3D URBAN MODELING FROM CITY-SCALE AERIAL LIDAR DATA

by

Qian-Yi Zhou

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

April 2012

Acknowledgments

I would like to express my sincere thanks to my advisor Prof. Ulrich Neumann. His insightful guidance, inspiring advice, and scientific knowledge is an invaluable treasure for me during my five years' study in the University of Southern California. I had a great time at the Computer Graphics and Immersive Technologies lab, and I hope our cooperation will continue in the future.

I would like to thank Prof. Suya You, Prof. C.-C. Jay Kuo, Prof. Jernej Barbič, and Prof. Aiichiro Nakano for taking their precious time to serve on the committee of my qualifying exam and thesis defense. I am very grateful to the people I have worked with or shared knowledge with. Special thanks to Prof. Shi-Min Hu from Tsinghua University, Prof. Tao Ju from Washington University in St. Louis, Thommen Korah from Nokia Research Center, Florent Lafarge from INRIA, and the current and previous CGIT lab members: Rongqi Qiu, Kelvin Chung, Luciano Nocera, Taehyun Rhee, Charalambos 'Charis' Poullis, Jonathan Mooser, Seon-Min Rhee, Lu Wang, Quan Wang, Sang Yun Lee, Tanasai Sucontphunt, Zhenzhen Gao, Wei Guan, Guan Pang, Yi Li, Jing Huang, Jiaping Zhao, and Yili Zhao. I acknowledge Airborne 1 Corp., Sanborn Corp., and Chevron Corp. for providing data, and USC Provost's PhD fellowship, Lockheed Martin Corp., and Nokia Corp. for financial support.

I am grateful to the support from my parents in China. I thank all my friends at USC for making my life here a happy and precious experience: Jing Zhao, Fan Sun, Chang Huang, Rui Wang, Xiaoxun Sun, and many others.

Finally, a very special thanks to my fiancee Na Chen, a woman who is excellent in all aspects, beautiful, smart, and generous. I am very grateful for her encouragement, her patience, and all her love and tenderness.

Table of Contents

List of Tables

List of Figures

Abstract

3D reconstruction from point clouds is a fundamental problem in both computer vision and computer graphics. As urban modeling is an important reconstruction problem that has various significant applications, this thesis investigates the complex problem of reconstructing 3D urban models from aerial LiDAR (Light Detection And Ranging) point cloud.

In the first part of this thesis, an automatic urban modeling system is proposed which consists of three modules: (1) the classification module classifies input points into trees and buildings; (2) the segmentation module splits building points into different roof patches; (3) the modeling module creates building models, ground, and trees from point patches respectively. In order to support city-scale data sets, this pipeline is extended into an out-of-core streaming framework. By storing data as stream files on hard disks and using main memory as only a temporary storage for ongoing computation, an efficient out-of-core data management is achieved. City-scale urban models are successfully created from billions of points with limited computing resource.

The second part of this thesis explores the 2.5D nature of building structures. The 2.5D characteristic of building models is observed and formally defined as "building structures are always composed of complex roofs and vertical walls". Based on this observation, a 2.5D geometry representation is developed for the building structures, and used to extend a classic volumetric modeling approach into a 2.5D method, named 2.5D

dual contouring. This algorithm can generate building models with arbitrarily shaped roofs while keeping the verticality of the walls. The next research studies the topology of 2.5D building structures. 2.5D building topology is formally defined as a set of roof features, wall features, and point features; together with the associations between them. Based on this research, the topology restrictions in 2.5D dual contouring are relaxed. The resulting model contains much less triangles but similar visual quality. To further capture the global regularities that intrinsically exist in building models because of human design and construction, a broad variety of global regularity patterns between 2.5D building elements are explored. An automatic algorithm is proposed to discover and enforce global regularities through a series of alignment steps, resulting in 2.5D building models with high quality in terms of both geometry and human judgement. Finally, the 2.5D characteristic of building structures is adopted to aid 3D reconstruction of residential urban areas: a more powerful classification algorithm is developed which adopts an energy minimization scheme based on the 2.5D characteristic of building structures.

