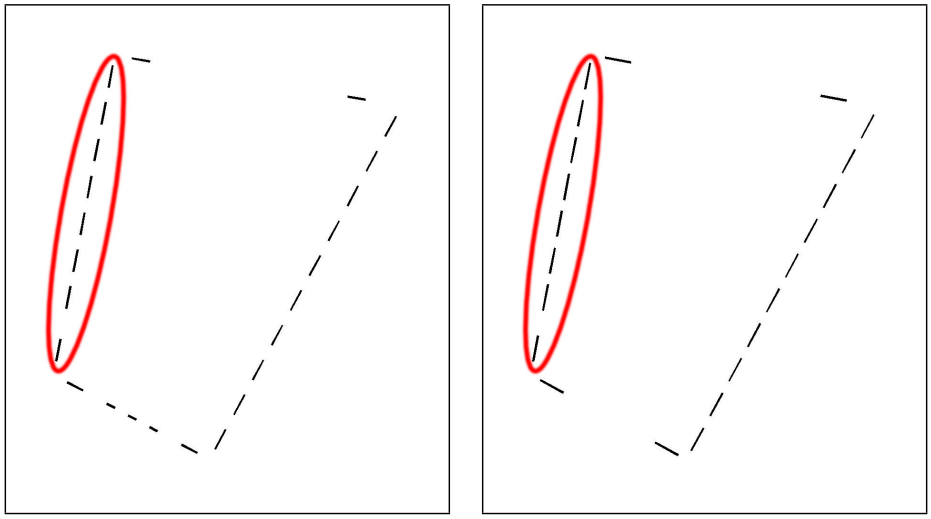


Figure 17: The window mask images: (a) the mask image corresponding to height range of bottom windows; (b) the mask image corresponding to height range of upper windows.

image examples for different height range of the cross-section slices. The regions circled in red are corresponding to the locations of the windows in Fig. 15(a) in different height range.

Once the mask images are obtained, a simple bit-and operation with the original 2D slice image eliminates the data points corresponding to windows or doors. That is, $I_n = I \& I_w$, where I is a 2D slice, I_w is the mask image, and I_n is the updated 2D slice with window or door data points removed. The updated I_n slices are used for keyslice detection.

3.2 Keyslice Detection With Distance Measurement

An intuitive way for keyslice detection is to compute the similarity between the sliced images. In other words, the sliced images are clustered into different groups based on the similarity among them. Similarity measure has been extensively studied in the image registration research community [79, 80]. Recently, Zitova and Flusser conducted an survey on image registration and they identified two major research methods for similarity measure of 2D images, i.e., area based methods and feature based methods [81]. Area based methods are also called correlation-like methods or template matching methods, which compute similarity of images without attempting to detect salient features or objects. The

feature based methods, on the other hand, aim to find the pairwise correspondence of features between two images based on spatial relations or various descriptors of features represented by control points, such as end points or centers of line features, centroid of regions, etc.

The area based methods are computational efficient but they usually can only be applied on binary or gray-scale images. The feature based ones are computational complex but more powerful and can be applied to images obtained using different sensors, such as multi-modal image registration. Because our 2D sliced images for similarity measure are pure binary images, and are projected from the same 3D range data, the area based methods are more efficient and appropriate.

For area based methods, usually a predefined window or even the entire image are used for the similarity measure. The classical measurement for area based method is the cross correlation as presented in [82]. However, this method becomes very time consuming when the size of window increases in order to capture the global relations between images. To address this issue, the phase correlation methods based on the Fourier Shift Theorem in frequency domain are proposed. The phase correlation was originally proposed for the registration of translated images. It can be combined with log-polar mapping of the spectral magnitude to handle rotation and scale transform of images. Wolberg and Zokai [83] showed such combination power to register affinely distorted images.

They used a variation of Levenberg-Marquardt nonlinear least squares optimization method for iterative estimation of perspective deformation. In addition to correlation methods, the mutual information (MI) methods, originating from the information theory, are proposed to measure statistical dependency between two images. MI methods are particularly suitable for images acquired from different modalities. Viola and Wells [84] applied MI on magnetic resonance images and employed the coarse-to-fine speed up using the gradient descent optimization method.

Although these approaches are very powerful, they are time consuming and not suitable for our scenario. Huttenlocher *et al.* [85] proposed Hausdorff distance (HD) based similarity measure for binary images. This method is computationally efficient and can handle images with perturbed pixel locations, which outperforms the correlation methods. We adopted a light-weighted global and efficient key image detection approach based on distance function similar to Hausdorff distance as the similarity measure. Let $P_r(x_r, y_r)$ be a data point in a reference image and let $P_i(x_i, y_i)$ be a data point in a new observed image I. The distance of image I to reference image I_r is defined as:

$$d_H(I, I_r) = \sum_{i=0}^{N} d_{min}(P_i, I_r) \tag{7}$$

where $d_{min}(P_i, I_r)$ is the minimum distance from data point P_i in image I to the

reference image I_r. Alternatively, we can also define the distance, $d_H(I_r, I)$, from reference image I_r to a new observed image I, using Eq. (7). These two distances are usually not equal to each other. As a rule of thumb, one can choose $d_{HD} = \text{MAX}\{d_H(I, I_r), d_H(I_r, I)\}$ as the distance function. To compute the keyslices, a threshold τ_d is used for d_{HD}. If $d_{HD} < \tau_d$, the two images I and I_r are considered similar to each other. Otherwise, a keyslice image is found and I_r is updated with I, the new keyslice image.

3.3 Keyslice Detection With Curvature

The accuracy of the keyslices detected by the distance measurement is closely tied to threshold τ_d. Small τ_d leads to more accurate models and will require more time and space to compute and store the results. When the threshold τ_d is relatively large, potential keyslices which contain salient structures may be missed. Therefore, there is a trade-off between model accuracy and time-space efficiency.

We can improve the model accuracy by computing curvature information as a complementary criteria for keyslice detection. The idea is based on the observation that the keyslices are generally located at large curvature changes along 2D slices extracted in the orthogonal direction (e.g., side view), as shown in Fig. 18(a). Therefore, instead of computing the difference between two images

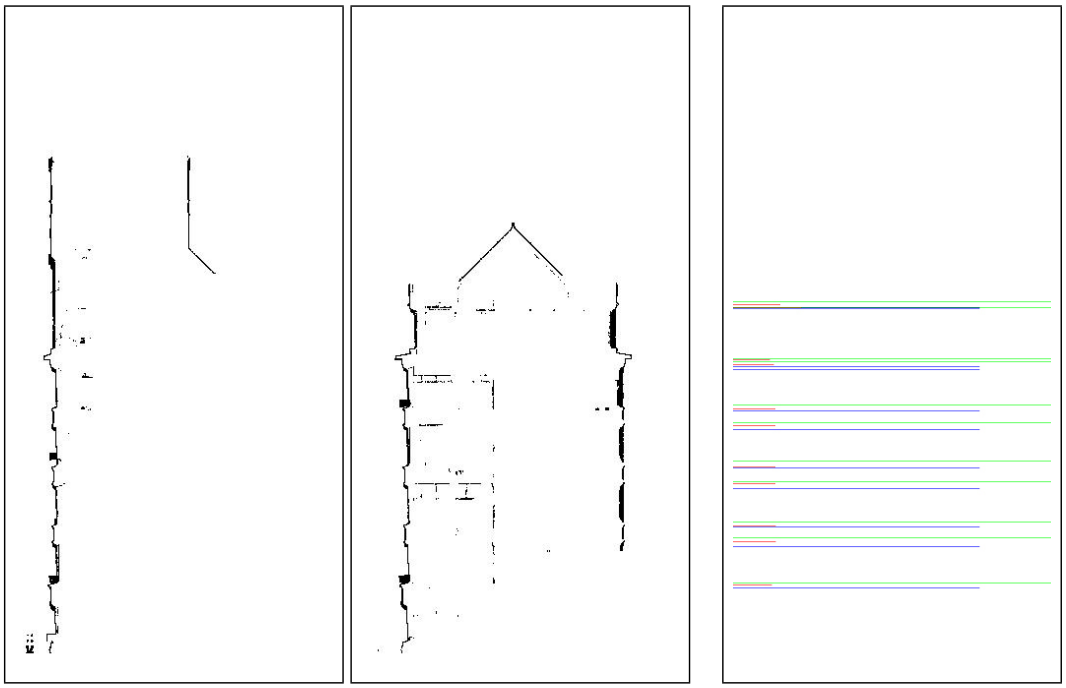


Figure 18: Curvature-based key slice detection: (a) a partial set of two 2D sliced images from the orthogonal direction (side view). The complete set will be used to extract the keyslices shown in Fig. 20. (b) the average curvatures detected over all of the sliced images along the orthogonal direction.

directly, we compute the curvatures of orthogonal 2D slices, map the positions of curvature extrema back to cross-sections in the original set of volumetric slabs, and mark these cross-sections as keyslices.

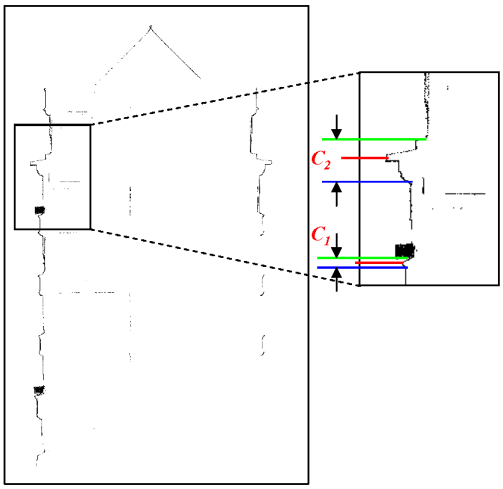


Figure 19: Curvature based key slice detection.

In the close-up view of a small region as shown in Fig. 19, two curvatures, c_1 and c_2, are computed in this small region. The red, blue and green lines indicate the locations of the center, the starting and the ending of a curvature. As a matter of fact, there is a third curvature computed around the black box sitting on top of the green line of the curvature c_1. However, this third one should not be considered as a real curvature in that the black box is due to an air conditioner during the data acquisition process, which means it is not a part of the building.

The third curvature, or *outlier curvature*, needs to be removed from the set of real curvatures. Based on the fact that these outlier curvatures exist only in a few 2D slices, we can exclude them by counting the number of appearance for each curvature. Only those curvatures appears in most of the 2D slices are kept for further computation.

Once the real curvatures are obtained, they can be used as a complemental way of distance measurement for keyslice detection. Each curvature c is mapped to a set s of original 2D cross-sectional slices, $i,\ldots,j$, where i and j are corresponding to the starting and the ending locations of c. After this, the set s is checked whether it contains any keyslice or not. If a keyslice k has been marked in s by distance measurement, nothing needs to be done and this curvature c is discarded. On the other hand, if s contains no keyslice, i and j will be marked as two new keyslices to keep the salient feature identified by this curvature c. The combination of distance measurement and curvature inference ensures that the salient structures of a building will be preserved. Fig. 20 depicts the keyslices derived from the uniform slices given in Fig. 6.

3.4 Algorithm Analysis of Keyslice Detection

The proposed keyslice detection is summarized in Algorithm 1. The algorithm consists of two independent functions. The function *Distance()* conducts

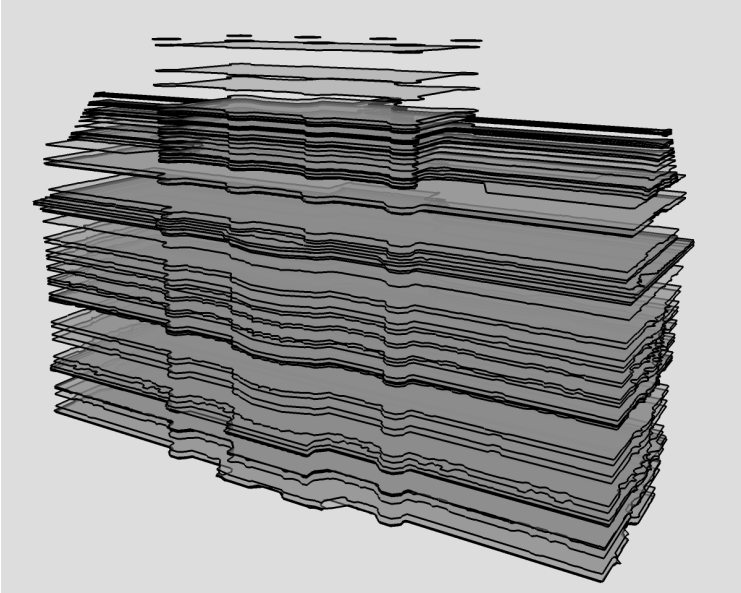


Figure 20: Keyslices derived from the input in Fig. 6.

distance measurement based keyslice detection. And the function *Curvature()* carries out the curvature based keyslice detection. A lookup table t is computed in the function *lookup()*. This lookup table is used to store for each pixel p, the distance to the nearest data point in the reference image I_r or I_i. If p itself is a data point, the distance is 0. The complexity of constructing t is bounded by $O(n)$, where $n = w * h$, and w and h are the image width and height respectively. The function *dis()* is basically doing hashing computation on t, which takes only $O(1)$ run time. The function *avg()* computes the average of d_1 and d_2, i.e., $(d_1 + d_2)/2$, which takes $O(1)$ time complexity. The function *append()* appends a new index value i to the linked list L, which takes $O(1)$ time complexity too. Because there are total K sliced images, and K is a constant value, the complexity of the function *Distance()* which composes of the above functions takes $O(K * n) = O(n)$ for the computation.

In the procedure *Curvature()*, the function *curv()* which computes curvatures in the orthogonal direction is of $O(n)$ complexity. For each orthogonal sliced image, the *mapping()* function maps all computed curvatures to corresponding indices of sliced images based on the locations of the curvatures and increase their counters by 1. Since each *curv()* computation takes $O(n)$, the iteration process takes $O(K * n) = O(n)$ time complexity. Overall, the algorithm *KSDA()* takes $O(n)$ computation complexity for keyslice detection.

For the space complexity, the keyslice detection algorithm also takes $O(n)$

Algorithm 1 The Keyslice Detection Algorithm

procedure DISTANCE(I, K)
  $L \leftarrow \emptyset$ ▷ vector to store the indexes of keyslices
  $I_r \leftarrow I_0$
  for $i \leftarrow 1, K-1$ **do**
    $t \leftarrow lookup(I_r); d_1 \leftarrow dis(I_r, I_i, t)$
    $t \leftarrow lookup(I_i); d_2 \leftarrow dis(I_i, I_r, t)$
    $d_{HD} \leftarrow avg(d_1, d_2)$ ▷ average distance
    if $d_{HD} > \epsilon$ **then**
      $I_r \leftarrow I_i$ ▷ update the reference image
      $append(L, i)$
    end if
  end for
  return L
end procedure
procedure CURVATURE(I, K)
  $L \leftarrow \emptyset$ ▷ vector to store the indexes of keyslices
  $V \leftarrow 0$ ▷ initialize the counter to 0
  for $i \leftarrow 0, K-1$ **do**
    $\bigcup C \leftarrow curv(I_i)$ ▷ curvature computation on image I_i
    $\bigcup c \leftarrow mapping(\bigcup C)$ ▷ mapping the curvature to indexes
    for each $c \in \bigcup c$ **do**
      $V(c) \leftarrow V(c)+1$ ▷ increase the number of curvature observed
    end for
  end for
  for each $c \in \bigcup c$ **do**
    if $V(c) > \epsilon$ **then** ▷ validate the curvatures
      $append(L, c)$
    end if
  end for
  return L
end procedure
procedure KSDA(I, K)
  $L_1 \leftarrow Distance(I, K)$ ▷ keyslices computed by distance
  $L_2 \leftarrow Curvature(I, K)$ ▷ keyslices computed by Curvature
  $L \leftarrow merge(L_1, L_2)$
end procedure

complexity for the computation. Again n is the total number of pixels for the 2D image. This is due to the lookup table in *Distance()* function. Also, the function *Curvature()* uses $O(n)$ memory for curvature computation which is based on ball-pivot algorithm introduced in Algorithm 3.

Chapter 4

Boundary Vectorization

After the keyslices are detected, N_K keyslices will be identified from a total of N_A image slices. Depending on the threshold τ_d, N_K is usually about one to two orders of magnitude smaller than N_A, e.g., N_K/N_A is 0.06 when $\tau_d = 4.0$ for the example in Fig. 4(b). To generate the 3D model, these keyslice images need to be vectorized to represent the contours of the building facade. The difficulty of the problem lies in outliers and holes of non-perfect images. We proposed a general framework based on an adaptive 2D ball-pivot algorithm (BPA) to efficiently suppress the noisy, fill holes and generate contours for those images.

4.1 Ball Pivot Algorithm

Raster image vectorization has been an active research topic and some classical methods are widely used in digital image processing for years, such as those described in [86–89]. However, these approaches only consider the cases where perfect raster image data is provided. For non-perfect data, these proposed methods were failed in various cases. The problem considered here is to compute contours for non-perfect binary images which may contain holes along the boundary as well as noise or outliers inside the boundary. Here, *boundary* refers to the out-most region of any raster image and *contour* refers to a vectorized boundary.

The Douglas-Peucker (DP) algorithm [90] is widely used to compute boundary vectorization for binary images. Although the complexity of this approach is $O(nlogn)$ after the improvement of implementation described in [91, 92], this method cannot fill the holes of the noisy data or handle the case where spurious interior points are presented. Agarwal *et al.* considered the problem of approximating a polygon chain using another one to minimize the number of vertices [93]. However, for a raster image, the vertices defining the contour were not known. For example, Fig. 21(a) shows a binary cross-section image of a 3D point cloud data of a building. Fig. 21(b) shows the contour generated from DP algorithm. The problem is that there are a lot of holes along the contour and the noise inside of the boundary are modeled inappropriately. Medeiros *et al.* [94]

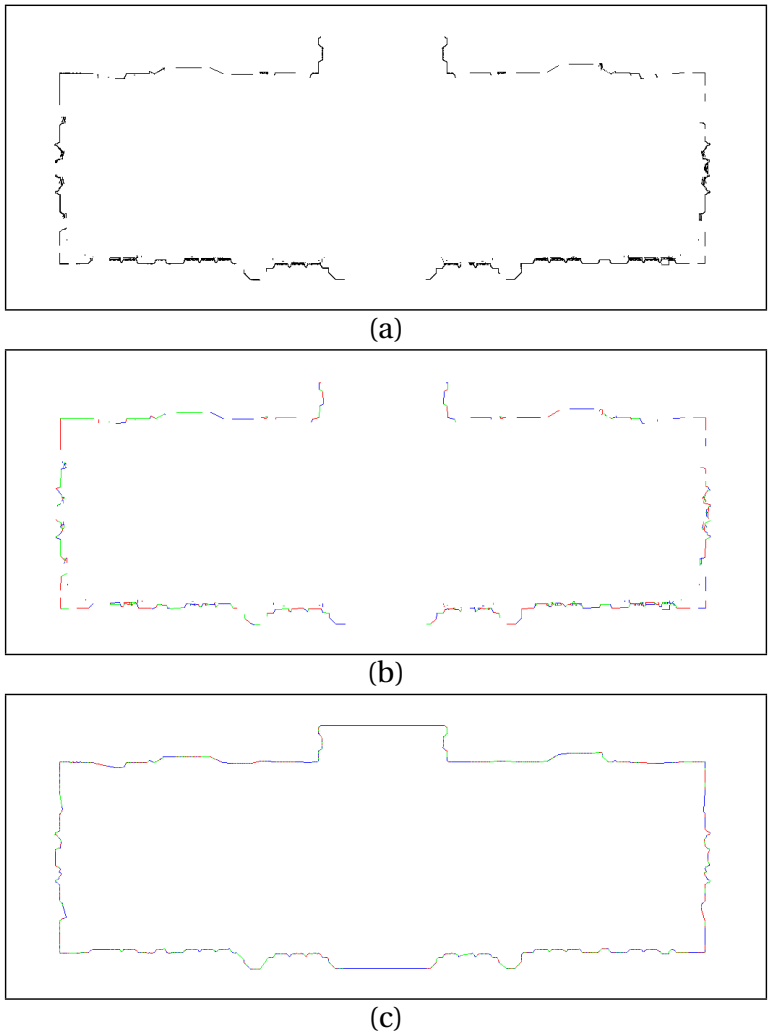


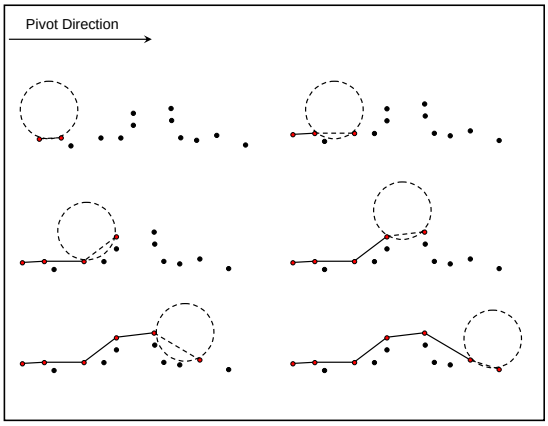
Figure 21: Binary image vectorization: (a) the example of binary image; (b) the vectorization result based on DP algorithm; (c) the vectorization result from proposed BPA algorithm.

applied ball-pivot algorithm (BPA) [30], which was original proposed on 3D point cloud data, on the 2D image to obtain vectorized contour. The 2D BPA algorithm works well and efficiently on computing contours with appropriate parameters. The key parameter for BPA to work successfully is the size of the ball for pivoting. We have proposed an adaptive BPA algorithm to address this problem. Fig. 21(c) shows the contour computed by the proposed method which fills holes between gaps and suppresses noise inside the boundary.

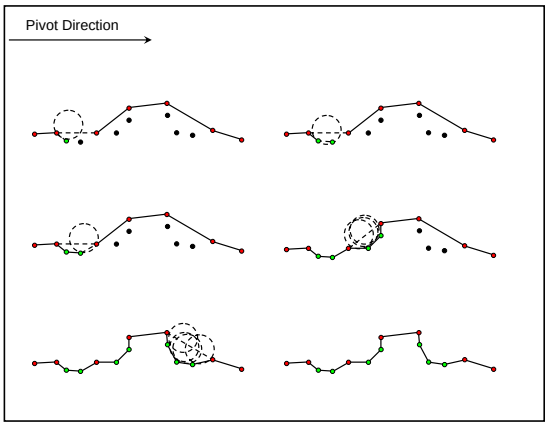
The original Ball Pivot Algorithm (BPA) was proposed to efficiently generate 3D mesh representation on 3D point cloud data. The idea of the BPA is straightforward: pivot a ball on a starting point in the image until it touches another data point as depicted in Fig. 22(a). Add the new touched data point into an ordered list as the vertices of the contour polygon and set it as the new pivoting point. Keep doing this pivoting process until all data points are touched.