This thesis demonstrates the effectiveness of all the algorithms on a range of urban area scans from different cities; with varying sizes, density, complexity, and details.

Chapter 1

Introduction

Three dimensional urban models are very useful in a variety of applications such as urban planning, virtual city tourism, surveillance, disaster simulation, and computer games. Most of these applications require city-scale urban models composed of simple and elegant building models, tree models, and ground.

The advance of acquisition techniques has made aerial LiDAR (Light Detection And Ranging) data a powerful 3D representation of urban areas. Equipped on aeroplanes, laser scanners are able to capture the surface geometry of large cities in an accurate and efficient manner. Billions of LiDAR points are collected for city-scale data sets. These data sets usually contain every detail in an urban area including both important features (*e.g.*, detailed rooftops) and trivial details (*e.g.*, residual sensor noise, undesired vegetation, and vehicles). In addition, the magnitude of LiDAR data limits its application.

The objective of this research is to reconstruct 3D urban models from aerial LiDAR data. Specifically, the modeling approach desire following properties:

- **Automation:** Urban models should be generated in an automatic manner. Few user interactions are introduced only when necessary.
- **Discrimination:** The method should be able to remove undesired features while preserving important elements, *e.g.*, buildings, and terrain.
- **Efficiency:** The method should create city-scale urban models from huge data sets within reasonable time and space.

- **Seamlessness:** The urban model should be generated in a seamless manner, meaning that the LiDAR data should be processed as one undivided piece.

- **Universality:** The method should be able to process LiDAR data sets scanned from different cities. Data sets with different density and scales can be handled by the same framework with only a few parameters configured.

- **High-quality outputs:** The method should produce simple polygonal models fitting the input point cloud in a precise manner. Since buildings are the key component in urban areas, there are three specific requirements for the building models.

    1. The reconstructed building models should contain small amount of vertices and triangles, to enable applications such as efficient rendering.
    2. Building models should fit the observation (*e.g.*, the raw scan data and the aerial imagery) in a precise manner.
    3. Building models should be visually convincing in terms of human judgement, because humans are the ultimate judge for model quality in most applications.

The first part of this thesis proposes a general 3D urban modeling system towards these requirements. This system contains four main modules, namely, classification, segmentation, building modeling, and terrain modeling. Given a LiDAR point cloud as input, irrelevant features such as trees and noises are first classified from roof and ground points, and removed in subsequent processing. A segmentation algorithm then takes the remaining point cloud, and extracts individual buildings from terrain. These point patches are sent into a building modeling module and a terrain modeling module respectively to create elegant polygonal models.

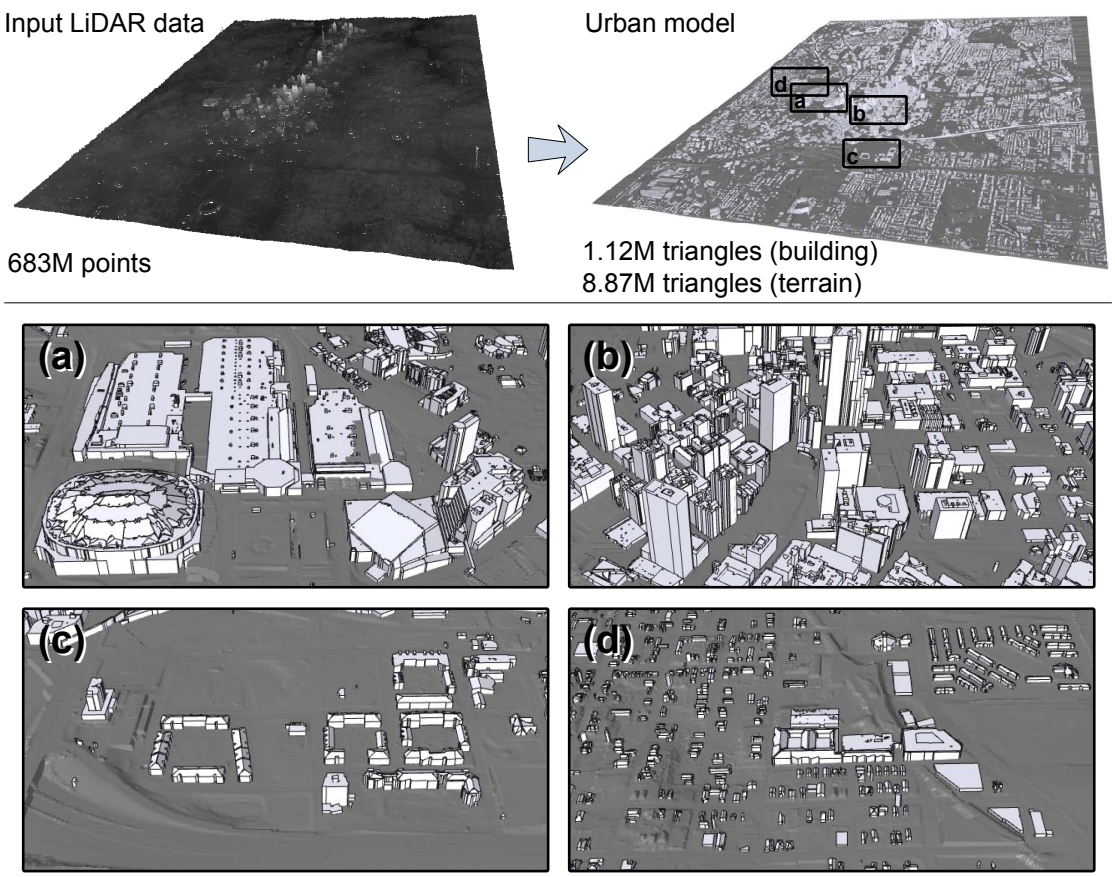


Figure 1.1: 3D reconstruction for the city of Atlanta from 683M LiDAR points, using the streaming urban modeling system presented in Chapter 4. Bottom: four closeups of the urban model, showing (a) an area with large flat structures; (b) a downtown area; and (c,d) two residential areas.

In order to handle extremely large date sets in a seamless manner, the general urban modeling pipeline is extended to an out-of-core execution architecture. The out-of-core architecture benefits from a *streaming* process which utilizes free hard-disk space as main storage while allocating main memory only as temporary space to store data for ongoing computation. In other words, the streaming process takes data as a stream (usually, a disk file) - each system component reads from an input stream, loads necessary data in-core, processes it; and once data is no longer needed for further processing, it is written into an output stream.

This automatic system is tested on four different data sets to show the universality. Urban models are automatically generated from aerial LiDAR data. The largest data set that has been processed is Atlanta LiDAR data containing 683M points stored in a 17.7GB disk file. The modeling system generates 1.12M triangles to represent the buildings and 8.78M triangles for terrain, by 25 hours of unattended processing using less than 1GB memory. The resulting urban model is shown in Figure 1.1. As illustrated in the closeups, different types of building models are created successfully over the entire urban area.

Based on this urban modeling system, the second part of this thesis studies the more fundamental problem about the 2.5D nature of building structures. As shown in Figure 1.2, the LiDAR sensor captures the details of roof surfaces, but collects few points on building walls connecting roof boundaries; in addition, users desire building models composed of complex roofs and vertical walls connecting different roof layers. Both the input and the objective of the building modeling problem show a 2.5D characteristic, caused by the 2.5D nature of building structures due to human design and construction.