4.2 Boundary Vectorization

The basic idea behind the proposed framework for contour computation is as follows: apply BPA on a selected starting point until either the ball pivots back to the starting point which indicates a closed boundary is detected, or the ball reaches a gap which means this contour is a non-closed contour and one of the



(a)



(b)

Figure 22: Adaptive ball pivoting algorithm: (a) initial pivoting with a ball of radius $2r$; (b) refinement with a ball of radius r.

end points is reached. Because the ball can start pivoting along either clock-wise direction or counter clock-wise direction, another pivoting process at start pivoting from the alternative direction is conducted. When both directions have been explored, a contour computation is done. After this, a *refinement* process is carried out for each line segments using the same BPA algorithm. For this refinement, the starting point is now the first end point of a line and the stopping case is that the ball reaches the other point or it reaches a gap as depicted in Fig. 22(b).

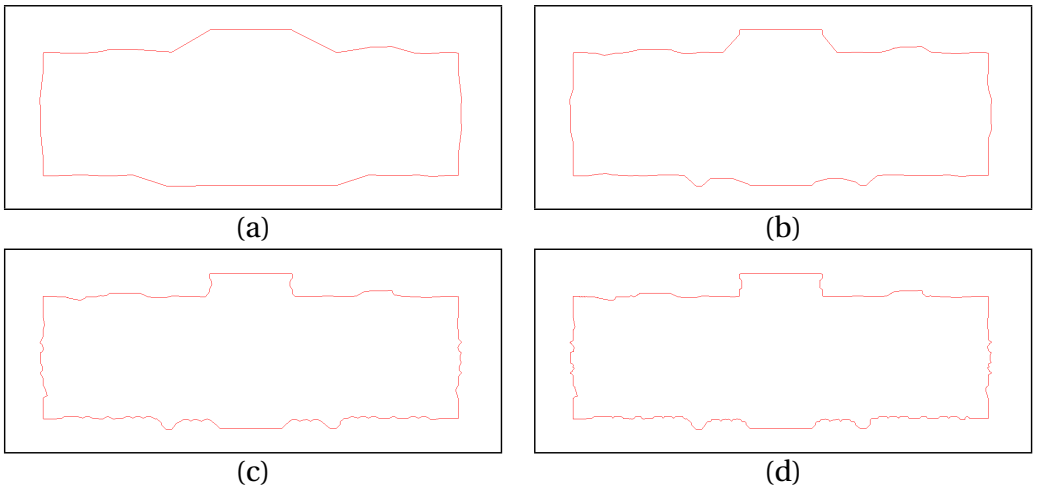


Figure 23: Boundary vectorization of a binary image with (a) radius = 128 (b) radius = 32 (c) radius = 8 (d) radius = 2.

Here is more detailed description for the algorithm. At the initial BPA stage, a relatively large radius r is chosen as a coarse step to cover all gaps between data

points. The output of the initial BPA, $\mathbf{\Phi}$, contains an ordered list of the boundary data points $\boldsymbol{P}$ and their corresponding directions $\overrightarrow{\boldsymbol{R}}$ in which the ball C starts pivoting. The BPA refinement process applies a smaller radius, say $r' = r/2$, to $\mathbf{\Phi}$ to get more accurate results, as shown in Fig. 22(b). For this stage, the length of each line segment formed by adjacent points is checked, $\ell = \overline{P_0 P_1}$, in $\mathbf{\Phi}$. For long line segments, the BPA is applied between the two adjacent points. When the ball reaches the second end point, a new list of ordered boundary points, $\mathbf{\Phi'}$, is inserted into $\mathbf{\Phi}$ between P_0 and P_1. This process continues until it finishes checking every adjacent point in $\mathbf{\Phi}$. The refinement algorithm is an iterative process, and the radius of the pivot ball is reduced for each new iteration. It stops when r' falls below a threshold τ_r.

The key parameter for the BPA algorithm to work successfully is to find a good initial size of the ball for pivoting. Here are some general guidances for choosing the radius. If a contour is known to be a closed polygon, the initial radius should be big, e.g. the width of an image, to cover all gaps along the boundary. Otherwise, if a boundary consists of sub-boundaries, one could select a relative small radius to start.

An example on the proposed framework is shown in Fig. 23. The initial BPA takes radius $\tau_r = 128$, which produced a coarse contour as shown in Fig. 23(a). Fig. 23(b) - Fig. 23(d) show that the contour is becoming more accurate as the radius τ_r decreases from 32 to 2. Note that the original image size is 1024x392

pixels as shown in Fig. 21(a).

4.3 Contour Simplification

Although the proposed framework is an efficient and straightforward approach to vectorize the contours of noisy images, it produces many short line segments which can be merged. In particular, when the underlying boundary is a straight line structure, the BPA produces excessive un-necessary short lines due to its way of generating the contour. An example is shown in the upper part of the contour in Fig. 24(b) where each colored segment represents a contour edge. One way to reduce the number of vertices of the polygon is to apply the approximation polygon method in [93]. The drawback of this method is that the topological structures, such as straight lines, would be lost. To solve this issue, an adaptive Hough transform (AHT) based method is proposed to replace short line segments with long lines and potentially eliminate noisy or outlier vertices around the boundary. The pseudo-code of the adaptive Hough transform is shown in Algorithm 2.

To combine the adaptive BPA with AHT, we have to apply AHT on binary images to obtain straight lines. The straightforward choice for the input binary image is the original sliced raster image. However, the issue is that some spurious lines may be detected because of noise or non-boundary data points, such as

those inside the contour of the raster image as shown in Fig. 24(a). Therefore, a better choice is to use the binary image generated using the line segments from BPA. This will solve the spurious line issue because there is only boundary data points associated with these line segments. These *clear* images can be generated by drawing lines on blank images using the information of these line segments.

The next step is to apply the HT algorithm on the clear image I to obtain all straight lines $\boldsymbol{L}$ and sort them by length. The longer lines give higher confidence to the line structure of the underlying images. Then a dilation operation [95] with 8-connected neighbors on I can be used to get the dilation image, I_d. The details on mathematical morphology are elaborated in Appendix B for self-containing purpose. We can measure how well the lines in $\boldsymbol{L}$ match with the data points in I_d, which determines whether a line in $\boldsymbol{L}$ should be used as a substitution or not.

If a line segment L is found to be a good candidate, the corresponding part of the BPA points in $\boldsymbol{\Phi}$ is to be located for substitution. This is done by first computing the closest two points P_i and P_j in $\boldsymbol{\Phi}$ to the two end points of L. If the vertices in $\boldsymbol{\Phi}$ represent a polygon, they will have a closed region layout, i.e., $\boldsymbol{P} = \{P_0, P_1, \ldots, P_{n-1}, P_0\}$. Assuming $i < j$, there are two possible choices to replace the series of the points: $\boldsymbol{P_1} = \{P_i, P_{i+1}, \ldots, P_{j-1}, P_j\}$, or $\boldsymbol{P_2} = \{P_j, P_{j+1}, \ldots, P_{i-1}, P_i\}$. To determine which choice is an appropriate one, one can compare the distance, D, from the line L to both set of the points $\boldsymbol{P_1}$ and $\boldsymbol{P_2}$. The point set with smaller D

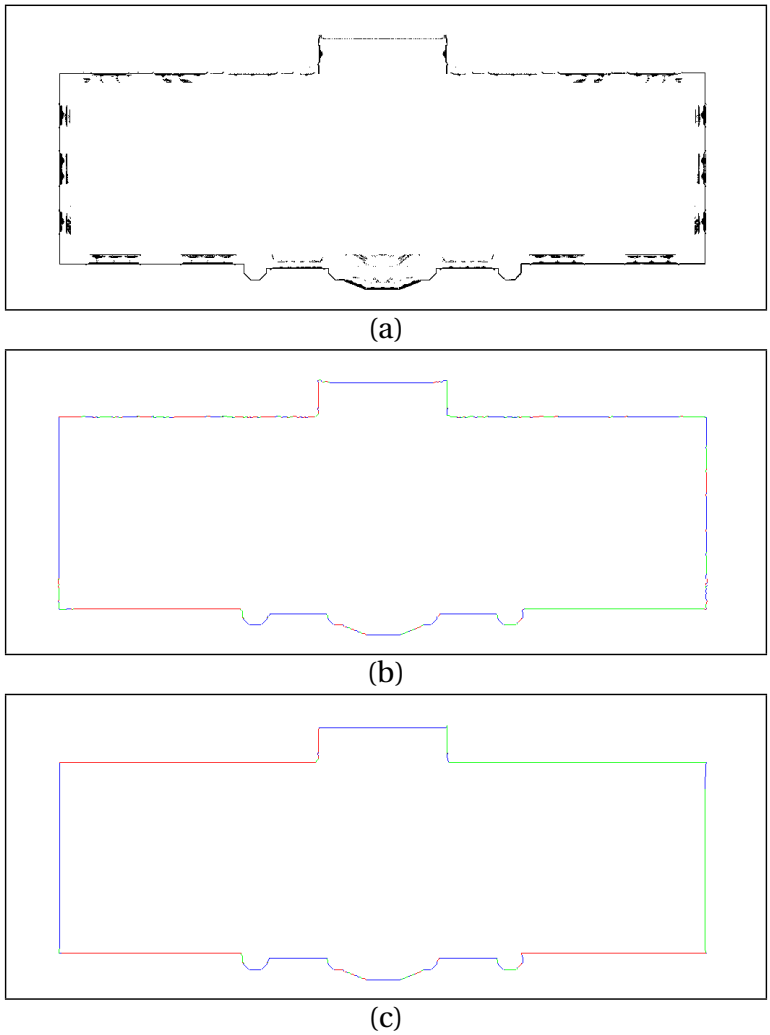


Figure 24: Boundary vectorization of noisy binary image: (a) the binary image to be processed; (b) the contour computed by proposed method; (c) the contour obtained by simplification with Hough transform.

is the one to be substituted by the line L.

$$D = \underset{\boldsymbol{P_1}, \boldsymbol{P_2}}{\arg\min} \sum \|P_i - L\| \qquad P_i \in \boldsymbol{P_1} \text{ or } P_i \in \boldsymbol{P_2}$$

where $\|P_i - L\|$ is the Euclidean distance from point P_i to line L.

After the integration of BPA contour with the Hough transform lines, the beautified contour is shown in Fig. 24(c). Notice that the top part of the contour, which consists of short line segments, was replaced with two long line segments. As one can see, this process reduces the noise, simplifies the contour, and produces clean results for contours with straight line structures.

In addition to the results shown in Fig. 21 and Fig. 24, more experimental results are shown in Fig. 25. In the first row of Fig. 25, the input image is of size 1024x392 and contains multiple contours. In the second row, the input image is 800x640 and contains nested contours. In the third row, the input image is 800x640 and contains a curved contour. As demonstrated, the proposed framework can handle all the cases properly and fill the holes as expected, which outperformed the standard DP algorithm.

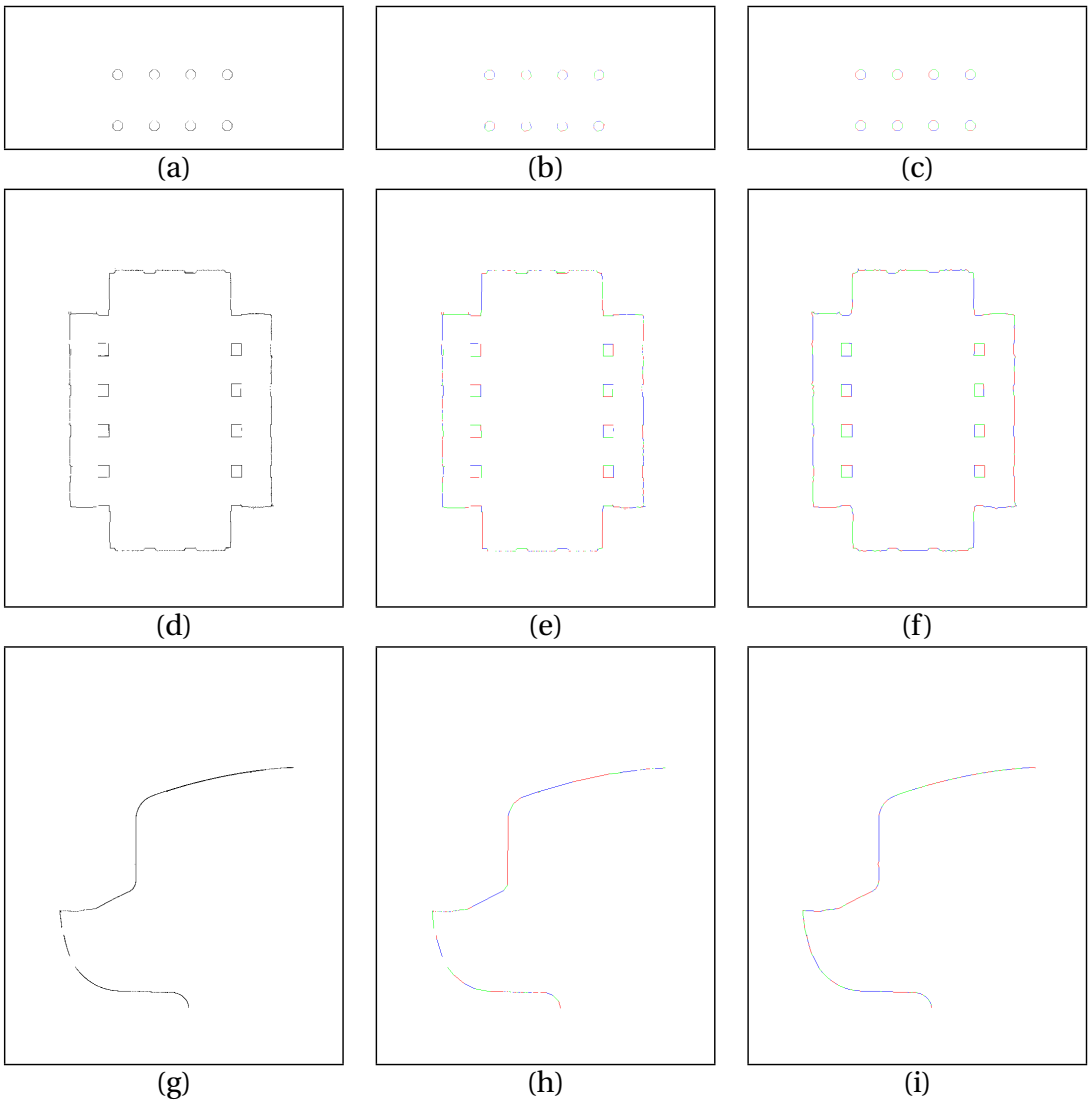


Figure 25: More experimental results: (a), (d) and (g) show the original noisy binary images; (b), (e) and (h) show the vectorization results from DP algorithm; (c), (f) and (i) show the vectorization results from proposed algorithm.

4.4 Algorithm Analysis of Boundary Vectorization

4.4.1 Algorithm Analysis of Adaptive Hough Transform

The Adaptive Hough Transform (AHT) algorithm stems from the standard Hough transform algorithm. The function *HT()* is a regular Hough Transform (HT), which returns the lines detected on the image. The computation complexity of HT is $O(n^{m-1})$, where n is the size of 2D image and m is the number of parameters [96]. In case of 2D line detection, $m=2$, and therefore the complexity is $O(n)$. The computation of the number of points covered by each line (ρ,θ) detected by *HT()* is of $O(n)$ complexity. The image update function *update()* remove data points around the longest line detected in each iteration, which is of $O(n)$ complexity. Because we are dealing with bounded number of data points, and the data points on the longest line of each iteration are removed for each iteration, the AHT will quickly converge.

The space complexity of AHT algorithm is bounded by the space complexity of the *HT()* algorithm. Because we are only working on line detection of 2D images, the space complexity is $O(n_\rho * n_\theta)$, where n_ρ and n_θ represents the number of bins for quantizing the value of ρ and θ respectively. For example, for a image of size 300x400, if we quantize the length ρ with one bin for each pixel, the n_ρ would be 500. Also, if we quantize the the angle with one bin for a degree, the n_θ would be 180. Based on this, the space complexity for this example can be easily

Algorithm 2 The Adaptive Hough Transform Algorithm

procedure AHTA(I, $\boldsymbol{L}$)
  while $true$ **do**
    $\bigcup(\rho,\theta) \leftarrow HT(I)$ ▷ regular hough transform to get a set of lines
    $n \leftarrow 0$
    for each $(\rho,\theta) \in \bigcup(\rho,\theta)$ **do**
      $c \leftarrow num(I,\rho,\theta)$ ▷ compute # of points around line (ρ,θ)
      if $n < c$ **then**
        $n \leftarrow c$
        $\rho_{max} \leftarrow \rho$
        $\theta_{max} \leftarrow \theta$
      end if
    end for
    if $n > \epsilon$ **then** ▷ sanity check for the line $(\rho_{max},\theta_{max})$
      $\boldsymbol{L} \leftarrow (\rho_{max},\theta_{max})$
      $update(I,\rho_{max},\theta_{max})$
    else
      $break$
    end if
    if $\omega(I) < \epsilon$ **then** ▷ check how many points left
      $break$
    end if
  end while
end procedure

obtained. The precise of the Hough transform algorithm is depend on the quantization of the parameters ρ and θ. The more bins are used for quantization, the more accurate the results will be, which implies more space are needed for the algorithm.

4.4.2 Algorithm Analysis of Ball-Pivot Algorithm

The 2D adaptive ball-pivot algorithm (ABPA) is summarized in Algorithm 3. The seed computation *Seed()* is to pick up a good starting point, which is only $O(1)$ complexity. The *append()* function is to concatenate the control points, which is only $O(1)$ complexity. The regular 2D ball-pivot algorithm, *BPA()*, carries out the geometry computation on each control point based on the size of the radius of the ball or circle. Basically, for each pivoting of the ball, the area in the image covered by this pivoting is checked. If there is a data point contained by this area, the pivoting ends for this control point. The worst case for computing the whole potential pivoting area is $O(K*n)$, where n is the total number of pixels on an image I and K is the number of pivoting for each point. Because the ball is pivoting for a fixed angle, δ, K is bounded by the value $2\pi/\delta$. The complexity to pick up the minimum angle of the pivoting is $O(1)$, therefore, the complexity for the whole 2D *BPA()* is $O(n)$.

For the refinement process, each line $\overline{P_iP_j}$ of the boundary generated in the previous iteration is checked using the same algorithm in *BPA()*. This only takes

Algorithm 3 The 2D Adaptive Ball-Pivot Algorithm

procedure ABPA(I, $\boldsymbol{L}$)
  $L \leftarrow \emptyset$
  $P, P_0 \leftarrow Seed(I)$ ▷ compute the *seed* point assigned to P and P_0
  $r \leftarrow W$ ▷ initialize radius r with a large value W
  while $true$ **do** ▷ initial BPA stage
    $append(L, P)$
    $P_i \leftarrow BPA(P, I, r)$ ▷ regular 2D BPA
    if $P_i = P_0$ **then**
      $break$ ▷ complete the initial BPA
    end if
    $P \leftarrow P_i$ ▷ find a new vertex P for the contour L
  end while
  $r \leftarrow W'$ ▷ a smaller radius W' for refinement
  while $r > \epsilon$ **do** ▷ iterative BPA refinement
    for each line $\overline{P_i P_j} \in \boldsymbol{L}$ **do**
      $L' \leftarrow \emptyset$
      $P \leftarrow P_i$
      while $true$ **do**
        $append(L', P)$ ▷ construct a sub contour L'
        $P_k \leftarrow BPA(P, I, r)$
        if $P_k = P_j$ **then** ▷ stop when reaching the other point
          $substitute(L, P_i, P_j, L')$ ▷ refine $\overline{P_i P_j}$ with L'
          $break$
        else if $isGap(L', P_k)$ **then** ▷ stop when reaching a gap
          $break$
        end if
        $P \leftarrow P_k$ ▷ find a new vertex P for L'
      end while
    end for
    $r \leftarrow r/2$ ▷ reduce the radius for next iteration
  end while
end procedure

$O(n)$ complexity. The *substitute()* and *isGap()* function insert an extra point and carry out a simple algebra computation to check whether a gap is encountered. The complexity for both operations is $O(1)$. Because the round of the refinement process is bounded by a predefined constant number, say C, the complexity of this refinement is also bounded by $O(C*n)=O(n)$.

The implementation of the ABPA uses $O(n)$ memory space, where n is the size of the 2D image. This complexity includes the enqueue/dequeue the control points of the boundary, the pivoting geometry computation and the *substitute()* and *isGap()* computations. Since the user can control the size of slices, memory requirements can be tailored to the available hardware.

Chapter 5

Tapered Structure Detection

After the keyslices are detected and vectorized, the contours of $\mathbf{N_K} = \{I_i | i = 0,...,K\}$ keyslices are used to represent the building based on the extrusion operation. That is, the space between each pair of keyslices, say I_i and I_j, can be interpolated by the lower keyslice, e.g., I_i in this case. This is valid because of the similarity between the intermediate slices and the keyslice I_i. By modeling a building using this series of keyslices $\mathbf{N_K}$, we significantly reduce the number of polygons for urban buildings. An example of the reconstructed model is shown in Fig. 26. This helps make possible of 3D web-based applications, such as 3D city navigation.

In addition to the extrusion operation, we can potentially further improve the model and reduce the model size based on the observation that part of the

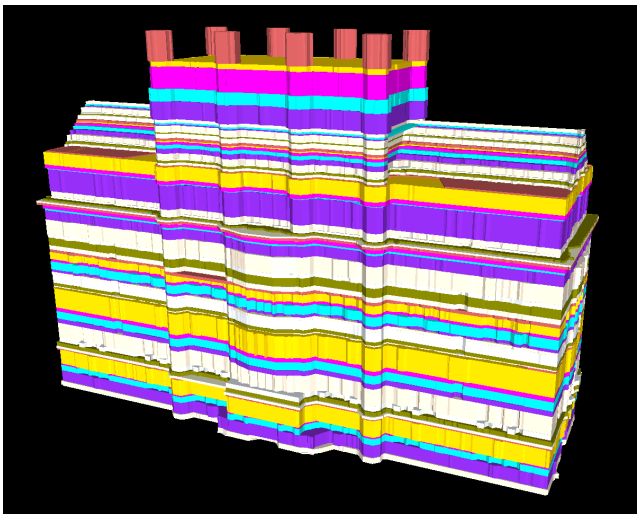
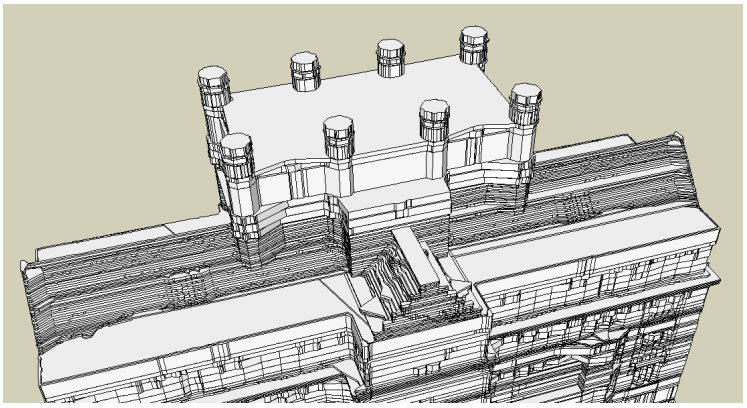


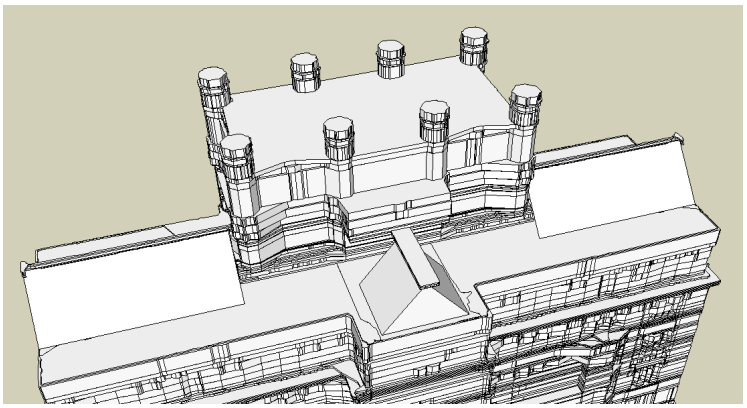
Figure 26: The reconstructed model based on extrusion operation on keyslices.

keyslice images belong to the same tapered structure. Fig. 27(a) shows the roof structure of the reconstructed model from a keyslice image extrusion operation with almost half of the keyslices dedicated to the tapered structure. After inferring the tapered structure, Fig. 27(b) shows the improvement of the modeling, which is much smoother than the previous model. In addition, the keyslices needed to represent the building, and its associated storage, are reduced in half.

The difficulty in inferring tapered structures is tied to the complexity of a building structure itself. Although the majority of building tapered structures are *linear tapering*, such as *tapering to point* (TTP), a cone shape geometry, as illustrated in Fig. 28(a), and *tapering to line* (TTL), a wedge shape geometry, as



(a)



(b)

Figure 27: The top view of the 3D building shown (a) without tapered structures and (b) with tapered structures.

shown in Fig. 28(b), it could also be a complicated *non-linear tapering*, such as a dome shape geometry which is depicted in Fig. 28(c). Furthermore, a structure may look like a tapered structure, but it is actually not a real one. For example, a series of small extruded structures form a tapering-like shape in Fig. 28(d).

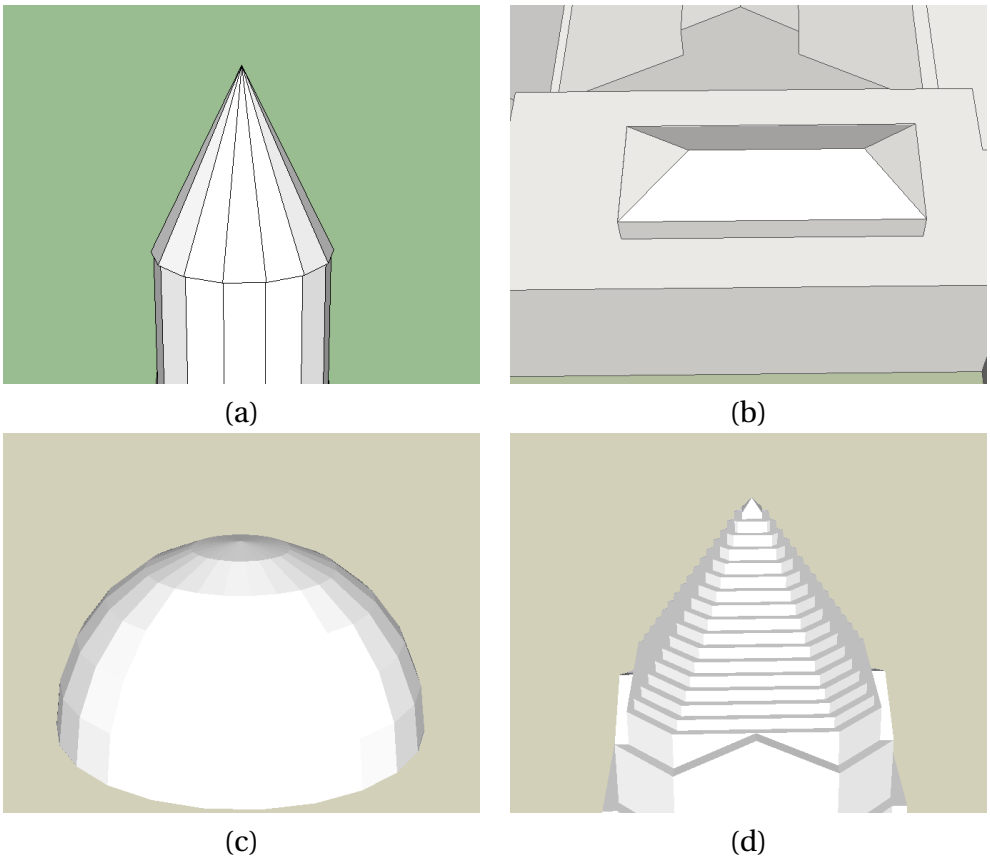


Figure 28: The component of (a) tapering to a point (b) tapering to a line (c) non-linear tapering (d) non-tapering.

Although modeling of tapered structure on building volumetric data has not been exploited much, a more generic topic which is the modeling of deformable primitives has being an active research area for a long time. Numerous work have been proposed by researchers on modeling deformation primitives [97–99]. The candidate set of primitives include geons, superquadrics and generalized cylinders, etc. In [97], Leonardis proposed the "recover and select" paradigm, which simultaneously fits many different models to the data and dynamically selects the most promising candidates while filtering out those that match the data poorly. This work deals with the unsegmented data directly and is time consuming due to possible multiple primitives existing in the dataset. The capability of dealing with multiple primitives is not necessary because our input data has been segmented as described in Sec. 2.3.

Biederman proposed to use geons to represent cuboids and cylinders, as well as curved and tapering versions of these base shapes in his theory of Recognition by Components [100]. Generalized cylinders extend the concept of a cylinder by allowing the cross-sectional shape to change as a function of position along the axis [101]. Also superquadrics are proposed to represent spheres, cylinders, cuboids, octahedrons, and interpolations between these shapes [102, 103]. Zhou and Kambhamettu extended the basic superquadrics to allow more complex shapes that bend or taper [104]. Essentially, all those models are fit to 3D data by searching for the parameters that best align the models' surface with

the data. However, the large number of parameters in these representations and potential for local minima in the optimization function presents challenges for these data fitting problems.

For building modeling, the major tapered structures are those linear tapering to a line or tapering to a point structures. Our intention is to rule out non-tapered structures as well as those complicated non-linear tapered structures which we do not want to handle for the sake of simplicity and efficiency. We only want to infer the relative simple tapered structures, including linear tapering to a point and tapering to a line. If the underlying structure is one of the above shapes, its keyslices can be substituted with the representation of these shapes. Otherwise, we can choose to model this special structure by fitting a triangular mesh to the underlying 3D point cloud to produce a polygonal model.

5.1 Inferring Linear Tapered Structure

We first check whether it is necessary to do tapering detection by examining the keyslices computed from all major sweeping directions. If the segment under analysis can be interpreted by a few keyslices from certain sweeping direction, we mark this segment as extrusion structure and therefore skip the tapering detection for it. Otherwise, the tapering detection will be conducted in standard bottom-up direction. For example, Fig. 29(a) shows a snapshot of the

point cloud data of a special tapering to line structure. Fig. 29 (b) - (d) show the keyslices computed from the three principle axes represented in blue, green and red. This model can be regarded as an extrusion structure since it can be interpreted by an extrusion operation on a single keyslice from left-right direction as shown in Fig. 29(d). However, the tapering detection has to be conducted on the models shown in Fig. 30, Fig. 31 and Fig. 32 since they are not able to be interpreted as simple extrusion operation on a few keyslices.

We proposed a two-step workflow to accomplish the tapering detection. The first step is to locate the potential tapering keyslices and infer the structure by making an assumption that the underlying tapered structure is either a TTP or a TTL. With this assumption, we check the last keyslice image of the potential tapered structure and analyze whether the whole structure is a TTP or a TTL. Unfortunately, the tapered structure under analyzing may not belong to these shapes. Therefore, a verification step is carried out to check the correctness of the inferred shape by measuring the error between the model and the corresponding *sampled* 3D point cloud data. By computing only sampled data points, we can largely reduce the computation requirement for large-scale PCD. If the average error is small, the inferred shape is confirmed and the underlying keyslices are substituted. Otherwise, we can choose to model this special structure by meshing the underlying 3D point clouds to produce polygonal model or simply by the tapering keyslices.

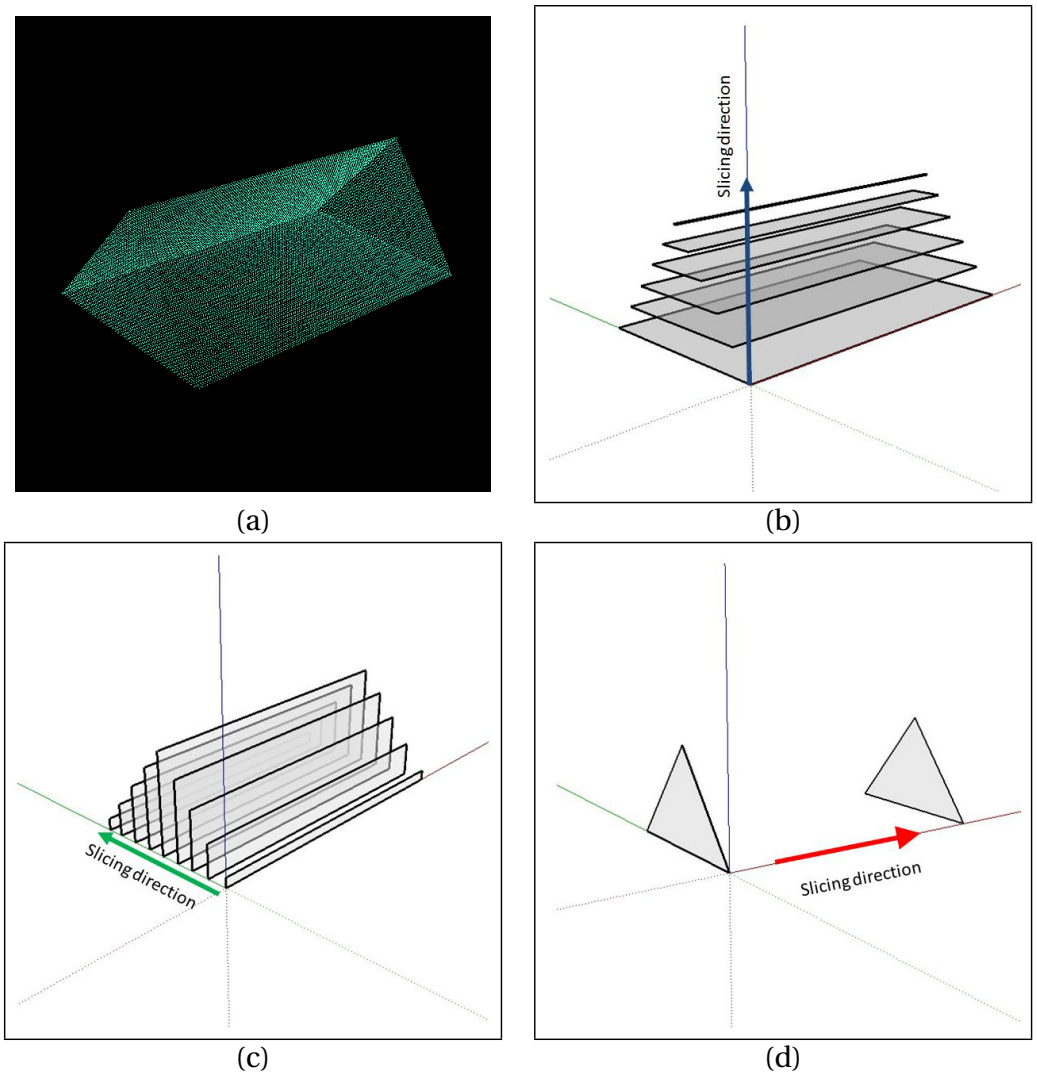


Figure 29: The modeling of a special tapering to line structure: (a) the snapshot of the point cloud data; (b) keyslices computed from bottom-up direction; (c) keyslices computed from face-inside direction; (d) keyslices computed from left-right direction.

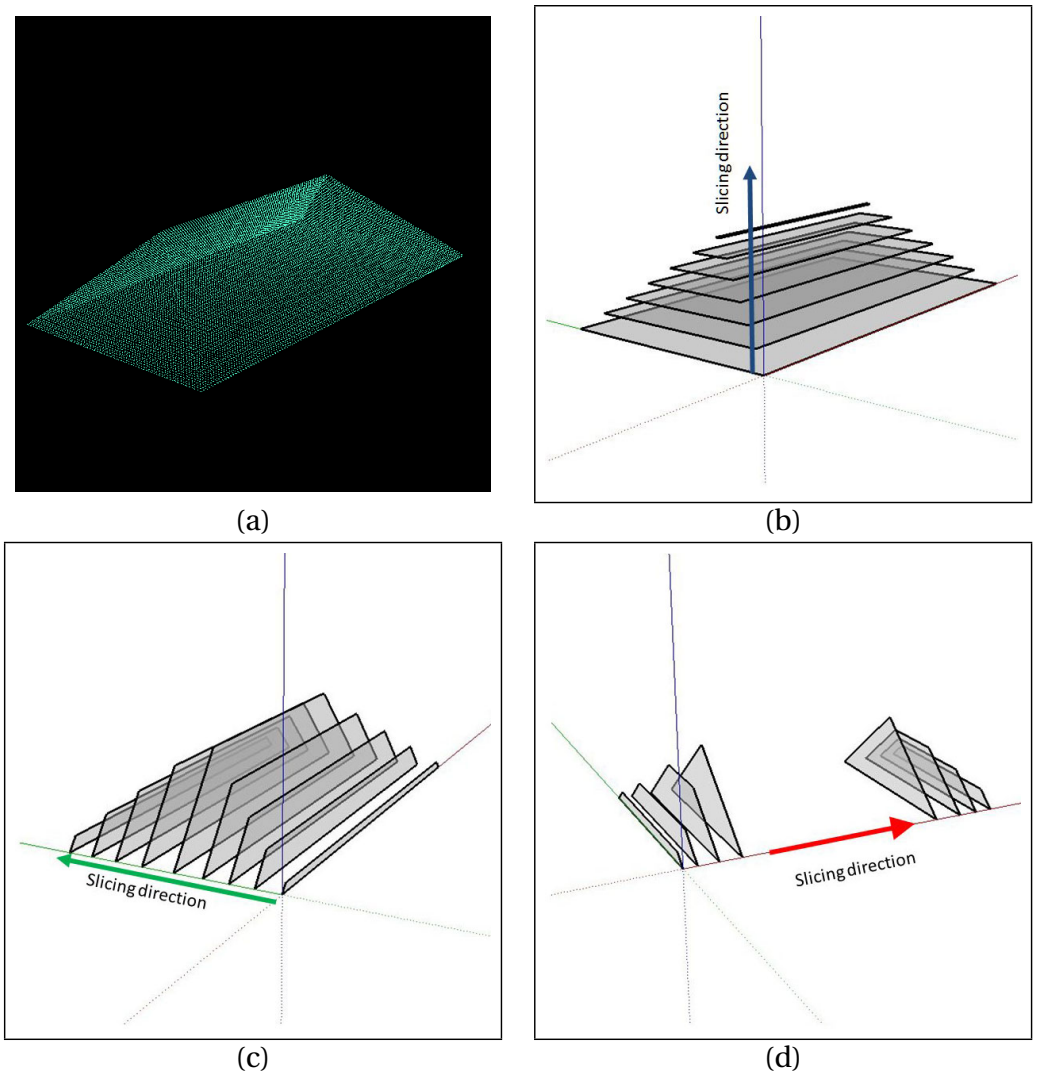


Figure 30: The modeling of a regular tapering to line structure: (a) the snapshot of the point cloud data; (b) keyslices computed from bottom-up direction; (c) keyslices computed from face-inside direction; (d) keyslices computed from left-right direction.

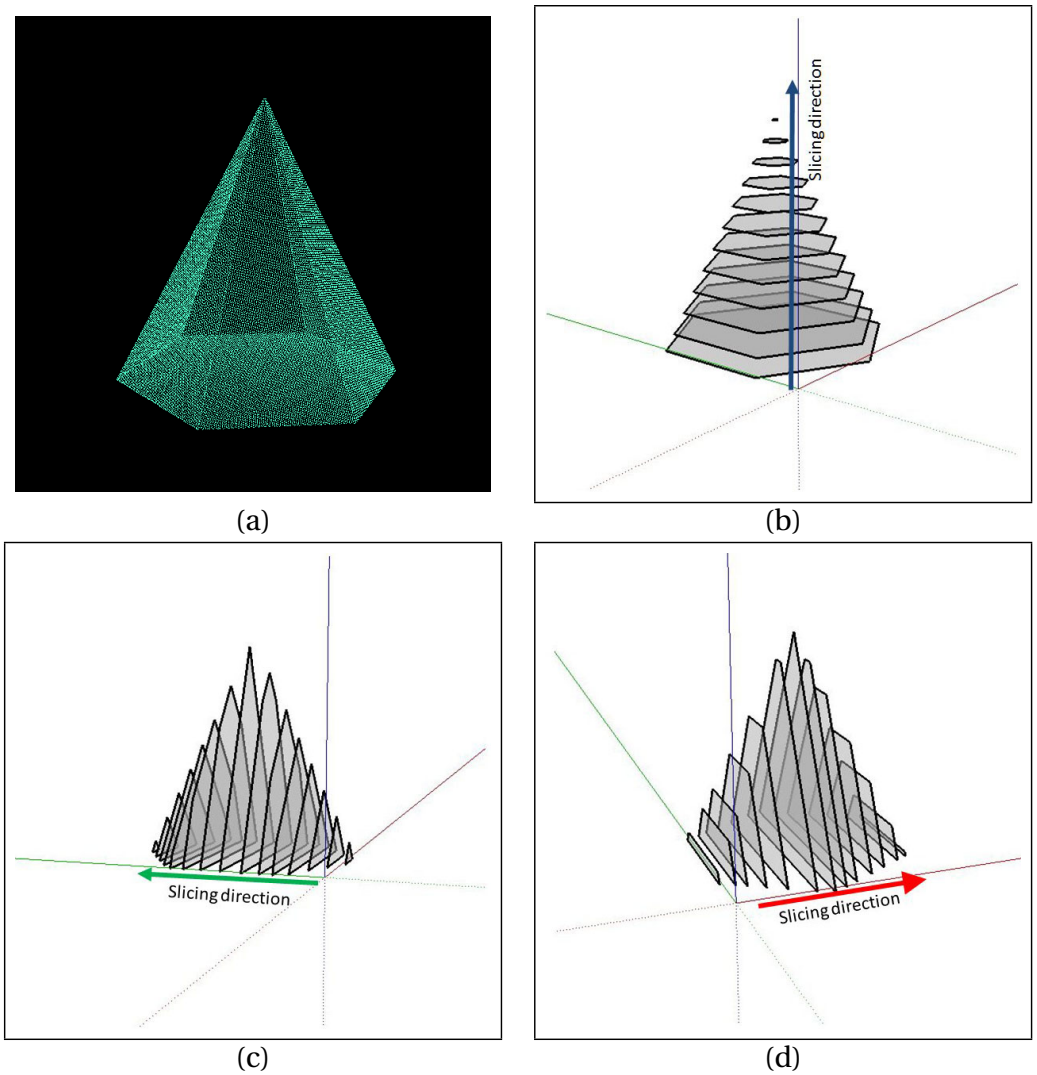


Figure 31: The modeling of a tapering to point structure: (a) the snapshot of the point cloud data; (b) keyslices computed from bottom-up direction; (c) keyslices computed from face-inside direction; (d) keyslices computed from left-right direction.