This 2.5D characteristic of building structures inspires a geometry representation of polygonal building models, which forces the verticality of wall polygons. A robust approach is developed in this thesis to create 2.5D building models from aerial LiDAR point clouds, named 2.5D dual contouring. This method is guaranteed to produce crack-free models composed of complex roofs and vertical walls connecting them. By extending classic dual contouring into a 2.5D method, building geometry is simultaneous optimized over the three dimensional surfaces and the two dimensional boundaries of roof layers. Thus, 2.5D dual contouring can generate building models with arbitrarily shaped roofs while keeping the verticality of connecting walls. An adaptive grid is introduced to simplify model geometry in an accurate manner. Sharp features are detected and preserved by a novel and efficient algorithm.

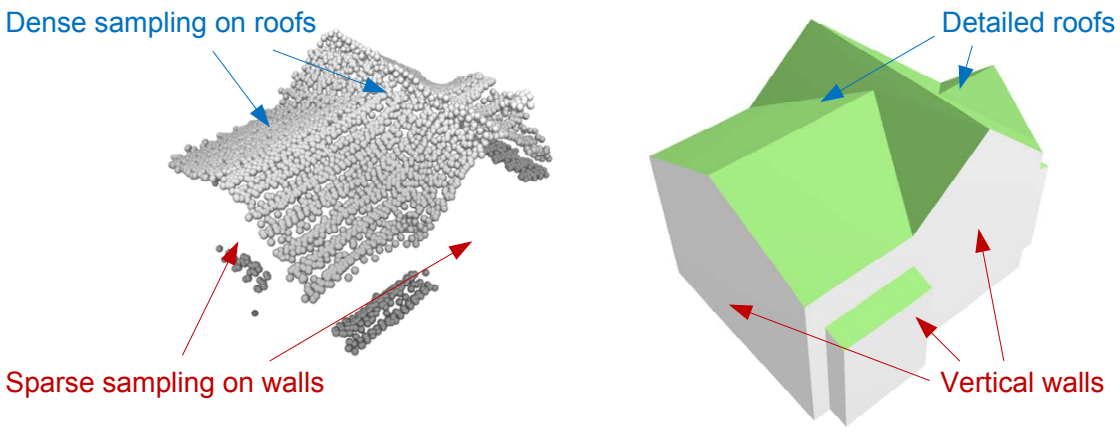


Figure 1.2: 2.5D nature of the building modeling problem. Left: aerial LiDAR has dense sampling on rooftops but very sparse sampling on building walls. Right: desired building models are usually composed of detailed roof structures and vertical walls.

While the building geometry describes where the building structures appear in the three dimensional space, the building topology determines the existence of structural pieces and the connections between them. My further research reveals that human vision tend to be very sensitive in topology changes even the related structural pieces are small. Therefore, 2.5D building topology is explored and defined as a set of roof features, wall features, and point features; together with the associations between them. Based on this definition, 2.5D dual contouring is extended into a 2.5D modeling method with topology control. Comparing with 2.5D dual contouring, the new modeling method put less restrictions on the adaptive simplification process. The reconstruction results preserve significant topology structures while the number of triangle is comparable to that of manually created model or primitive-based models.

Besides the 2.5D characteristic of building structures, other building natures due to human design and construction are further explored. They appear in the form of global regularities that encodes the orientation and placement similarities between planar elements in building structures. An automatic modeling algorithm is designed based on the

study of 2.5D global regularities. This algorithm simultaneously detects locally fitted plane primitives and global regularities: while global regularities are extracted by analyzing the plane primitives, they adjust the planes in return and effectively correct local fitting errors. By aligning planar elements to global regularities, the modeling method significantly improves the reconstruction quality in terms of both geometry and human judgement.