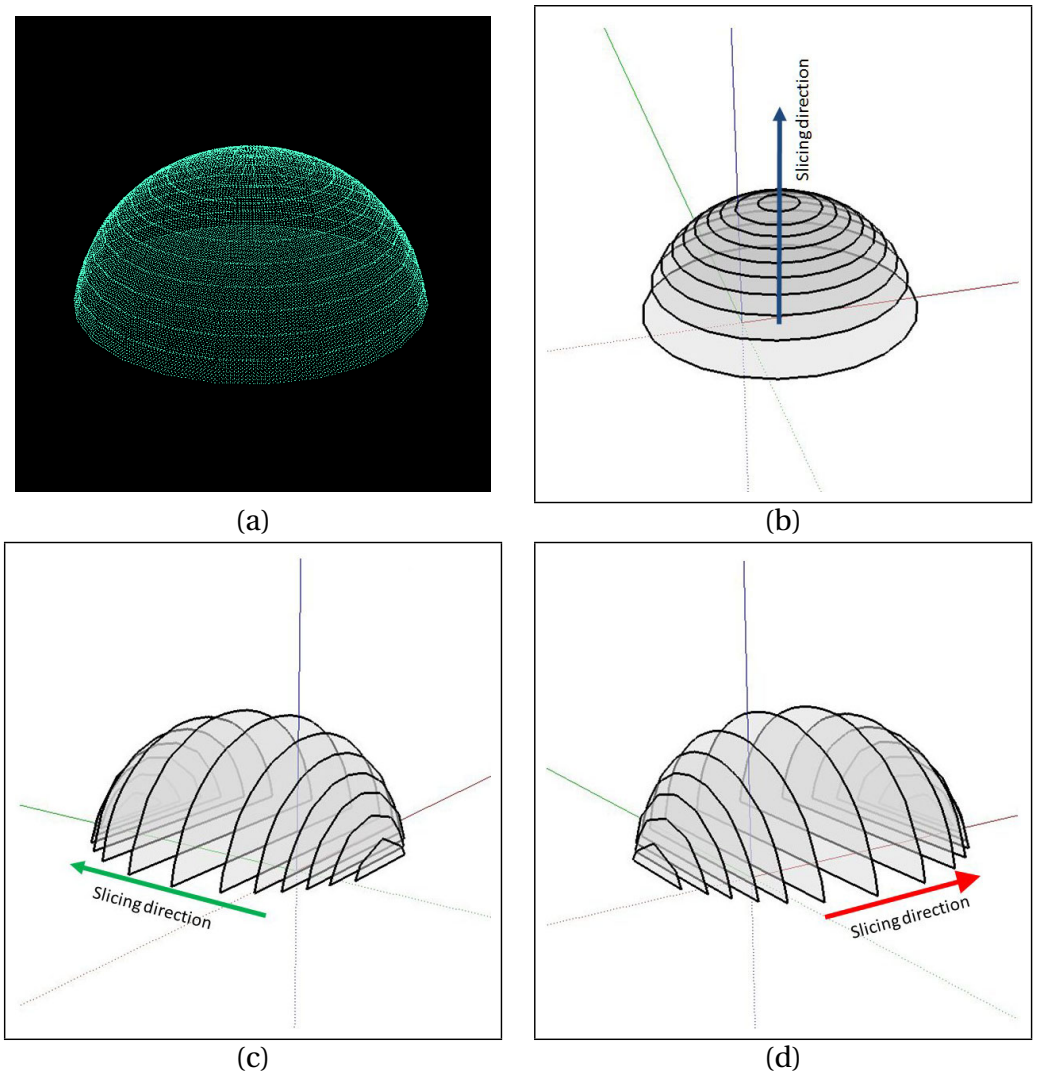
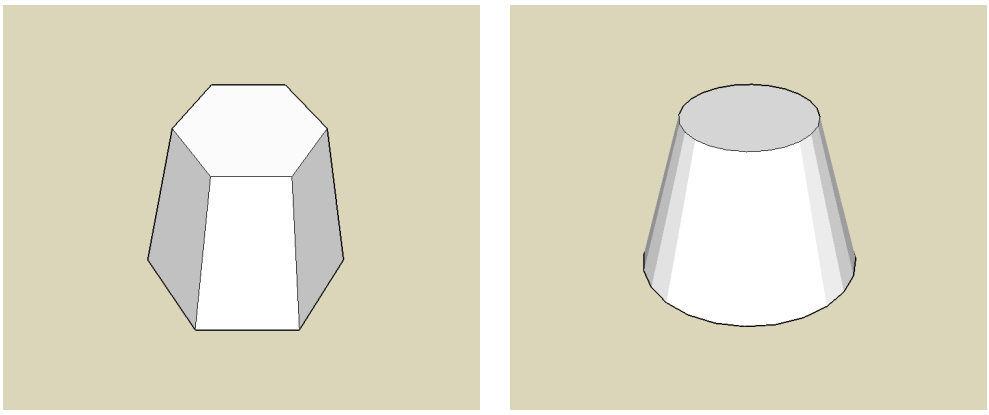


Figure 32: The modeling of a non-linear tapered structure: (a) the snapshot of the point cloud data; (b) keyslices computed from bottom-up direction; (c) keyslices computed from face-inside direction; (d) keyslices computed from left-right direction.

The assumption of linear tapering to a line or point makes the tapered structure inference much simpler and more efficient. For keyslice set $\mathbf{N_K} = \{I_i | 0 \leq i \leq N\}$, where i is the projected slice index, we first compute the sets of keyslices for potential tapered structures. This is based on the observation that for tapered structures, the underlying keyslices indices form nearly an arithmetic progression. Fig. 30(b) and Fig. 31(b) depict the underlying keyslices from bottom-up direction forming a tapering to a line and tapering to a point respectively. That is, $\mathbf{N_T} = \{I_t | I_t \in \mathbf{N_K}\}$, and for any two successive indices, i, j, we have $c_1 \leq |index(I_i) - index(I_j)| \leq c_2$, where c_1 and c_2 are two constants. In practice, $|c_1 - c_2| \leq 1$ and the common difference for the arithmetic progression is either 3 or 4.

The keyslice set $\mathbf{N_T}$ indicates a potential section of a tapered structure in the projected range. $\mathbf{N_T}$ also provides a very important information for inferring the underlying structure: based on the assumption, the first element, I_0, in $\mathbf{N_T}$ is the base geometry shape for the linear tapered structure. And the last element, I_f, of $\mathbf{N_T}$ is the converged or partially converged geometry shape. If I_f contains either a line segment or a point structure, this indicates that the underlying tapered structure of $\mathbf{N_T}$ is completely converged, as illustrated in Fig. 28(a) and Fig. 28(b). If I_f contains a polygon structure, the underlying tapering of $\mathbf{N_T}$ is partially converged structure, which is also called *tapering to offset*, as depicted in Fig. 33(a) and Fig. 33(b).



(a) (b)

Figure 33: Tapering to offset structures: (a) tapering to offset with a polygon base; (b) tapering to offset with a circle base

To compute geometry primitives in I_f, we first compute the bounding box $[X_{min}, X_{max}], [Y_{min}, Y_{max}]$ for I_f. Then, the distance $T_d = max(|X_{min} - X_{max}|, |Y_{min} - Y_{max}|)$ is checked. If $T_d < \delta$, where δ is small threshold, say 4 pixels, we regard the underlying structure m as a linear tapering to a point. Fig. 34(a) shows such a geometry primitive. To represent this TTP structure, the control points of the vectorized based geometry image and the centroid of the I_f are used for reconstruction.

If $T_d >= \delta$, we compute all line segments $\mathbf{L_f}$ in I_f using the adaptive Hough transform described in Algorithm 2. If $\mathbf{L_f}$ only contains one line segment $l = (p_1, p_2)$, as shown in Fig. 34(b), the underlying tapering is completely converged

to a line. To represent this TTL structure, we compute the converging point for each control point $p(x,y)$ of the vectorized contour of I_0, which is given by the smaller distance, $min(\overline{pp_1}, \overline{pp_2})$.

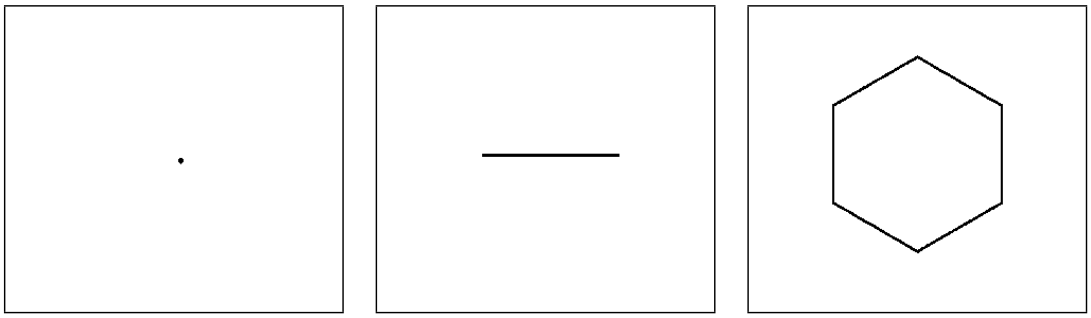


Figure 34: The converged slice I_f for various tapered structures: (a) tapering to point; (b) tapering to line; (c) tapering to offset.

When $\mathbf{L_f}$ contains multiple line segments, as shown in Fig. 34(c), the underlying tapered structure is a tapering to offset structure. Under the assumption, the virtually converged shape could be either a line or a point. To infer the underlying structure, we first compute the median length l_m of all line segments in $\mathbf{L_f}$. If l_m is larger than the threshold, say $l_m > 10$, the base geometry shape I_0 is usually a regular polygon structure as shown in Fig. 33(a). We compute the intersection points of all lines for both I_f and I_0. If the total number of intersection points matches, we can represent this tapered structure using the control points in I_0 and their corresponding control points in I_f.

If l_m is smaller than or equal to the threshold, the shape of I_f is usually a circle or other curved shapes, rather than a regular polygon structure as shown in Fig. 33(b). We regard the underlying structure as a converging to a point structure. To represent this structure, we cannot match each control point in I_0 with its corresponding converging point in I_f anymore. Therefore, we compute the virtual converged point P_v using the triangular equation. We compute the centroids P_{c1} and P_{c2} and the average distance d_1 and d_2 from data points to centroids for both I_f and I_0. P_v can be computed by,

$$\frac{d_1}{d_2} = \frac{|P_v - P_{c1}|}{|P_v - P_{c2}|}$$

Once P_v is computed, the actual partially converged point P_x for the control point P can be computed as,

$$\frac{d_1}{d_2} = \frac{|P_v - P_x|}{|P_v - P|}$$

5.2 Verification of Inferred Tapered Structure

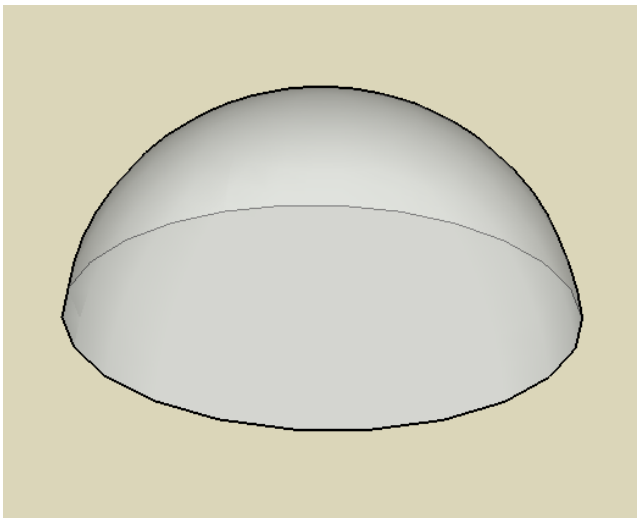
After applying the approach described in Sec. 5.1, we need to verify whether the new detected shape is valid or not. This is because the assumption we made in Sec. 5.1 that the underlying tapered structure is either TTP or TTL. This assumption may not be held for some complicated cases as shown in Fig. 28(c) and Fig. 28(d). The basic idea for the verification stage is to measure the error for the

new inferred tapered structure. For large-scale PCD, it would be time consuming to compute the error for all eligible data points. Therefore, we only sample a certain amount of data points for error measurement.

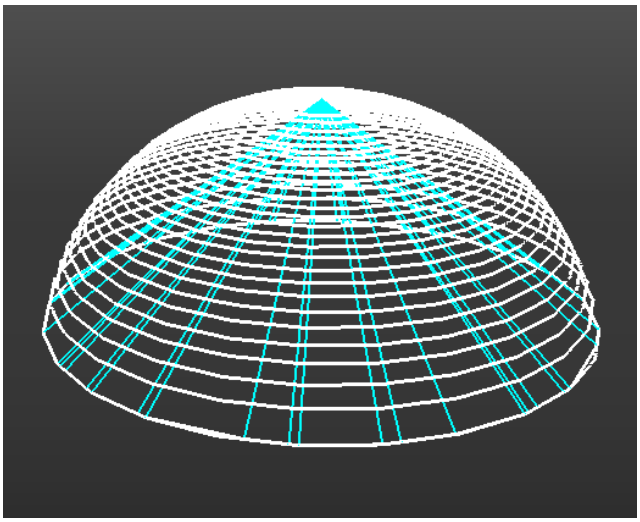
Let N_c and N_t be the current number and the total number of data points for error measurement. A data point $p(x,y,z)$ from current PCD segment is sampled via uniform sampling process. We first compute the 2D sliced index I_p for p using Eq. (5). If $I_0 \leq I_p \leq I_f$, that is, the point p is between the height range of the base location and converged location, p is regarded as a good sampled data and is used for further error computation, which increases N_c by 1. Otherwise, p is discarded and another round of sampling is carried out.

To compute the distance between p and the inferred model M, we first compute all the 3D planes formed by points in the base polygon I_0 and the converged points in the I_f. Given two consecutive control points $p1, p2$ in I_0, and the third converged point $p3$ in I_f, the plane P_t can be computed using the equation Eq. (10) as described in Appendix A. We compute all the planes $\mathbf{P}$ for the tapered structure M.

The distance between a 3D point $p(x,y,z)$ and a plane P_t is given by Eq. (12) as shown in Appendix A. The distance between p and the structure M is defined as the minimum distance from p to all planes in $\mathbf{P}$. that is, $d_p = min(p, P_t)$, where $P_t \in \mathbf{P}$. Therefore, the average error for the tapered structure M is defined as $E_M = \sum_p (d_p)/N_t$. If E_M is smaller than a threshold, t, the tapered structure



(a)



(b)

Figure 35: The verification of the inferred tapered structure: (a) the original model; (b) the computed keyslices (in white) and inferred tapering to point structure (in blue).

is approved and the corresponding keyslices are substituted to further compress the model. Otherwise, the underlying tapered structure is rejected, and we can choose to model this special structure by its keyslices or by fitting a triangular mesh to the underlying 3D point cloud to produce a polygonal model. Fig. 35 shows a dome model which was first modeled as a taper to a point structure (in blue) but later on was rejected by the verification process. We chose $t = \tau_d/2$ as the threshold, and it worked well on our synthetic and real datasets. Here, τ_d is the threshold for keyslice detection as introduced in Sec. 3.2.

Chapter 6

Model Generation

In previous chapters, we have segmented the original 3D PCD, computed the keyslices and their contours for each segment, and identified tapered structures. All these computation lead to the final stage of model generation. The reconstruction of a single segment is straightforward: transform the underlying 2D geometry into 3D coordinate system. For each segment, we first exam whether it can be modeled by extrusion operation on simple keyslices along any major plane. For an extrusion segment, it is a push-pull operation of the basic geometry along its sweeping direction; for a tapered segment, the faces formed by the control points of the base geometry polygon and their corresponding converged points are constructed.

The results of the extrusion and tapered structures computation, together

with keyslice image vectorization, are stored in the 2D image coordinate system. To generate the 3D model, these results need to be transformed back into the 3D world coordinate system. Let $P(x,y)$ be a point in the 2D image coordinate system, and let $P'(x,y,z)$ be the 3D world coordinate of P, where y is the bottom-up (height) coordinate. For all points in the same contour, they lie in the same plane and hence share the same value of y. The equation for transforming P back to P' is as follows,

$$\begin{pmatrix} x^{3D} \\ y^{3D} \\ z^{3D} \end{pmatrix} = \begin{pmatrix} \eta & X_{min} & 0 \\ 0 & \zeta + Y_{min} & 0 \\ 0 & Z_{min} & \eta \end{pmatrix} \begin{pmatrix} x^{2D} \\ 1 \\ z^{2D} \end{pmatrix} \tag{8}$$

where $\eta = (X_{max} - X_{min})/W$, $\zeta = \kappa \cdot \delta$, κ is the index of the 2D slice and δ is the height of a slab. Note that $[X_{min}, X_{max}]$, $[Y_{min}, Y_{max}]$, and $[Z_{min}, Z_{max}]$ pairs define the 3D bounding box, and W is the width of the 2D sliced image as described in Sec. 2.2.

6.1 Model Merging

Because the whole point clouds of a building may be partitioned into segments for modeling, we need to assemble those segments to obtain the final

model. Furthermore, the extrusion operation on 2D keyslice contours may introduce discrepancy on vertices and edges for the adjacent keyslices. Therefore, model merging needs to be carried out for correctly reconstructing the model. Essentially, this process is to merge the vertices and edges shared by the adjacent keyslices or neighboring segments.

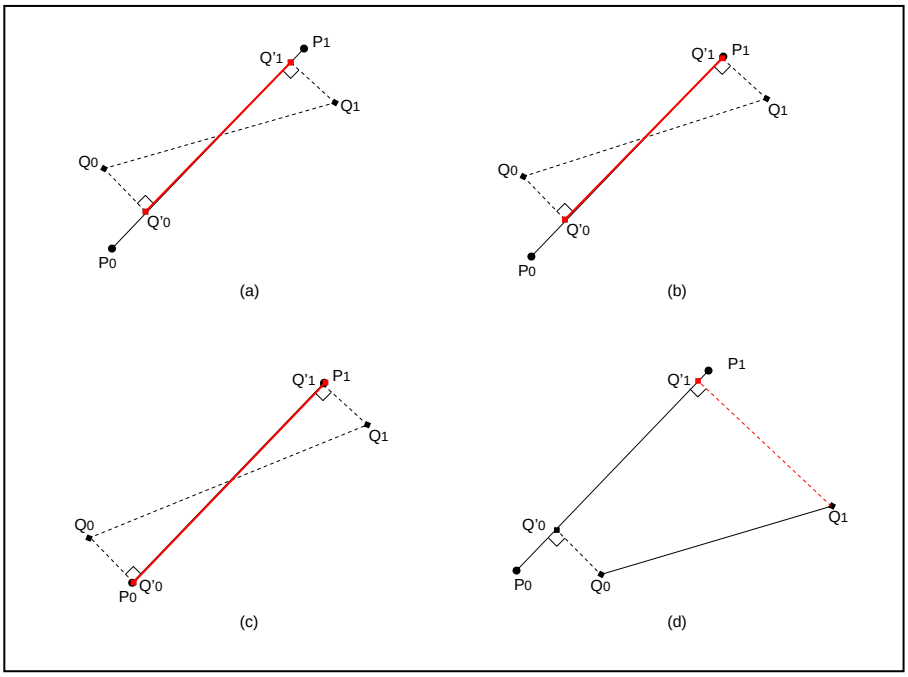


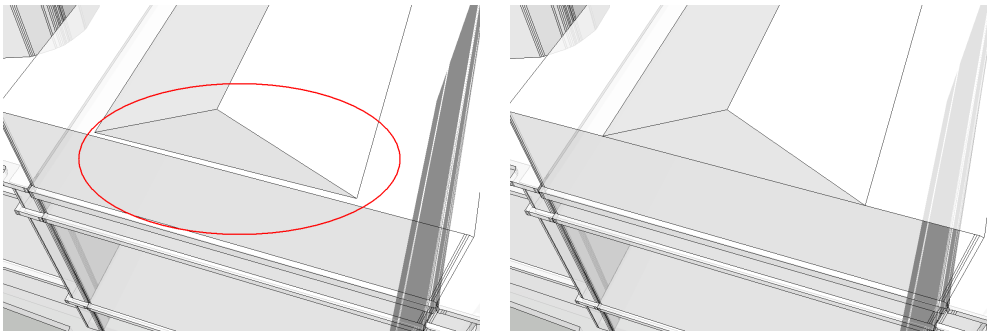
Figure 36: Vertices adjustment for model merging: (a) the two new points Q_0 and Q_1 are projected to an existed line as Q'_0 and Q'_1; (b) after (a), Q'_1 is merged with an existed point P_1; (c) after (a), both Q'_0 and Q'_1 are merged with existed point P_0 and P_1 respectively; (d) one of the point Q_1 is too far away for adjustment.

The first segmented part can be easily reconstructed without merging. Let **M** represent all lines or planes added into the final model so far. When adding a new line segment Q_0Q_1 to the model, we need to compute the closest line $\mathbf{L}=P_0P_1$ in **M** to Q_0Q_1. Let $d=max(dist(Q_0,\mathbf{L}),dist(Q_1,\mathbf{L}))$, i.e., the larger distance between Q_0 to **L** and Q_1 to **L**. The distance from a point Q to a line **L** can be computed using the equation:

$$d(Q,\mathbf{L})=|\mathbf{w}-(\mathbf{w}\cdot\mathbf{u})\mathbf{u}|$$

where $\mathbf{w}=(Q-P_0)$ and **u** is the unit direction vector of **L**.

There are several cases to be considered. If $d<=\delta$, that is, both Q_0 and Q_1 are close enough to L, the projected point Q_0' and Q_1' are used to replace Q_0 and Q_1 respectively as shown in Fig. 36(a). Furthermore, if one of the projected point, say Q_1' is close to an end point of L, say P_1, Q_1' would be merged with P_1 as shown in Fig. 36(b). If both Q_0' and Q_1' are matched with P_0 and P_1, the line Q_0Q_1 would be merged totally with the line L, which was shown in Fig. 36(c). However, if $d>\delta$, no merging should be conducted as shown in Fig. 36(d). Fig. 37(a) and Fig. 37(b) show the structures before and after being merged.



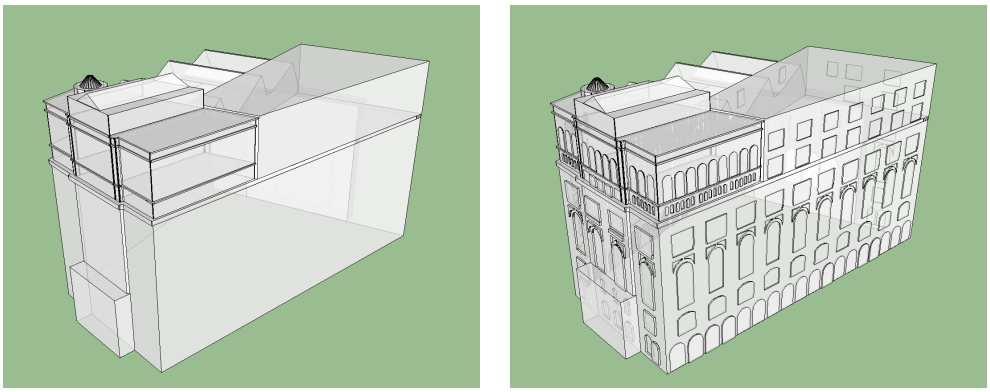
(a) (b)

Figure 37: Model merging computation: (a) two structures inside red ellipse are separated before merged; (b) new structures after merged.