When the urban modeling is applied to residential areas, rich vegetation becomes an important urban feature, and thus should be carefully handled. Therefore. a robust classification algorithm is proposed in this thesis, which effectively classifies LiDAR points into trees, buildings, and ground. The classification algorithm adopts an energy minimization scheme based on the 2.5D characteristic of building structures: buildings are composed of opaque skyward roof surfaces and vertical walls, making the interior of building structures invisible to laser scans; in contrast, trees do not possess such characteristic and thus point samples can exist underneath tree crowns. Once the point cloud is successfully classified, the system reconstructs buildings and trees respectively, resulting in a hybrid model representing the 3D urban reality of residential areas.

The rest part of this thesis is organized as follows: Chapter 2 summarizes the related literature to the urban modeling problem. Chapter 3 details a general urban modeling pipeline that is extended into an out-of-core streaming architecture in Chapter 4. The research based on the 2.5D nature of building structures include: 2.5D geometry representation and 2.5D dual contouring in Chapter 5; 2.5D building topology research in Chapter 6; 2.5D building modeling with global regularities in Chapter 7; and residential urban area reconstruction in Chapter 8. The conclusion and future work is finally presented in Chapter 9.

Chapter 2

Related Work

This chapter briefly reviews the related work in the scope of 3D urban reconstruction from aerial LiDAR data. Previous research related to each specific topic is reviewed individually in Section 4.1 for streaming approaches, Section 5.1 for classic dual contouring, Section 6.1 for topology control in volumetric modeling, Section 7.1 for shape-from-symmetry methods, and Section 8.1 for tree detection and modeling algorithms.

Pioneer building modeling methods [1, 11, 22, 49, 50, 64] start by converting LiDAR point cloud into a DEM (Digital Elevation Model), and then apply image processing algorithms on these depth images to detect building footprints, fit parametric models and reconstruct polygons. All of them share a similar building reconstruction pipeline with three major steps: classification, segmentation, and building modeling.

Most existing research work is built upon this pipeline and advances the reconstruction quality by improving individual steps. Früh *et al.* [18] create building models with facade by integrating aerial LiDAR and ground based LiDAR. Hu *et al.* [24] employ aerial image and ground images to achieve a high resolution solution. Wang *et al.* [62] concentrate on building footprint extraction problem. Verma *et al.* [60] propose a roof-topology graph to find complex roof patterns from aerial LiDAR data. Lafarge *et al.* [33] present a two-stages method which can find optimal configuration of parametric models via a RJMCMC sampler. Matei *et al.* [41] specialize segmentation for densely built areas. Zebedin *et al.* [65] detect planes and surfaces of revolution. Poullis and You [48] create simple 3D models by simplifying boundaries of fitted planes. Toshev *et al.* [58] propose parse trees as a semantic representation of building structures. Lafarge and

Mallet [34] combine primitives and a general mesh representation to achieve hybrid reconstruction.

Since building models are usually considered to be the most important component in urban areas, these research efforts have an emphasis on building reconstruction. In general, most of these efforts attack the building modeling problem with primitive fitting approaches. Specifically, local plane fitting is a popular strategy in extracting simple roof primitives [41, 48, 60]. Strong urban priors are frequently introduced to restrict the plane primitives. Typical priors include roof topology [60] and Manhattan-world grammars [41, 48]. Other research work aims at extending the ability of representing complicated shapes, by introducing additional primitive shapes and optimizing junctions between fitted primitives [33, 34, 65].

Another branch of the urban reconstruction problem lies in vegetation detection, which is believed to be a difficult task in the modeling systems. Several research efforts [4, 5, 34, 39, 52, 58, 60, 62] address this problem. In general, these efforts employ both geometry and reflection properties at each LiDAR sample point, and use a general classifier to assign each point a class label indicating its category. Typical classifiers include simple thresholding [60], SVM classifiers [52], or Adaboost classifiers [39, 62].

Although automatic solutions have been provided for various LiDAR data sets, to the best of our knowledge, none of them can process an extremely large LiDAR data set in a seamless manner. Instead, many of them (*e.g.* [41, 48]) partition huge LiDAR data into tiles, process them one at a time, and merge the partial results together to generate the aggregate model of a large scale city area. Artifacts can occur at seams even with padding between tiles.