6.2 Window Installation

After the model merging, the whole reconstructed model has been generated. We call this *naked* model in that there is no windows or doors being installed yet as shown in Fig. 38(a). If the point cloud contains windows or doors, we need to add them onto the naked model. The window and door structures have been computed as described in Section 3.1. The problem is which face or plane should they be added.

Let **M** be the set of lines from a naked model. Let $Q_0 = [X_{min}, Z_{min}], Q_1 = [X_{max}, Z_{max}]$ define the bounding box of a window w, i.e., $[Q_0, Q_1]$ is the diagonal points for w. Let $[Y_{min}, Y_{max}]$ be the height range of w. The goal is to



(a) (b)

Figure 38: Windows and doors reconstruction: (a) the reconstructed naked model; (b) the whole model with windows and doors added.

find the right face (closest line segment) for w to be added. For each line segment $L = P_0P_1$ in **M**, we first compute the projected points Q_0', Q_1' of Q_0, Q_1 on L to check whether they are in between $[P_0, P_1]$. If not, L is not eligible for adding w since a window could not fall outside of a facade. Otherwise, the extruded range L_{min}, L_{max} of L is computed. If $Y_{min} > L_{min}$ and $Y_{max} < L_{max}$, that is, the window w is totally covered by the extruded face F which is generated from L, F is a potential face for adding w. The distance $d_p = max(|Q_0Q_0'|, |Q_1Q_1'|)$ is used to compare with global minimum distance d_{p_min} which was set to be a large value initially. If $d_p < d_{p\_min}$, which means a closer face for w is found, d_{p_min} is updated to d_p and the corresponding line L_{win} and its face F_{win} are updated to L and F respectively.

When all lines in **M** are checked, F_{win} will give us the right face for adding w. To add w on F_{win}, we need to compute the projected point P' for each vertex P in w on F_{win}. Let $F_{win} = ax+by+cz+d$, the equation to compute $P(x_0, y_0, z_0)$ is

$$P' = P - \frac{(ax_0+by_0+cz_0+d)}{a^2+b^2+c^2}\mathbf{n}$$

where $\mathbf{n}$ is the normal of F_{win}. This will generate a closed face F_w on F_{win}. A simple push-pull operation on F_w with the distance d_w which was computed in the Section 3.1 will add the window w onto F_w of the naked model. Fig. 38(b) shows the entire model with windows and doors added for the model in Fig. 38(a).

6.3 Experimental Results

We have applied the proposed method on both synthetic and real datasets. Fig. 39 through Fig. 48 shows the experimental results for synthetic datasets. The reconstructed model on real datasets are shown in Fig. 49 through Fig. 51 which are corresponding to Thomas Hunter building at Hunter College, Cooper Union building in downtown New York, and Grand Central Terminal building in middletown New York, respectively.

For each set of the figures, the snapshot of the original SketchUp model (or the building photo for real dataset), the point cloud dataset, the segmentation, the keyslices, the extruded model, and the final models from different viewpoints

are shown in order. If applicable, the snapshot of the tapered model and the model with windows and doors installed are also shown. The thresholds used for the computation are $\tau_r = 4$ and $\tau_d = 4$.

In addition to exterior models, we have also applied the lightweight reconstruction algorithm to the range data of interior scans. The snapshot of an interior scan of Hunter Theater is shown in Fig. 52(a) and its reconstructed 3D model is shown in Fig. 52(b). This model is primarily reconstructed using the extrusion operation upon the main structures of the interior. The chairs and some other fine details were not able to be handled and therefore were manually culled.

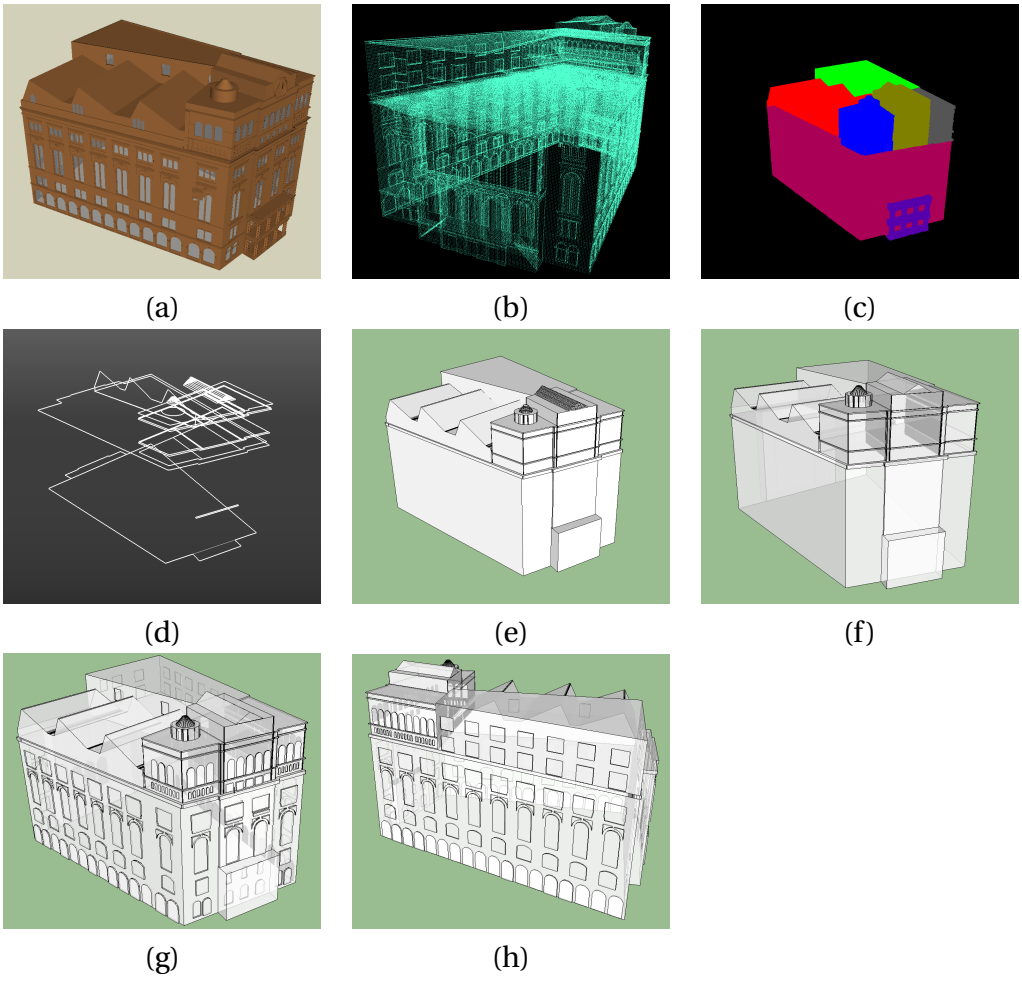


Figure 39: Experimental results of Cooper Union model: the snapshots of (a) the original SketchUp model; (b) synthetic point cloud data generated from (a); (c) segmentation; (d) keyslices; (e) extruded model; (f) tapered model; (g) reconstructed model with windows installed (I) ; (h) reconstructed model with windows installed (II) .

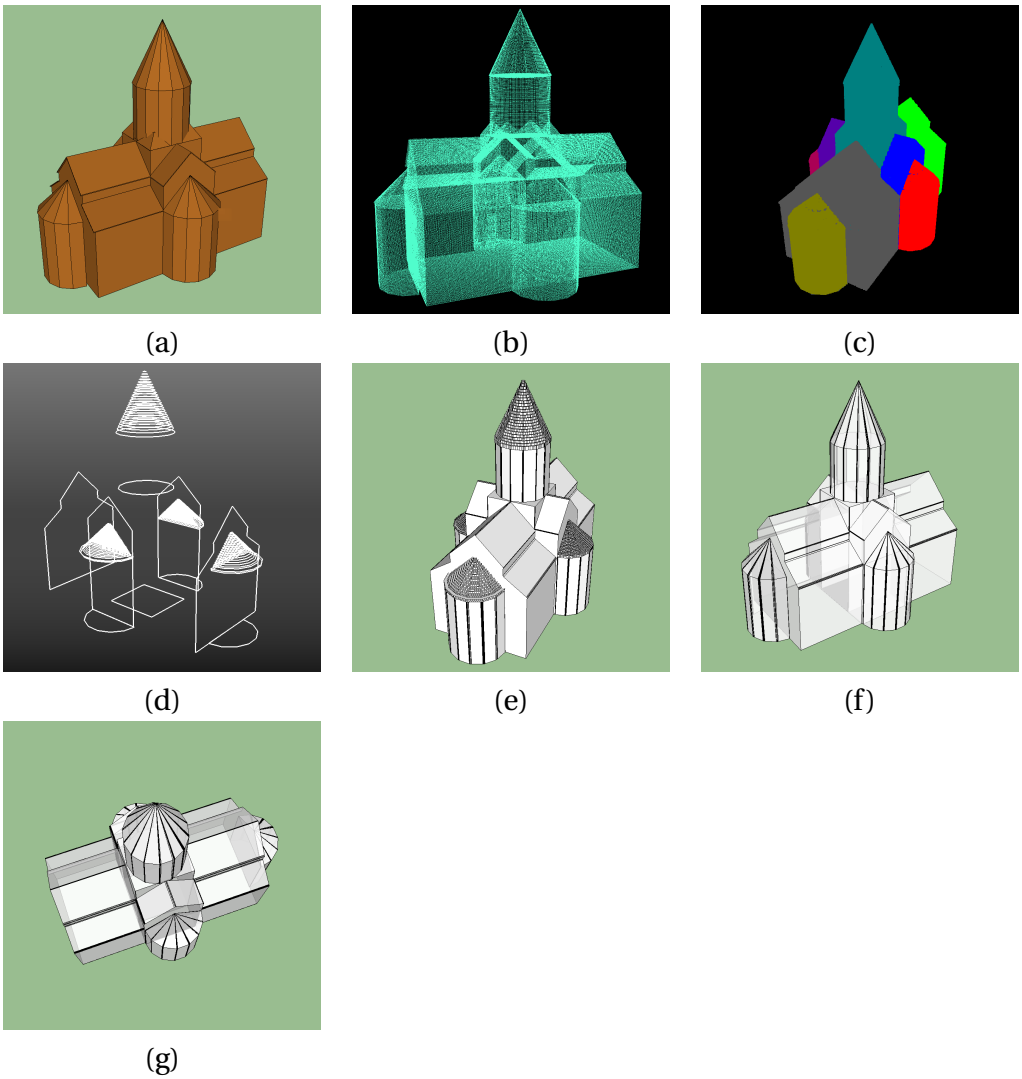


Figure 40: Experimental results of Spitak model: the snapshots of (a) the original SketchUp model; (b) synthetic point cloud data generated from (a); (c) segmentation; (d) keyslices; (e) extruded model; (f) tapered model; (g) reconstructed model.

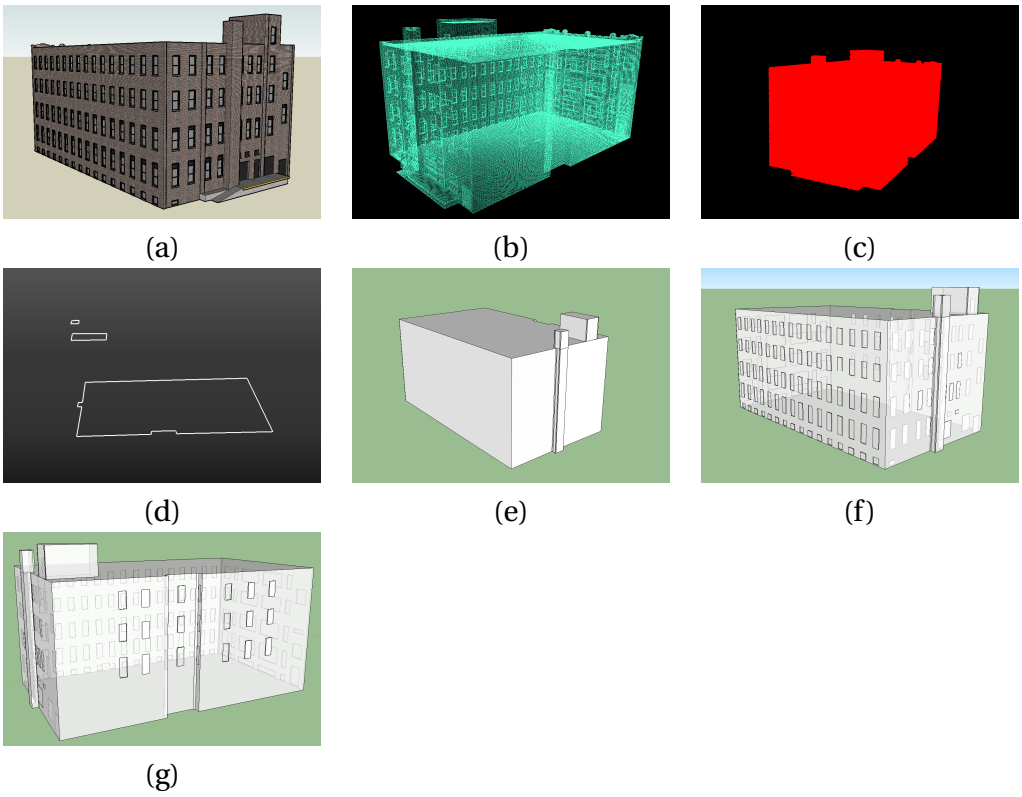


Figure 41: Experimental results of apartment model: the snapshots of (a) the original SketchUp model; (b) synthetic point cloud data generated from (a); (c) segmentation; (d) keyslices; (e) extruded model; (f) the model with windows installed; (g) reconstructed model.

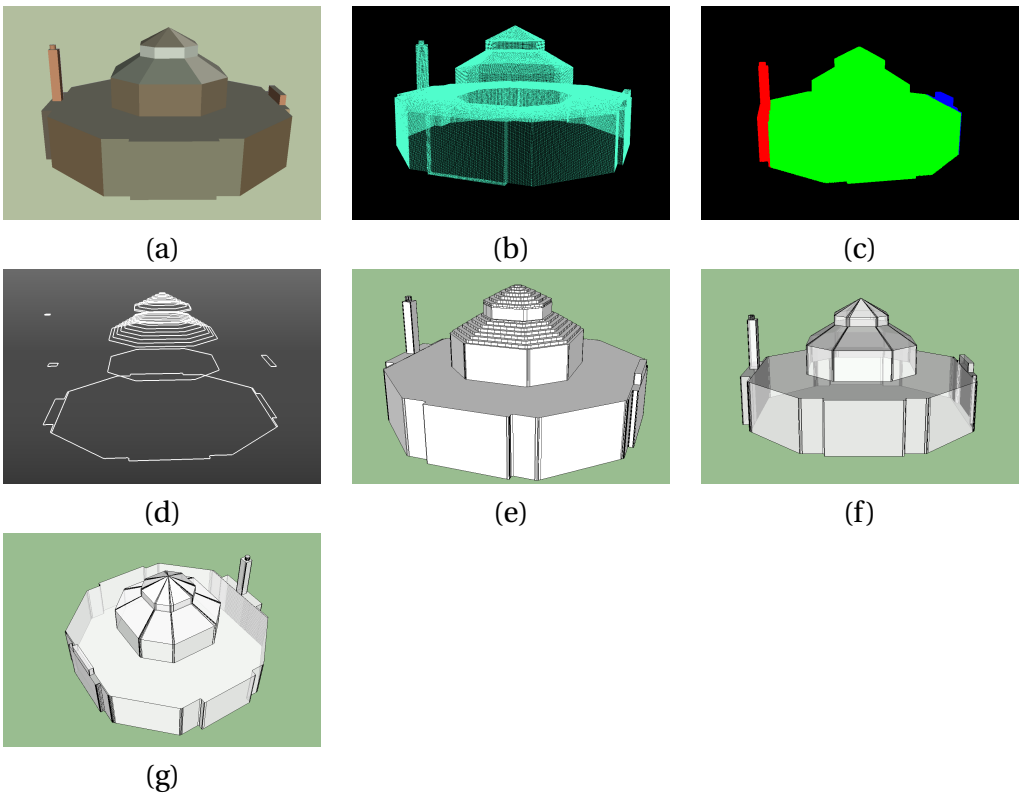


Figure 42: Experimental results of Branch library model: the snapshots of (a) the original SketchUp model; (b) synthetic point cloud data generated from (a); (c) segmentation; (d) keyslices; (e) extruded model; (f) tapered model; (g) reconstructed model.

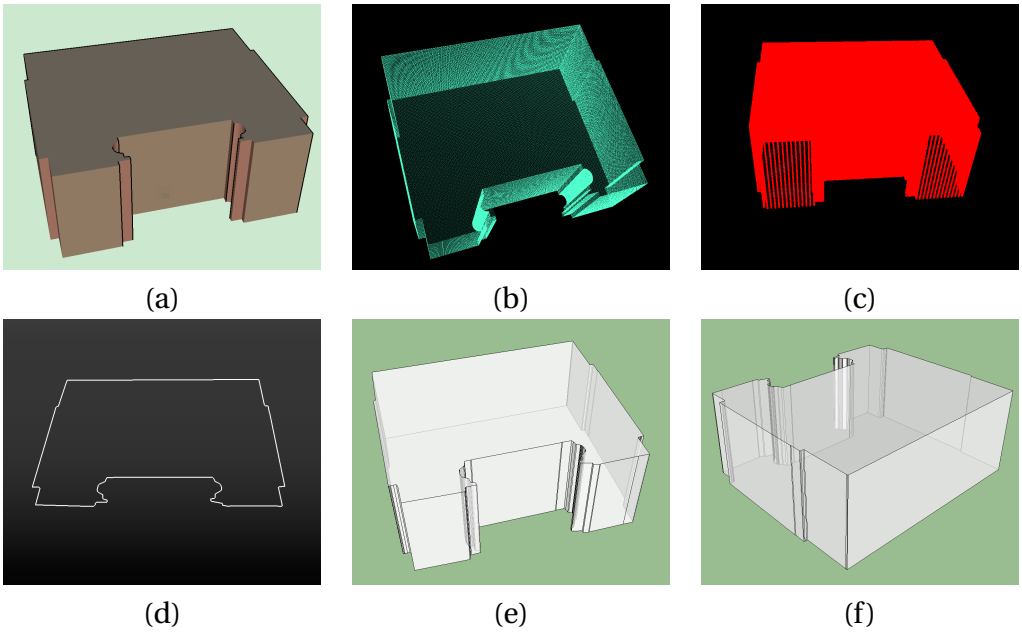


Figure 43: Experimental results of Clements library model: the snapshots of (a) the original SketchUp model; (b) synthetic point cloud data generated from (a); (c) segmentation; (d) keyslices; (e) extruded model; (f) reconstructed model.

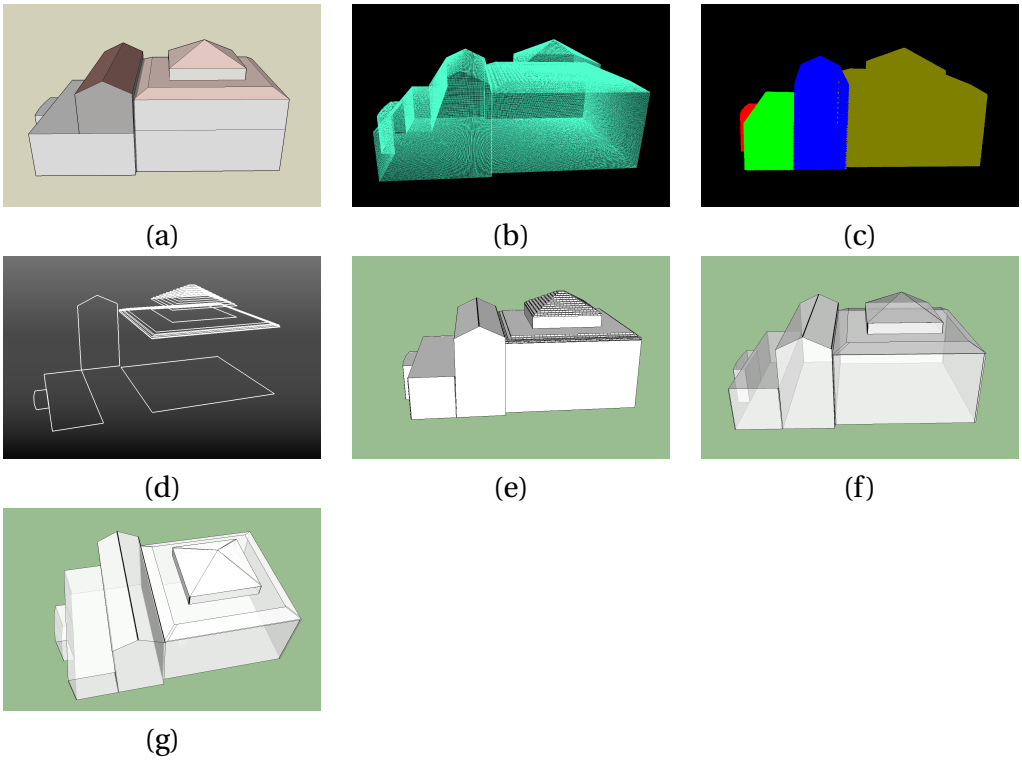


Figure 44: Experimental results of Doe library model: the snapshots of (a) the original SketchUp model; (b) synthetic point cloud data generated from (a); (c) segmentation; (d) keyslices; (e) extruded model; (f) tapered model; (g) reconstructed model.

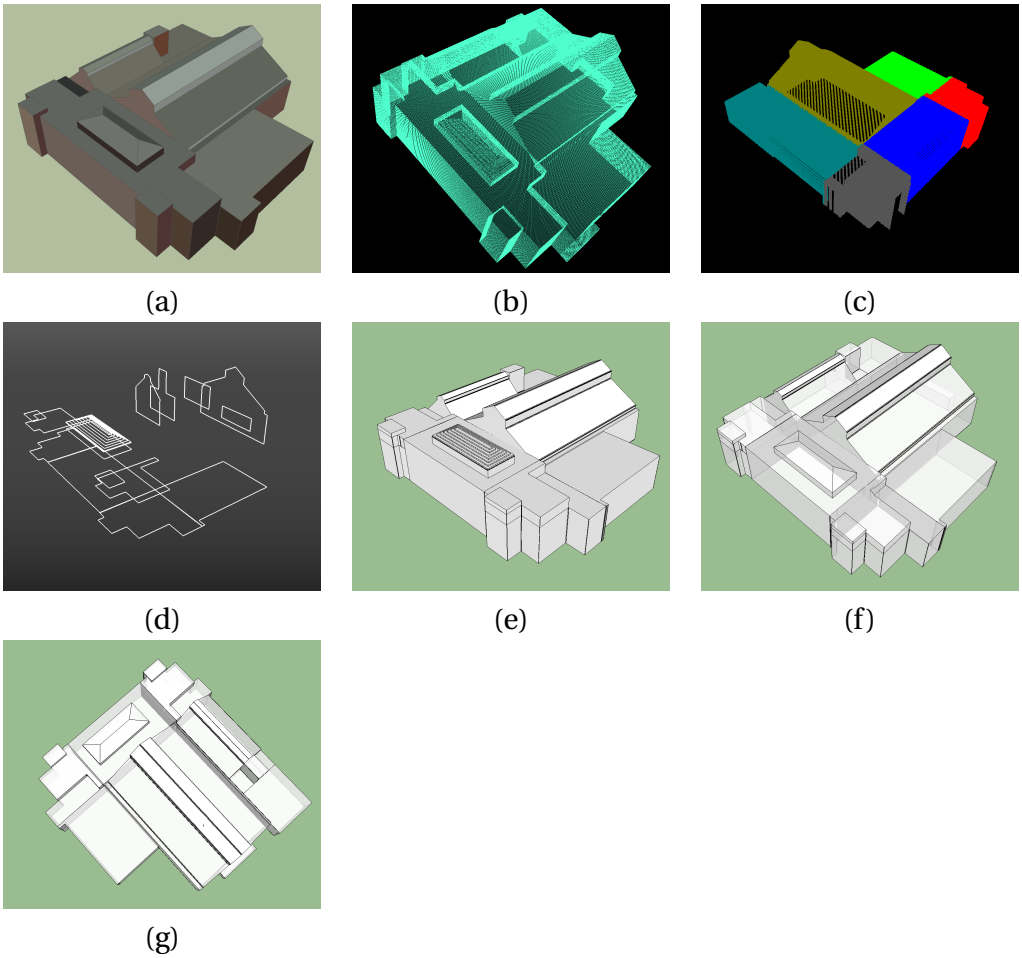


Figure 45: Experimental results of Health town library model: the snapshots of (a) the original SketchUp model; (b) synthetic point cloud data generated from (a); (c) segmentation; (d) keyslices; (e) extruded model; (f) tapered model; (g) reconstructed model.

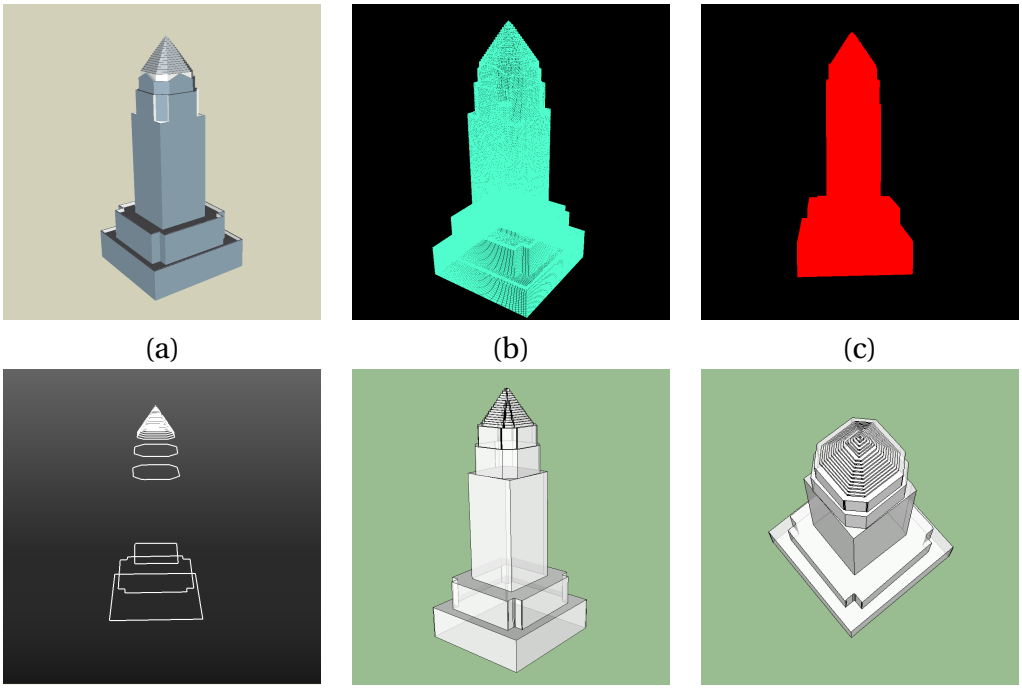


Figure 46: Experimental results of Miami Date Courthouse model: the snapshots of (a) the original SketchUp model; (b) synthetic point cloud data generated from (a); (c) segmentation; (d) keyslices; (e) extruded model; (f) reconstructed model.

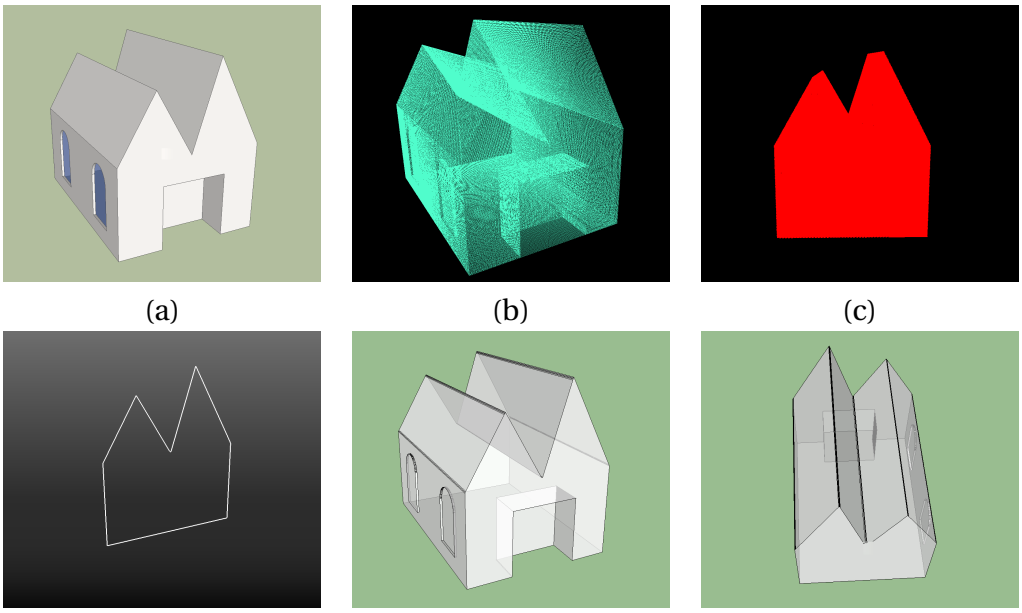


Figure 47: Experimental results of a simple model: the snapshots of (a) the original SketchUp model; (b) synthetic point cloud data generated from (a); (c) segmentation; (d) keyslices; (e) extruded model; (f) reconstructed model.

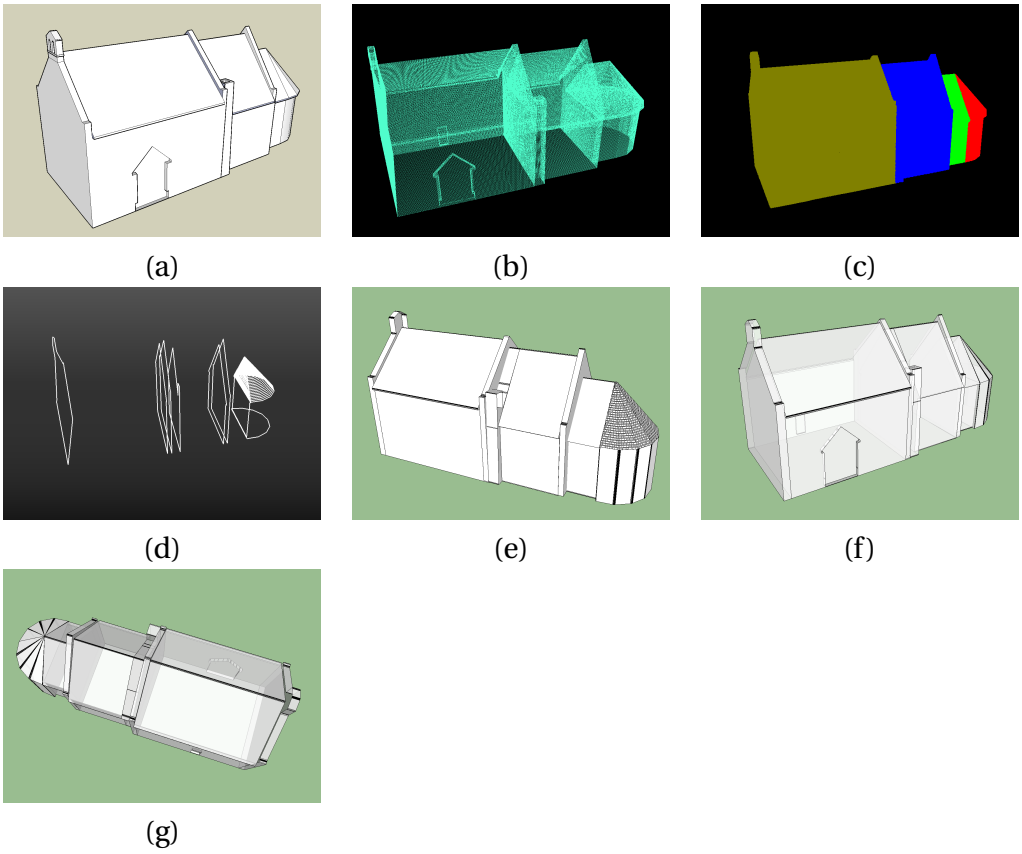


Figure 48: Experimental results of St. Michael church model: the snapshots of (a) the original SketchUp model; (b) synthetic point cloud data generated from (a); (c) segmentation; (d) keyslices; (e) extruded model; (f) tapered model; (g) reconstructed model.

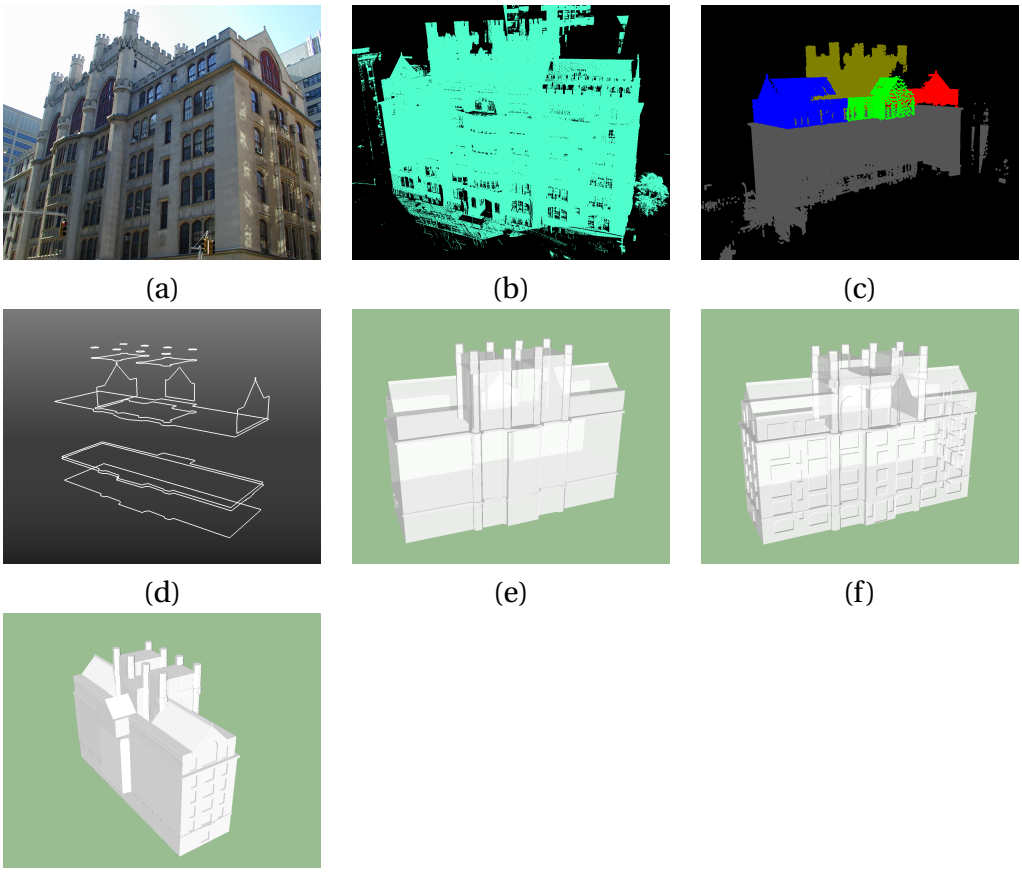


Figure 49: Experimental results of Thomas Hunter building: the snapshots of (a) real building; (b) point cloud dataset acquired from laser scanner; (c) segmentation; (d) keyslices; (e) extruded model; (f) reconstructed model with windows installed; (g) reconstructed model.

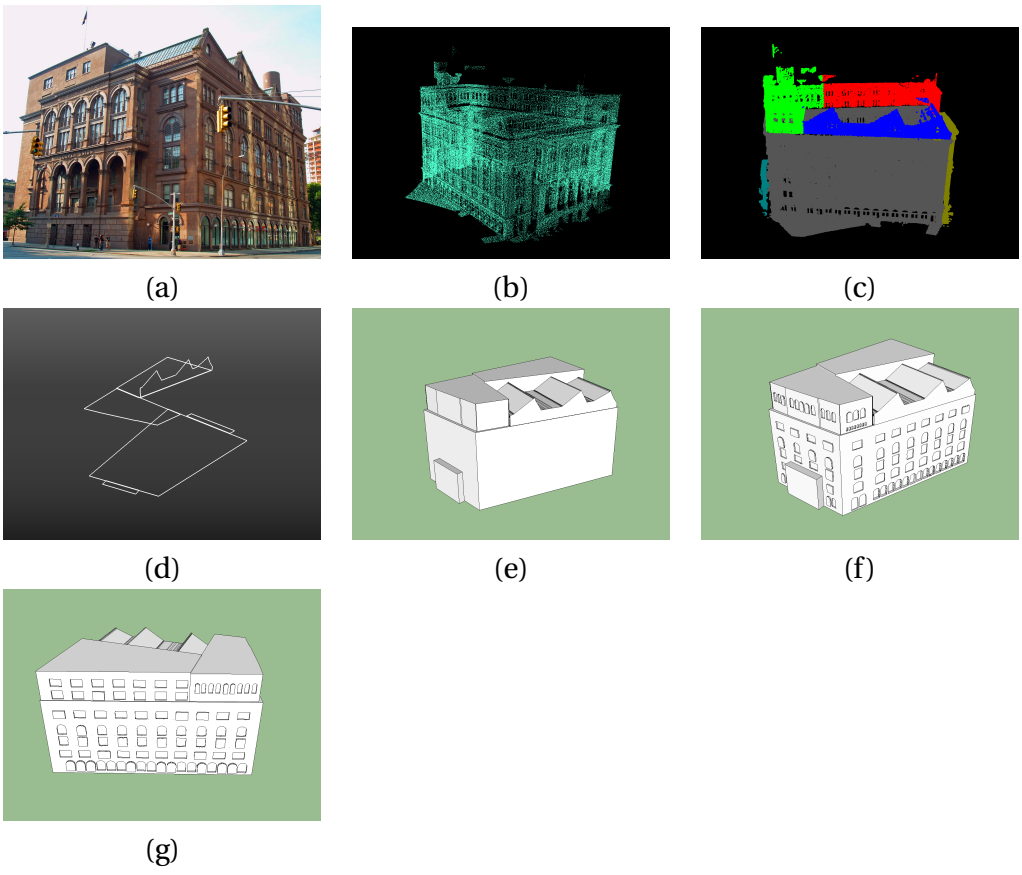


Figure 50: Experimental results of Cooper Union building: the snapshots of (a) real building; (b) point cloud dataset acquired from laser scanner; (c) segmentation; (d) keyslices; (e) extruded model; (f) reconstructed model with windows installed; (g) reconstructed model.

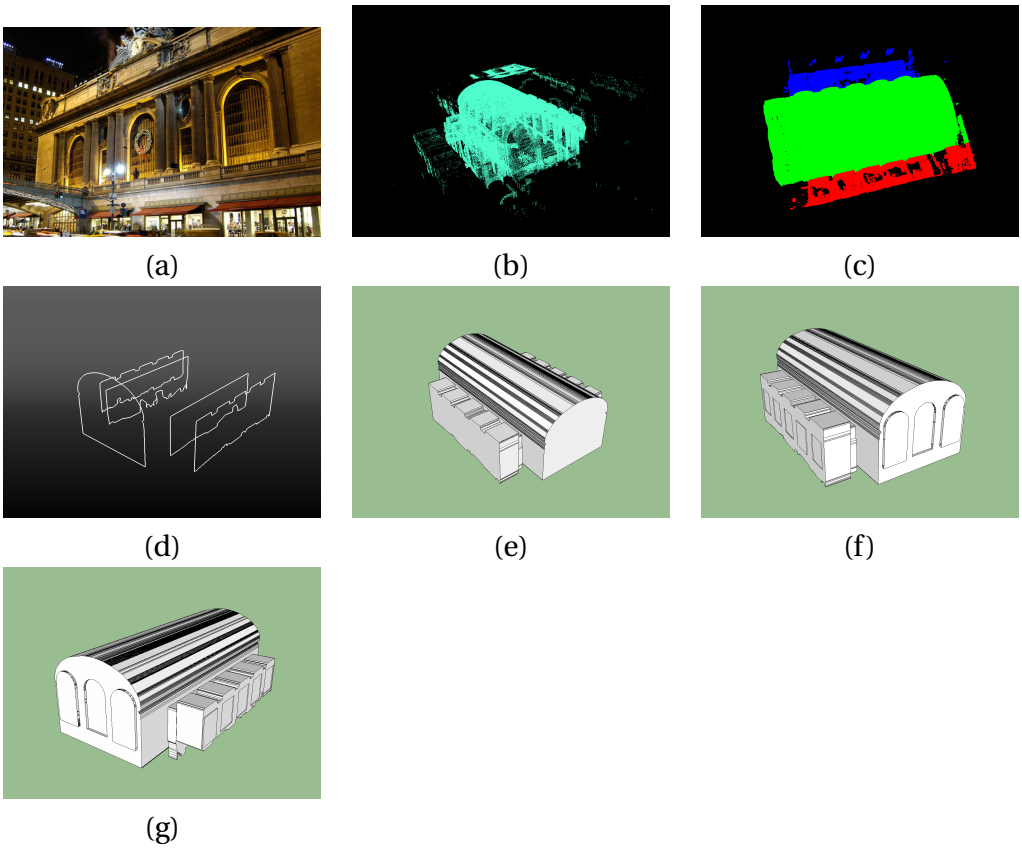
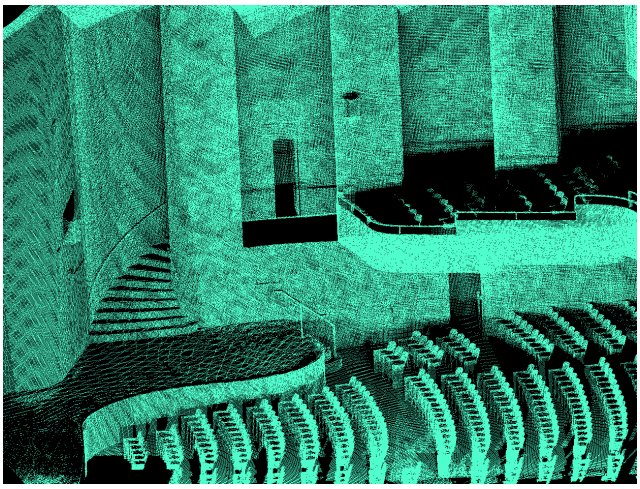
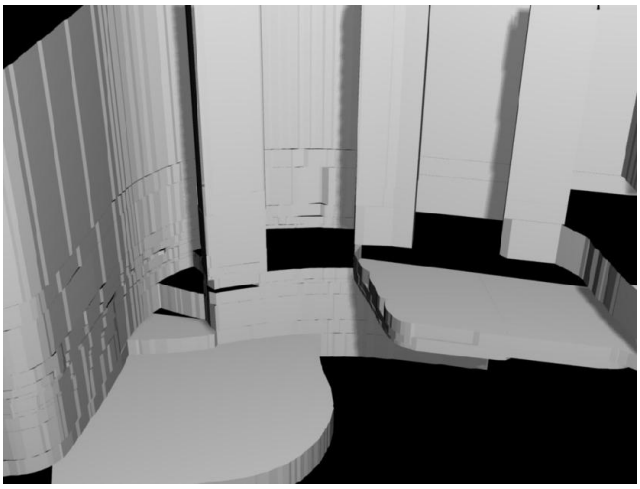


Figure 51: Experimental results of Grand Central Terminal building: the snapshots of (a) real building; (b) point cloud dataset acquired from laser scanner; (c) segmentation; (d) keyslices; (e) extruded model; (f) reconstructed model with windows installed; (g) reconstructed model.



(a)



(b)

Figure 52: The model of an interior scan: (a) the input point cloud data; (b) the reconstructed lightweight 3D model.

Chapter 7

Performance Evaluation

7.1 Complexity Analysis

Given a 3D point cloud dataset with N points, the kd-tree construction in the major plane detection takes $O(NlogN)$ time for both space and computational complexity [52]. The query operation for kd-tree takes $O(N^{1-1/k})$ time, where k is the dimension of the kd-tree. The MLS based normal computation is bounded by $O(m)$, where m is the number of neighbor points. Because m is bounded by a constant value, the computational complexity for the major plane detection is of $O(m*NlogN)=O(NlogN)$. The space complexity of the whole major plane detection is also bounded by $O(NlogN)$.