Another limitation of these methods is that they are usually designed and tested for one specific data set. Prior knowledge learnt from this input data set is integrated into

the algorithm. Thus they lose universality and may not perform well when dealing with other data sets.

Finally, this thesis is the first to discover and formally define the 2.5D characteristic of building structures. Theories and algorithms are developed based on the 2.5D nature of buildings. The urban modeling results in this thesis are competitive in terms of both geometry fitting quality and human visual judgement.

Chapter 3

General Urban Modeling Approach

This chapter presents a fully automatic system which creates simple and elegant urban models from aerial LiDAR data. The system takes only the aerial LiDAR point cloud as input and works directly on the raw data without rasterizing it into a DEM (Digital Elevation Model). The output contains two parts: a polygonal model representing the terrain, and a set of simple and elegant building models. In a local area, roof boundaries are aligned together as much as possible to improve the quality of the building models.

Section 3.1 proposes a general urban modeling pipeline, followed by the details of each module in subsequent sections. Section 3.6 shows experiments on a couple of small data sets chopped from different cities. These data sets contain small amount of points, thus can be loaded in-core and processed at once.

3.1 Urban Modeling Pipeline

This section proposes an automatic urban modeling pipeline which sequentially executes four individual processes as illustrated in Figure 3.1:

1. **Classification:** A classification module classifies vegetation and noise points from building and ground points. The detected vegetation and noise points (green and black points in the second sub-figure in the top row) are considered as irrelevant parts and thus are removed in subsequent processing.

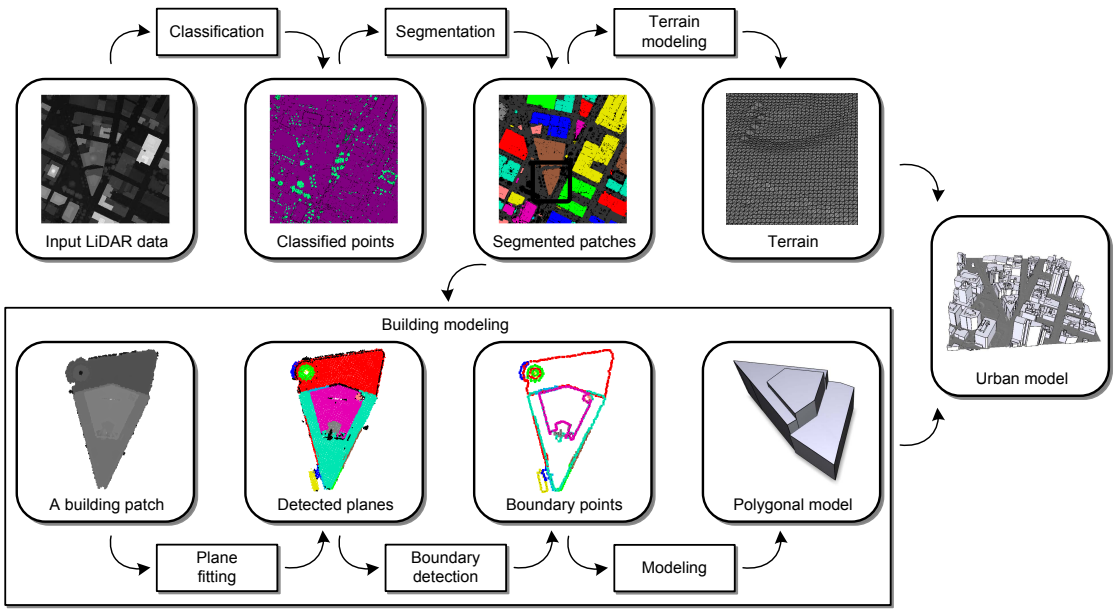


Figure 3.1: The general urban modeling pipeline includes four modules: classification, segmentation, building modeling, and terrain modeling. In particular, the building modeling module takes an individual point patch as input; detects plane patterns; extracts boundaries of these planes; and finally create polygonal building models based on the boundaries.