Based on Eq. (5), the projection computation for each 3D point takes only

$O(1)$ run time. Therefore, the projection of 3D point cloud to 2D images algorithm is of $O(N)$ computational complexity for each major plane. Because there are only limited number of major planes for a building, the whole 3D projection process is of $O(N)$ computational complexity. For our fast projection implementation, every 3D point is stored in memory and therefore space complexity of projection is also $O(N)$.

The image processing based algorithms are much cheaper compared to the above 3D computations. The noise removal process takes $O(n)$ complexity, where n is the size of the 2D image. The hole filling is based on symmetry computation of translation, which is bounded by $O((w+h)*n)$, where w and h are the width and height of the 2D image. Therefore, the complexity of the hole filling is $O(n)$.

As pointed out in Sec. 3.4, the keyslice detection algorithm takes $O(n)$ complexity. The boundary vectorization algorithm also takes $O(n)$ complexity. The space complexity of both the keyslice detection and the boundary vectorization is also bounded by $O(n)$. The tapering detection algorithm consists of two stages. The first stage is based on the keyslices computed earlier, and is bounded by the complexity of $O(n)$. The second stage which computes the errors for the reconstructed tapered model is bounded by $O(N)$. As we know, $N > n$, therefore the overall time and space complexity for the whole system is bounded by $O(NlogN)$.

7.2 Parameters and Error Estimation

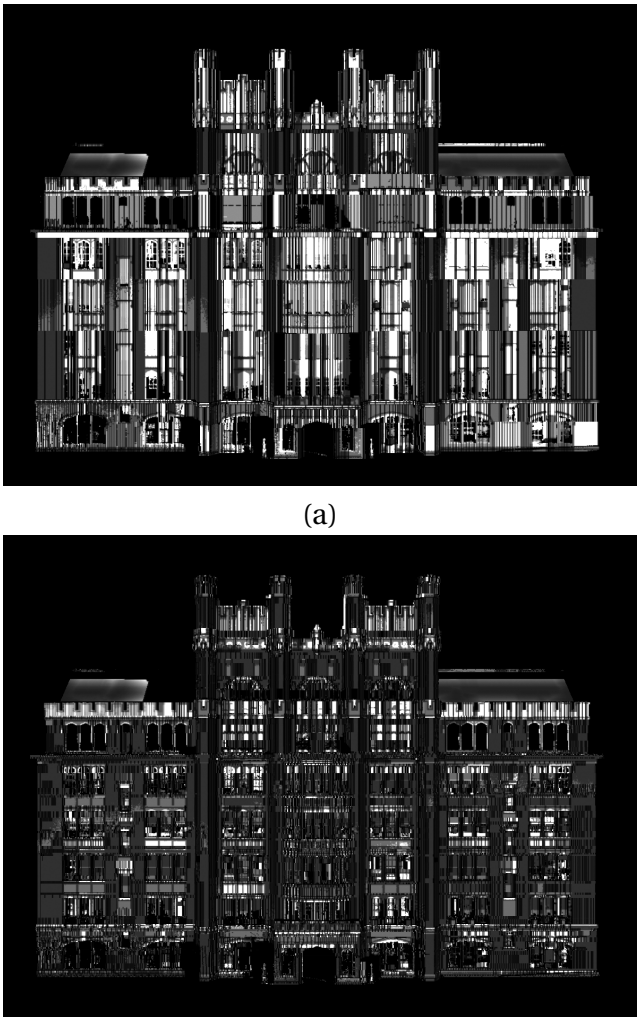
To measure the error of a reconstructed 3D model, we first transform it to the 3D point cloud coordinate system. The error E is measured as the distance between the 3D points in the point cloud dataset and their closest planes in the reconstructed model M:

$$E = \frac{1}{|X|} \sum_{x \in X} d(x, M) \tag{9}$$

where X is the set of 3D points in the point cloud, and distance $d(x,M) = \min_{p \in M} \|x - p\|$ is the minimum Euclidean distance from a 3D point x to its closest face p of M.

To visualize the error between real 3D data and the inferred model, we generate deviation map images. Two such images are shown in Fig. 53 for the point cloud data in Fig. 4(b). The deviation maps are constructed as follows. For each face p of M, a corresponding texture image is computed. The intensity of each pixel in the texture image is determined by the error of the corresponding 3D points computed by Eq. (9). The accuracy of the reconstructed model is controlled by distance measurement threshold τ_d and BPA refinement radius τ_r. Threshold τ_d determines the accuracy of keyslice detection and τ_r determines the accuracy of boundary vectorization.

Table 1 lists the relationship among the τ_d, errors, number of faces, and model size for the input data in Fig. 4(b). The unit for τ_d is in pixels and the unit



(a)

(b)

Figure 53: The deviation map of the 3D point cloud: (a) the result with $\tau_r = 4$ and $\tau_d = 32$; (b) the result with $\tau_r = 1$ and $\tau_d = 4$.

for error is in meters. The size of the original point cloud for the 3D building is more than 700 MB. From the table, one can see that even for the most accurate model, the size is dramatically reduced compared with the original 3D point cloud data. This is a desirable property for web-based applications. The low resolution ($\tau_d = 64$) and high resolution ($\tau_d = 4$) models were generated in 15 and 120 minutes, respectively, on a laptop PC using an Intel Core 2 T7200 CPU at 2.0 GHz with 2.0 GB RAM. Future work includes the optimization of the BPA vectorization module since it consumes approximately 70% of the computation time.

τ_d (pixel)	Error (m)	# of faces	Size (KB)	time (s)
64	$.658 \pm .158$	1471	15	1977 (32'57")
32	$.294 \pm .103$	3284	32	2353 (39'13")
16	$.141 \pm .058$	8574	86	3008 (50'08")
8	$.131 \pm .074$	13955	137	3696 (61'36")
4	$.094 \pm .068$	27214	261	5391 (89'51")
2	$.088 \pm .036$	31331	335	7586 (126'26")
1	$.083 \pm .041$	32187	337	10927 (182'07")

Table 1: Error measurements for reconstruction of Thomas Hunter dataset using distance measurement threshold τ_d and BPA radius threshold $\tau_r = 4$.

τ_d (pixel)	Error (m)	# of faces	Size (KB)	time (s)
64	$.085 \pm .023$	5741	55	617 (10'17")
32	$.063 \pm .021$	8614	86	623 (10'23")
16	$.052 \pm .032$	9810	94	674 (11'14")
8	$.028 \pm .015$	15338	149	740 (12'20")
4	$.022 \pm .017$	42877	420	1042 (17'22")
2	$.012 \pm .009$	91736	988	1688 (28'08")
1	$.011 \pm .009$	101466	990	1708 (28'28")

Table 2: Error measurements for reconstruction of Hunter Theater dataset using distance measurement threshold τ_d and BPA radius threshold $\tau_r = 4$.

7.3 Model Comparison

Although models generated by 3D BPA are of high resolution, they usually require excessive storage capacity. The model in Fig. 54, for example, needs almost 400 MB of storage, which prevents this solution from being applied to web-based applications. One way to improve matters is to apply some approximation/decimation technique to reduce the space required by these models.

The holes in the 3D BPA model in Fig. 54 are present in the original dataset. They are due to the fact that the laser never reflected back to the scanner after penetrating the glass windows. The 3D BPA method is deficient in filling these holes. We counter this problem by first applying a symmetry-based hole filling algorithm on the 2D slices to create enhanced slices that are processed by an adaptive 2D BPA method to fill gaps. Finally, an extrusion operation is applied to

create a watertight 3D model.

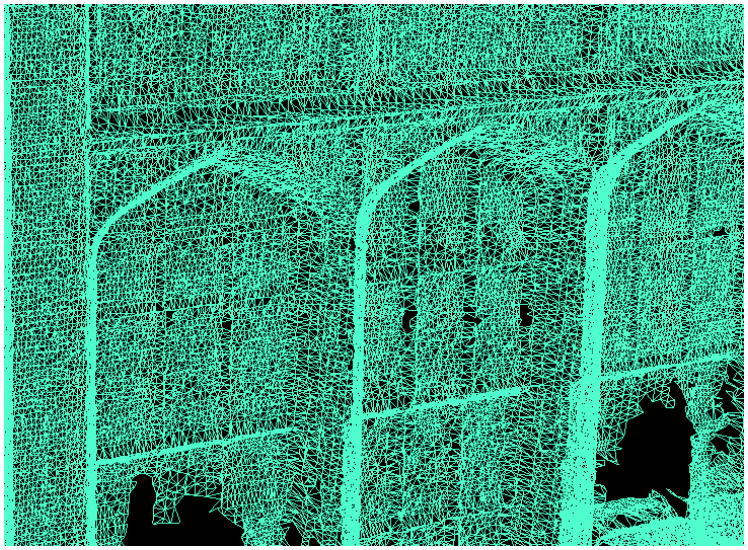


Figure 54: Dense triangulated BPA mesh cropped from Fig. 4(b).

Among all mesh reduction techniques, *qslim* is one of the most sophisticated and efficient algorithms. We carried out a comparison between models generated by our proposed method and those approximated by *qslim*. The comparisons were conducted on models sharing the same number of faces. It is worth noting that *qslim* ran out of memory on the 3D model data generated by BPA in Fig. 54. In order to reduce the size of the model for *qslim* to work, we had to either down-sample the 3D model generated by BPA or split it into sub-models which then can be handled by *qslim*.

Fig. 55(a) and Fig. 55(c) respectively depict the models generated by *qslim*

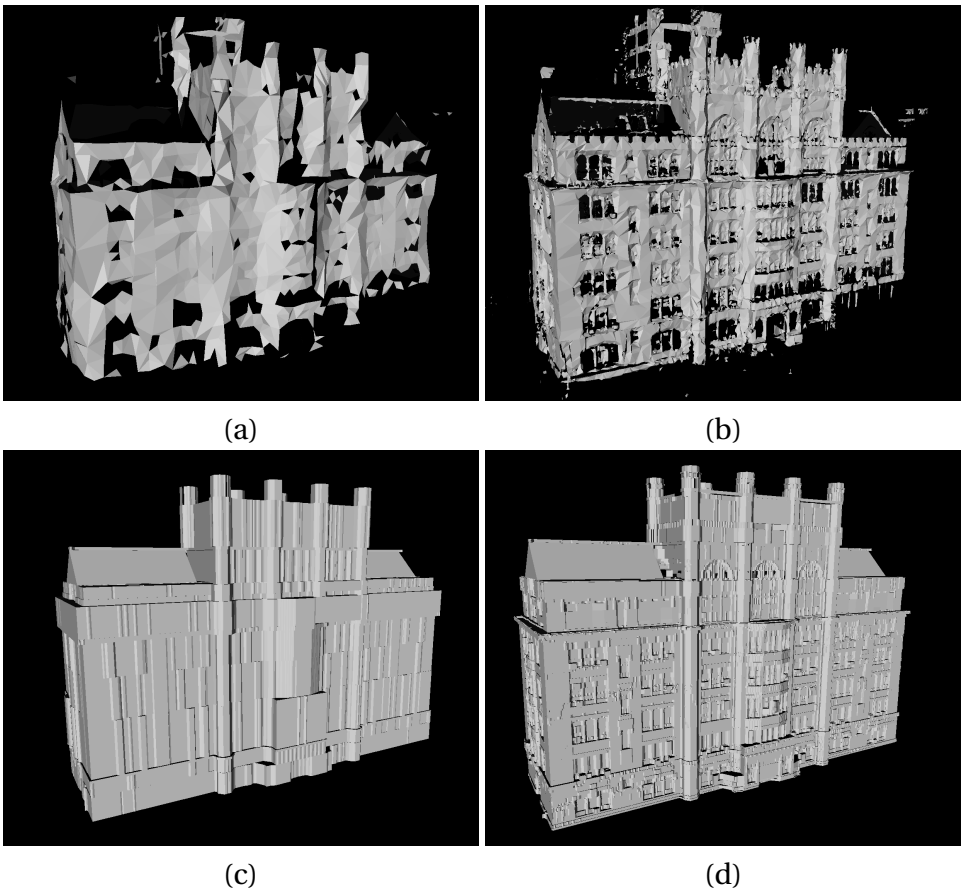


Figure 55: Models generated by *qslim* having (a) 2,000 and (b) 32,000 faces. Models generated by our approach having (c) 2,000 and (d) 32,000 faces.

and our proposed method with approximately 2,000 faces each. Higher resolution models with roughly 32,000 faces each are shown in Fig. 55(b) and Fig. 55(d). Notice that the models approximated by *qslim* are inferior since they do not preserve the sharpness of the original model and are replete with holes. Our symmetry detector and extrusion operation guarantees no holes.

Chapter 8

Conclusion and Future Work

This thesis has presented an efficient algorithm for lightweight 3D modeling of urban buildings from range data. The work is motivated by the observation that buildings can be viewed as the combination of two basic components: extrusion and tapering. The proposed framework combines the benefits of *a priori* knowledge of urban buildings and fast 2D image processing techniques to perform 3D modeling of urban buildings and can generate models across a wide spectrum of resolutions. This offers the benefit of a cost-effective geometry compression approach for voluminous range data within the domain of urban structures.

Major planes and facades are first identified for the unstructured input point cloud dataset, which then is partitioned into volumetric slabs whose 3D points

are projected onto a series of uniform cross-sectional images. For real datasets, which usually contain lots of noise and hole, the image processing techniques are applied to improve the quality of the projected images. The whole range data is also segmented into smaller units which contain only either extrusion or tapered structures. This divide and conquer strategy makes the original problem much easier to tackle.

For each segment, prominent keyslices are extracted and these keyslices are vectorized using an adaptive BPA algorithm to form a set of polygonal contour slices. We achieve further geometry compression by detecting a series of slices that coincide with a tapering operation. In the current work, we demonstrate how to infer the linear tapering geometry structure, including tapering to point, tapering to line and tapering to offset. Applying extrusion and tapering operations to these keyslice contours forms lightweight 3D models for each segment. The entire model is reconstructed by merging all the 3D models of the segments.

The contributions of this thesis include:

- A framework for lightweight modeling of urban buildings from heavy 3D point cloud datasets based on a series of extrusion and tapering operations along dominated axes.
- A *separators* detection based segmentation algorithm which focuses on salient features dedicated to buildings.
- A similarity measure on binary images which combines the area based and

feature based methods to achieve better accuracy and efficiency.

- A boundary vectorization algorithm based on 2D BPA which can fill holes and suppress noise inside the boundary.
- An efficient tapering inference algorithm which identifies and verifies potential linear tapered structures.

Experimental results on both synthetic and real urban building datasets have been presented to validate the proposed approach. Our proposed approach outperforms the combination of BPA and *qslim* by preserving the sharp detail of the original model. We have also shown how to tune the two key parameters, τ_r and τ_d, to adjust the performance of the reconstruction, which shows the trade-off between the qualify of the results and the time and space resources needed for it. One direction of future research is to carry out theoretical study on these key parameters, so that we can compute the optimal values for them based on the characteristics of the input point cloud datasets.

There are some limitations on the proposed method though. First of all, the error introduced in the earlier stages could be propagated to the later stages, which could largely affect the final results. For example, if the computation of the major planes introduces some errors and therefore the normals for sweeping directions are not precise enough, the 2D slices extracted along these directions would be problematic. This may cause the failure of the keyslices detection, and therefore leading to the distorted reconstructed model.

Another limitation of the proposed method is that it could not handle the intersection of two structures from different directions. For example, Fig. 56(a) shows a case where two extruded structures intersect and Fig. 56(b) shows a case with intersection of a tapered structure and an extruded structure. Both of them could not be handled by the proposed approach yet.

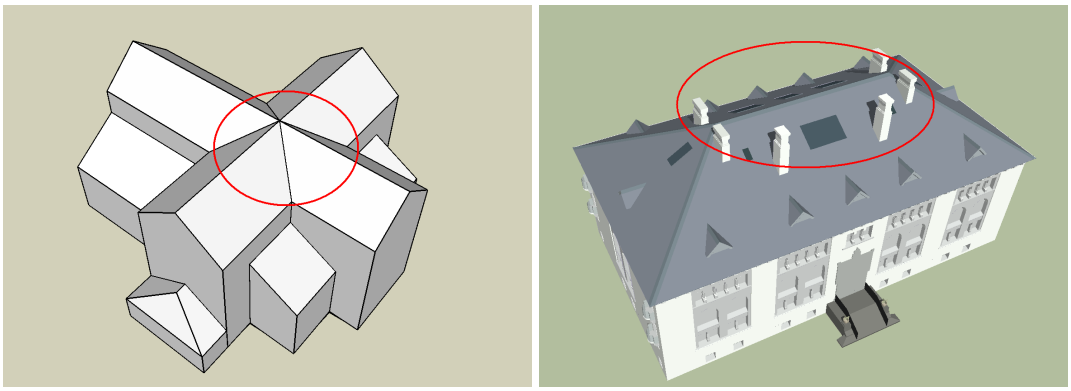


Figure 56: Examples of failed cases: (a) intersection of two extruded structures; (b) intersection of a tapered structure with an extruded structure.

On the direction of enhancing the modeling ability, we plan to handle the intersection of two structures from different directions by using more advanced segmentation techniques. This will allow us to segment those complicated structures and therefore make each individual part modeling easier. We also plan to extend the tapered structure inference to include some non-linear geometry

prototype which can be modeled as *follow-me* structures. This follow-me structure modeling is a complicated tool featured in Google SketchUp. Essentially, the follow-me models can be generated by moving a cross-sectional unit along a curve trajectory. For example, revolving a half-circle/curve around an axis generates a dome/spherical model.

Appendix A

Computational Geometry on 3D Space

The standard equation of a 3D plane is $ax+by+cz+d=0$, where the vector (a,b,c) specifies the normal of the plane. Given 3 non-collinear points in 3D space, (x_1,y_1,z_1), (x_2,y_2,z_2), (x_3,y_3,z_3), the plane defined by them can be computed by solving the following equations:

$$ax_1+by_1+cz_1+d=0$$
$$ax_2+by_2+cz_2+d=0$$
$$ax_3+by_3+cz_3+d=0$$

Solving the above equations gives

$$a=\begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} b=\begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix} c=\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} d=\begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}$$

Expanding the above equations gives

$$\begin{cases} a = y1(z2-z3)+y2(z3-z1)+y3(z1-z2) \\ b = z1(x2-x3)+z2(x3-x1)+z3(x1-x2) \\ c = x1(y2-y3)+x2(y3-y1)+x3(y1-y2) \\ d = x_1(y_3z_2-y_2z_3)+x_2(y_1z_3-y_3z_1)+x3(y_2z_1-y_1z_2) \end{cases} \tag{10}$$

Note that if the points are collinear, the normal (a,b,c) as calculated above will be (0,0,0).

Alternatively, a 3D plane can be represented using the normal form that specifies point V_0 on the plane and a normal vector $\mathbf{n}$ which is perpendicular to it. This representation can be used to compute intersections since it results in compact and efficient formulas. The normal vector $\mathbf{n}$ can be computed from the cross-product $\mathbf{n}=\mathbf{u}\times\mathbf{v}=(V_1-V_0)\times(V_2-V_0)$ as shown in the Fig. 57(a). Any point P on the plane satisfies the implicit equation:

$$\mathbf{n}\cdot(P-V_0)=0 \tag{11}$$

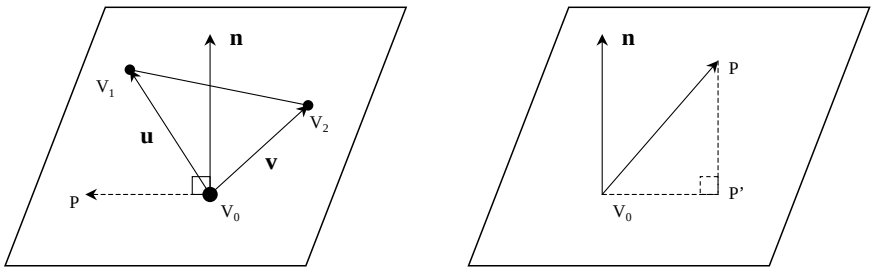


Figure 57: The point and plane in 3D space: (a) the representation of a plane; (b) the projection and distance from a point to a plane.

A.1 Distance of a point to a plane

For a plane $\Pi : ax+by+cz+d=0$ and a point $\mathbf{P}=(x_0,y_0,z_0)$ not necessarily lying on the plane, the distance can be computed by using the dot product to get the projection of the vector $(P-V_0)$ onto $\mathbf{n}$ as shown in Fig. 57(b):

$$d(P,\Pi)=|P-V_0|cos\theta=\frac{\mathbf{n}\cdot(P-V_0)}{|\mathbf{n}|}=\frac{\left|ax_0+by_0+cz_0+d\right|}{\sqrt{a^2+b^2+c^2}}. \tag{12}$$

It follows that $\mathbf{P}$ lies in the plane if and only if $d(P,\Pi)=0$.

If $\sqrt{a^2+b^2+c^2}=1$, meaning that a, b, and c are normalized then the equation becomes

$$d(P,\Pi)= |ax_0+by_0+cz_0+d|$$

A.2 The projection point of a point to a plane

There are situations where one wants to know the projection of P onto Π. For example, when we want to install windows on walls or facade, we want to know the projected points for window vertices. This can be computed by taking a line through P that is perpendicular to Π, and computing it's intersection with the plane. The perpendicular line through P is given by: $P(s)=P+s\mathbf{n}$. It intersects Π when $P(s)$ satisfies the plane equation $\mathbf{n}\cdot(P(s)-P)=0$. Solving this for the intersection point, we get:

$$P'=P-\frac{ax_0+by_0+cz_0}{a^2+b^2+c^2}\mathbf{n} \tag{13}$$

Appendix B

Mathematical Morphology

In this appendix, the basic morphological operations on binary image used in the project are described. The morphological operations are conducted on set, including the binary image A, and the structuring element B. A and B are defined on a 2D Cartesian grid, where the 1's are the elements of those sets. We denote by B_{xy} the structuring element after it has been translated so that its origin is located at the point (x,y). The output of a morphological operation is another set.

1. **Dilation**

   The dilation operation of A by the structuring element B is defined by:

   $D = A \oplus B = \{(x,y) | A \cap B_{xy} \neq \varnothing\}$

   Namely, the output binary image D of the dilating A by structuring element

B is the set of points (x,y) such that if B is translated to the origin (x,y), its intersection with A is not empty.

Essentially, simple dilation (3x3 kernel of B) is the process of incorporating into the object all the background points that touch it, leaving it larger in area by one pixel all around its perimeter. If the object is circular, its diameter increases by two pixels with each dilation. If two objects are separated by less than 3 pixels at any point, they will become connected at that point. Therefore, dilation is useful for filling holes in segmented objects.

2. **Erosion**

   The erosion operation of A by the structuring element B is defined by:

   $E = A \ominus B = \{(x,y) | B_{xy} \subseteq A\}$

   Namely, the output binary image E of the eroding A by structuring element B is the set of points (x,y) such that if B is translated to the origin (x,y), it is completely contained within A.

   Essentially, simple erosion (3x3 kernel of B) is the process of eliminating all the boundary points from an object, leaving it smaller in area by one pixel all around its perimeter. If the object is circular, its diameter decreases by two pixels with each erosion. If it narrows to less than 3 pixels thick at any point, it will become disconnected at that point. Objects with no more than two pixels thick in any direction are eliminated. Therefore, erosion is

useful for removing from a segmented image objects that are too small to be of interest, such as those noise or outlier data.

Appendix C

Hough Space Plane Detection

The Hough transform was introduced to detect complex patterns of points in binary images and became quickly a popular algorithm to detect lines and simple curves. The key idea is to map a difficult pattern detection problem into a simple peak detection problem in the space of the parameters of the curve. The Hough transform has several attractive characteristics [105]. First, it is relatively robust to noise because those outlier data points are unlikely to form consistently any geometry shapes, and therefore vote to the same bins for a particular parameter. Second, it is able to compute multiple instances of a dataset in a single pass, which outperforms most of the pattern matching algorithms based on iteratively refinement process.

The classical Hough transform [96] for line detection in images is based on

the slope-intercept formulation of a line, i.e., $y = mx + c$, where (x, y) is a point on the line, m is the slope and c is the $y-$intercept. The Hough transform proceeds by discretizing m and c and for each image point $p = (x_p, y_p)$, increasing all cells i and j which satisfy $c_i = y_p - m_j x_p$. The largest accumulator value in the Hough space gives the hypotheses for the line. However, there is a major weakness with this parameterization: because the valid range of m is from $-\infty$ to $+\infty$, which cannot be properly discretized, the detection gets much poorer as lines become vertical.

This problem for line detection can be solved by parameterizing the line using polar coordinate representation, that is $xcos\theta + ysin\theta = r$. Where θ is the angle of the normal direction with the $x-$axis and r is the perpendicular distance from the origin to the line. Using this formulation, the Hough space is still 2D but it does not have the discretization problem of the classical method described above since $0 \leq \theta < \pi$. Consequently, this formulation enables the unbiased detection of vertical lines.

The extension of the principle of classical Hough transform from 2D to 3D for plane detection is quite straight-forward. A plane is represented by its explicit equation $z = ax + by + c$, which indicates a 3D Hough space corresponding to a, b and c. This extension suffers from the same problems as its 2D counter part, that is, the near-vertical plane detection is hard to be detected reliably due to unbounded range of parameters.

The solution for unbiased planar detection in 3D is quite similar to the one for 2D. The plane is parameterized by its normal direction $\hat{\mathbf{n}}=(n_x,n_y,n_z)$ and its perpendicular distance from the origin ρ. As there is a constraint on the magnitude of the normal of the plane, i.e., $||\hat{\mathbf{n}}||=1$, there are only three degrees of freedom left. Because an unconstrained representation with the minimum number of parameters is more efficient for the Hough transform, one can use spherical coordinates of a unit sphere (θ,ϕ) for representing the unit normal $\hat{\mathbf{n}}$.

$$\hat{\mathbf{n}}=(cos\theta sin\phi \quad sin\theta sin\phi \quad cos\phi) \qquad 0\leq\theta<2\pi \quad 0\leq\phi\leq\pi \tag{14}$$

Hence, there are three parameters for 3D Hough space, which are θ,ϕ and ρ. Each given point (x_p,y_p,z_p) in the input point cloud votes for all bins θ_i,ϕ_j and ρ_k which satisfy

$$\rho_k=x_p cos\theta_i sin\phi_j+y_p sin\theta_i sin\phi_j+z_p cos\phi_j \tag{15}$$

Bibliography

[1] Pingbo Tang, Daniel Huber, Burcu Akinci, Robert Lipman, and Alan Lytle. Automatic reconstruction of as-built building information models from laser-scanned point clouds: A review of related techniques. *Automation in Construction*, 19(7):829–843, 2010.

[2] Fausto Bernardini and Holly Rushmeier. The 3d model acquisition pipeline. *Computer Graphics Forum*, 21:149–172, 2003.

[3] Leica. Leica geosystems, http://hds.leica-geosystems.com/. 2001.

[4] Google. Google earth, http://www.google.com/earth/index.html. 2004.

[5] Microsoft. Bing maps, http://www.microsoft.com/maps/. 2008.

[6] Pascal Mueller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. *ACM Transactions on Graphics (TOG)*, pages 614–623, 2006.

[7] Peter Wonka, Michael Wimmer, François Sillion, and William Ribarsky. Instant architecture. *ACM SIGGRAPH*, pages 669–677, 2003.

[8] Yoav I. H. Parish and Pascal Müller. Procedural modeling of cities. *ACM SIGGRAPH*, pages 301–308, 2001.

[9] Robert B. Fisher. Applying knowledge to reverse engineering problems. *Computer-Aided Design*, 36:501–510, 2004.

[10] U. Castellani, S. Livatino, and R. B. Fisher. Improving environment modeling by edge occlusion surface completion. *Proc. Int. Symp. on 3D Data Processing Visualization and Transmission (3DPVT)*, pages 672–675, 2002.

[11] L. D. Cohen and T. Deschamps. Multiple contour finding and perceptual grouping using minimal paths. *Journal of Mathematical Imaging and Vision*, 14:225–236, 2001.

[12] B. Okorn, X. Xiong, B. Akinci, and D. Huber. Toward automated modeling of floor plans. *Proc. Int. Symp. on 3D Data Processing Visualization and Transmission (3DPVT)*, 2010.

[13] S. Or, K. Wong, Y. Yu, and M. Chang. Highly automatic approach to architectural floorplan image understanding and model generation. *Proc. Vison, Modeling, and Visualization*, 2005.

[14] Xuetao Yin, Peter Wonka, and Anshuman Razdan. Generating 3D building models from architectural drawings: A survey. *IEEE Computer Graphics and Applications*, pages 20–30, 2009.

[15] W. Thompson, J. Owen, H. J. Germain, S. R. Stark, and T. C. Henderson. Feature-based reverse engineering of mechanical parts. *IEEE Transactions on Robotics and Automation*, 15, 1999.

[16] I. Braude, J. Marker, K. Museth, J. Nissanov, and D. Breen. Contour-based surface reconstruction using implicit curve fitting, and distance field filtering and interpolation. *In Proc. International Workshop on Volume Graphic*, pages 95–102, 2006.

[17] A. B. Konovalov and V. V. Lyubimov. High-resolution restoration of diffuse optical images reconstructed by the photon average trajectories method. *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, 5474:66–79, 2004.

[18] Alexei A. Samsonov, Eugene G. Kholmovskib, and Chris R. Johnson. Image reconstruction from sensitivity encoded mri data using extrapolated iterations of parallel projections onto convex sets. *Medical Imaging*, 5032:1829–1838, 2003.

[19] C.O.S. Sorzano, R. Marabini, G.T. Herman, and J.M. Carazo. Multiobjective algorithm parameter optimization using multivariate statistics. *Pattern recognition*, 38:2587–2601, 2005.

[20] R. Barbuzza, M. Vénere, and A. Clausse. Tomographic reconstruction using heuristic monte carlo methods. *Journal of Heuristics*, 13:227–242, 2007.

[21] F. J. Sigworth. A maximum-likelihood approach to single-particle image refinement. *Journal of Structual Biology*, pages 328–339, 1998.

[22] Jinhui Hu, Suya You, and Ulrich Neumann. Integrating lidar, aerial image and ground images for complete urban building modeling. *Proc. Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06)*, pages 184–191, 2006.

[23] Avideh Zakhor and Christian Frueh. Automatic 3D modeling of cities with multimodal air and ground sensors. *Multimodal Surveillance: Sensors, Algorithms, and Systems, Artech House, Boston*, Chapter 15, 2007.

[24] A. Akbarzadeh, J.-M. Frahm, P. Mordohai, B. Clipp, C. Engels, D. Gallup, P. Merrell, M. Phelps, S. Sinha, B. Talton, L. Wang, Q. Yang, H. Stewenius, R. Yang, G. Welch, H. Towles, D. Nister, and M. Pollefeys. Towards urban 3D reconstruction from video. *Proc. Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06)*, pages 1–8, 2006.

[25] Jianxiong Xiao, Tian Fang, Ping Tan, Peng Zhao, Eyal Ofek, and Long Quan. Image-based facade modeling. *ACM SIGGRAPH Asia*, 2008.

[26] Alexander Toshev, Philippos Mordohai, and Ben Taskar. Detecting and parsing architecture at city scale from range data. *IEEE Computer Vision and Pattern Recognition*, pages 398–405, 2010.

[27] Karim Hammoudi, Fadi Dornaika, and Nicolas Paparoditis. Extracting building footprints from 3d point clouds using terrestrial laser scanning at street level. *International Archives of Photogrammetry and Remote Sensing (IAPRS)*, pages 32–37, 2009.

[28] Matthew Johnston and Avideh Zakhor. Estimating building floor plans from exterior using laser scanners. *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, 6815:137–147, 2008.

[29] Carlos Vanegas, Daniel Aliaga, and Bedrich Benes. Building reconstruction using manhattan-world grammars. *IEEE Computer Vision and Pattern Recognition*, pages 358–365, 2010.

[30] F. Bernardini, J. Mittlelman, H. Rushmeir, and C. Silva. The ball-pivoting algorithm for surface reconstruction. *IEEE Transaction on Visualization and Computer Graphics*, 5:349–359, 1999.

[31] Michael Garland and Paul Heckbert. Surface simplification using quadric error metrics. *ACM SIGGRAPH*, 1997.

[32] Kuldeep K. Sareen, George K. Knopf, and Robert Canas. Contour-based 3d point cloud simplification for modeling freeform surfaces. *IEEE Toronto International Conference on Science and Technology for Humanity (TIC-STH)*, pages 381–386, 2009.

[33] Pablo Rodriguez Gonzalvez, Diego Gonzalez Aguilera, and Javier Gomez Lahoz. From point cloud to surface: Modeling structures in laser scanner point clouds. *ISPRS Workshop on Laser Scanning*, pages 338–344, 2007.

[34] Jan Bohm, Susanne Becker, and Norbert Haala. Model refinement by integrated processing of laser scanning and photogrammetry. *ISPRS International Workshop 3D-Arch 2007 (3D Virtual Reconstruction and Visualization of Complex Architectures)*, 2007.

[35] Susanne Becker and Norbert Haala. Combined feature extraction for facade reconstruction. *ISPRS Workshop on Laser Scanning*, pages 241–247, 2007.

[36] YouCity. Youcity social network system, http://youcity.com/. 2009.

[37] ClearEdge3D. Edgewise, http://www.clearedge3d.com/. 2006.

[38] L. Liu and I. Stamos. Automatic 3D to 2D registration for the photorealistic rendering of urban scenes. *IEEE Computer Vision and Pattern Recognition*, 2:137–143, 2005.

[39] L. Liu, I. Stamos, G. Yu, G, Wolberg, and S. Zokai. Multiview geometry for texture mapping 2D images onto 3D range data. *IEEE Computer Vision and Pattern Recognition*, 2:2293–2300, 2006.

[40] Ioannis Stamos, Lingyun Liu, Chao Chen, George Wolberg, Gene Yu, and Siavash Zokai. Integrating automated range registration with multiview geometry for the photorealistic modeling of large-scale scenes. *International Journal of Computer Vision*, 78(2-3):237–260, 2008.

[41] M. Fischler and R. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, pages 281–395, 1981.

[42] Jann Poppinga, Narunas Vaskevicius, Andreas Birk, and Kaustubh Pathak. Fast plane detection and polygonalization in noisy 3d range images. *International Conference on Intelligent Robots and Systems (IROS)*, 2008.

[43] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient ransac for point-cloud shape detection. *Computer Graphics Forum*, pages 214–226, 2007.

[44] David A. Forsyth and Jean Ponce. Computer vision - a modern approach. *Prentice Hall*, 2003.

[45] Michael Ying Yang and Wolfgang Forstner. Plane detection in point cloud data. *Technical Report, University of Bonn*, 2010.

[46] Niloy Mitra and An Nguyen. Estimating surface normals in noisy point cloud data. *special issue of International Journal of Computational Geometry and Applications*, 14(4–5):261–276, 2004.

[47] George Vosselman and Sander Dijkman. 3d building model reconstruction from point clouds and ground plans. *International Archives of Photogrammetry and Remote Sensing (IAPRS)*, pages 37–43, 2001.

[48] George Vosselman. Building reconstruction using planar faces in very high density height data. *International Archives of Photogrammetry and Remote Sensing (IAPRS)*, pages 87–92, 1999.

[49] Tahir Rabbani Shah. Automatic reconstruction of industrial installations using point clouds and images. *Publications on Geodesy 62, Delft*, pages 43–44, 2006.

[50] Zhiquan Cheng, YZ Wang, B. Li, K. Xu, G. Dang, and SY Jin. A survey of methods for moving least squares surfaces. *Proc. IEEE/EG Symp. Volume and Point-Based Graphics*, pages 21–35, 2008.

[51] David Levin. The approximation power of moving least-squares. *Mathematics of Computation*, pages 1517–1531, 1998.

[52] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Ruth Silverman. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of ACM*, 45(6):891–923, 1998.

[53] Marc Alex, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):3–15, 2003.

[54] J. Podolak, P. Shilane, A. Golovinskiy, S. Rusinkiewicz, and T. Funkhouser. A planar-reflective symmetry transform for 3D shapes. *ACM SIGGRAPH*, pages 549–559, 2006.

[55] H. Zabrodsky, S. Peleg, , and D. Avnir. Symmetry as a continuous feature. *IEEE Pattern Analysis and Machine Intelligence*, 17, 1995.

[56] S. Thrun and B. Wegbreit. Shape from symmetry. *Proc. IEEE Intl. Conf. on Computer Vision (ICCV)*, 2005.

[57] N. Mitra, L. Guibas, and M. Pauly. Partial and approximate symmetry detection for 3D geometry. *ACM SIGGRAPH*, 25:560–568, 2006.

[58] Adam Hoover, Gillian Jean-Baptiste, Xiaoyi Jiang, Patrick J. Flynn, Horst Bunke, Dmitry B. Goldgof, Kevin Bowyer, David W. Eggert, Andrew Fitzgibbon, and Robert B. Fisher. An explerimental comparison of range image segmentation algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18:673–689, 1996.

[59] Xiaoyi Jiang and Horst Bunke. Fast segmentation of range images into planar regions by scan line grouping. *Machine Vision and Applications*, 7:115–122, 1994.

[60] George Sithole and George Vosselman. Automatic structure detection in a point cloud of an urban landscape. *Proceedings of 2nd GRSS/ISPRS Joint Workshop on Remote Sensing and Data Fusion over Urban Areas*, pages 67–71, 2003.

[61] Klaus Koster and Michael Spann. Mir: An approach to robust clustering - application to range image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(5):430–444, 2000.

[62] Angel D. Sappa and Michel Devy. Fast range image segmentation by an edge detection strategy. *Proceedings of 3nd International Conference on 3-D Digital Imaging and Modeling*, pages 292–299, 2001.

[63] Inas Khalifa, Medhat Moussa, and Mohamed Kamel. Range image segmentation using local approximation of scan lines with application to cad model acquisition. *Machine Vision and Applications*, 13:263–274, 2003.

[64] Xiaoyi Jiang and Horst Bunke. Edge detection in range images based on scan line approximation. *Computer Vision and Image Understanding*, 73:183–199, 1999.

[65] Mike Heath, Sudeep Sarkar, Thomas Sanocki, and Kevin Bowyery. Comparison of edge detectors a methodology and initial study. *Computer Vision and Image Understanding*, 69:68–54, 1998.

[66] Rihua Xiang and Runsheng Wang. Range image segmentation based on split-merge clustering. *Proceedings of the 17th International Conference on Pattern Recognition*, 3:614–617, 2004.

[67] T. Rabbani, F. A. van den Heuvel, and G. Vosselman. Segmentation of point clouds using smoothness constraints. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36:248–253, 2006.

[68] Shi Pu and George Vosselman. Automatic extraction of building features from terrestrial laser scanning. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36:254–258, 2006.

[69] Jorge Hernandez and Beatriz Marcotegui. Point cloud segmentation towards urban ground modeling. *IEEE Joint Urban Remote Sensing Event*, pages 1–5, 2009.

[70] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[71] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004.

[72] Y. Haxhimusa and W.G. Kropatsch. Hierarchy of partitions with dual-graph contraction. *Proceedings of German Pattern Recognition Symposium*, pages 338–345, 2003.

[73] David Sedlacek and Jiri Zara. Graph cut based point-cloud segmentation for polygonal reconstruction. *Proceedings of the 5th International Symposium on Advances in Visual Computing*, pages 218–227, 2009.

[74] Sung Chun Lee and Ram Nevatia. Extraction and integration of window in a 3d building model from ground view images. *Proceedings of Computer Society Conference on Computer Vision and Pattern Recognition*, 2004.

[75] Hoang-Hon Trinh, Dae-Nyeon Kim, Suk-Ju Kang, and Kang-Hyun Jo. Window extraction using geometrical characteristics of building surface. *Springer-Verlag Berlin Heidelberg*, pages 585–594, 2009.

[76] Ildiko Suveg and George Vosselman. Knowledge based reconstruction of buildings. *International Conference on Image Analysis and Processing (ICIAP)*, pages 484–489, 2001.

[77] Petko Faber and Bob Fisher. How can we exploit typical architectural structures to improve model recovery? *3D Data Processing Visualization and Transmission (3DPVT)*, pages 824–833, 2002.

[78] Shi Pu and George Vosselman. Extracting windows from terrestrial laser scanning. *ISPRS Workshop on Laser Scanning*, pages 320–325, 2007.

[79] L.G. Brown. A survey of image registration techniques. *ACM Computing Surveys*, 24:326–376, 1992.

[80] Y. F. Wu, Y. S. Wong, H. T. Loh, and Y. F. Zhang. Modeling cloud data using an adaptive slicing approach. *Computer-Aided Design*, 36:231–240, 2004.

[81] B. Zitova and J. Flusser. Image registration methods: A survey. *Image and Vision Computing*, pages 977–1000, 2003.

[82] W.K. Pratt. Digital image processing. *Second Ed., Wiley, New York*, 1991.

[83] George Wolberg and Siavash Zokai. Robust image registration using log-polar transform. *International Conference on Image Processing*, pages 12–15, 2000.

[84] Paul Viola and William M. Wells. Alignment by maximization of mutual information. *International Journal of Computer Vision*, 24(2):137–154, 1997.

[85] D.P. Huttenlocher, G.A. Klanderman, and W.J. Rucklidge. Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intellinence*, 15:850–863, 1993.

[86] I. Reyer and M. Petrovtseva. System of vectorization of binary images. *Pattern Recognition and Image Analysis*, 17:204–210, 2007.

[87] S. Wu and M. R. G. Marquez. A non-self-intersection douglas-peucker algorithm. *IEEE Proc. Computer Graphics and Image Processing*, pages 60–66, 2003.

[88] Wen-Jan Chen Shih-Chang Liang. Extraction of line feature in binary images. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E91-A:1890–1897, 2008.

[89] B. Aronov, T. Asano, N. Katoh, K. Mehlhorn, , and T. Tokuyama. Polyline fitting of planar points under min-sum criteria. *International Journal of Computational Geometry and Applications*, 16:97–116, 2006.

[90] D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required for represent a digitzed line or its caricature. *Canadian Cartographer*, 10:112–122, 1973.

[91] J. Hershberger and J. Snoeyink. Speeding up the douglas-peucker line-simplification algorithm. *Proc. 5th International Symposium Spatial Data Handling*, pages 134–143, 1992.

[92] J. Hershberger and J. Snoeyink. An o(nlogn) implementation of the douglas-peucker algorithm for line simplification. *Proceedings of the tenth annual symposium on Computational geometry*, pages 383–384, 1994.

[93] P. K. Agarwal1 and K. R. Varadarajan2. Efficient algorithms for approximating polygonal chains. *Discrete and Comptational Geometry*, 23:273–291, 2000.

[94] E. Medeiros, L. Velho, and W. Lopes. Restricted bpa: applying ball-pivoting on the plane. *IEEE Proc. Computer Graphics and Image Processing*, pages 372–379, 2004.

[95] Rafael C. Gonzalez and Richard E. Woods. Digital image processing. *Prentice Hall, Second Edition*, 2002.

[96] P. V. C. Hough. Method and means for recognizing complex patterns. *U.S. Patent 3.069.654*, 1962.

[97] Ales Leonardis, Ales Jaklic, and Franc Solina. Superquadrics for segmenting and modeling range data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:1289–1295, 1997.

[98] Chao Zhang and Baozong Yuan. Representation and reconstrution of 3-d objects using nonlinear deformable. *IAPR Workshop on Machine Vision Applications*, 1992.

[99] Laurent Chevalier, Fabrice Jaillet, and Atilla Baskurt. Segmentation and superquadric modeling of 3d objects. *WSCG'03*, 11, 2003.

[100] Irving Biederman. Recognition-by-components: A theory of human image understanding. *Psychological Review*, 94:115–147, 1987.

[101] T.O. Binford. Visual perception by computer. *Proceedings of the IEEE Conference on Systems and Control*, 1971.

[102] Alan H. Barr. Superquadrics and angle-preserving transformations. *Computer Graphics and Applications*, 1:11–23, 1981.

[103] Alan H. Barr. Global and local deformations of solid primitives. *ACM SIGGRAPH*, pages 21–30, 1984.

[104] Lin Zhou and Chandra Kambhamettu. Extending superquadrics with exponent functions: Modeling and reconstruction. *Proceedings of Computer Vision and Pattern Recognition*, pages 73–78, 1999.

[105] Emanuele Trucco and Alessandro Verri. Introductory techniques for 3-d computer vision. *Prentice Hall*, 1998.