2. **Segmentation:** Using a distance-based region growing approach, the remaining building and ground points are segmented into individual patches (points with different colors in the third sub-figure in the top row).

3. **Building modeling:** Mesh models are created for each individual building patch, following three steps:

   (a) *Plane fitting:* Plane patterns are detected and fitted from the point cloud.

   (b) *Boundary detection:* A watertight boundary is extracted for each plane detected in the previous step.

(c) *Modeling:* Mesh models are created from the boundary points. The roof boundaries are simplified and neighboring boundary edges are snapped together to reduce the cracks in the final result.

4. **Terrain modeling:** Taking the ground point patch as input, the terrain modeling module generates a polygonal model with rasterization. Holes caused by occlusion are filled by solving a Laplace's equation.

To further improve the reconstruction quality of building models, I adopt the assumption that the roof boundary edges usually fall into a set of *principal directions*. An efficient method is proposed to learn the principal direction set from original data, and align roof boundary segments along these directions automatically. This mechanism enables arbitrary angles between neighboring roof boundary segments. In contrast, many previous methods are limited by the pre-assumptions made on these angles, *e.g.*, $90°$, $45°$ and $135°$ in [62].

Another extension to this pipeline is the non-planar roof extension. In an industrial site which contains several oil tins and tanks, the geometry of these models is in the form of cylinders and cones. Thus, the user has the option to determine the shape of each object via minimal interactions. RANSAC [14] is applied to estimate the parameters for the pre-defined primitives.

3.2 Vegetation Detection

Vegetation, mainly in the form of trees, is an irrelevant part in most downtown urban areas, thus should be eliminated in the first step. This section presents a novel classification algorithm for vegetation detection.

3.2.1 Features

Given a sample point in the raw LiDAR point cloud $\mathbf{p} \in P$, $N_p = \{\mathbf{q} | \mathbf{q} \in P, \|\mathbf{p} - \mathbf{q}\| < \delta\}$ denotes the set of points within a small neighborhood of point $\mathbf{p}$, and $\overline{\mathbf{p}}$ is the centroid of all points in N_p.

Note that if $\mathbf{p}$ lies in an extremely sparse area, it loses its geometry significance. Therefore, $\mathbf{p}$ is considered as a *noise* point. Specifically, $\mathbf{p}$ is detected as a noise point *iff* $|N_p| < \kappa$, where κ is a user-given threshold, empirically, equals to 10.

For a sample point $\mathbf{p}$ with enough points within its neighborhood N_p, five features are defined based on the local differential geometry properties:

1. **Regularity:** It measures if the point distribution around $\mathbf{p}$ is regular by calculating

$$\mathcal{F}_1 = \|\mathbf{p} - \overline{\mathbf{p}}\|. \tag{3.1}$$

   Intuitively, sample distribution around a roof point is more likely to be regular. Thus the distance between a roof point and its neighbors' centroid should be smaller than that of a vegetation point.

2. **Horizontality:** Since aerial LiDAR data captures roof from a top view, the normal at a roof point is more likely to be vertical. The second feature is defined as

$$\mathcal{F}_2 = 1 - |\mathbf{n}_p \cdot \mathbf{e}_z|, \tag{3.2}$$

   where $\mathbf{e}_z = (0, 0, 1)$ is the vertical direction, and $\mathbf{n}_p$ is obtained through covariance analysis [47], *i.e.*, to solve the eigenvector problem for the covariance matrix:

$$\mathbf{C}_p = \frac{1}{|N_p|} \sum_{\mathbf{q} \in N_p} (\mathbf{q} - \overline{\mathbf{p}})(\mathbf{q} - \overline{\mathbf{p}})^T. \tag{3.3}$$

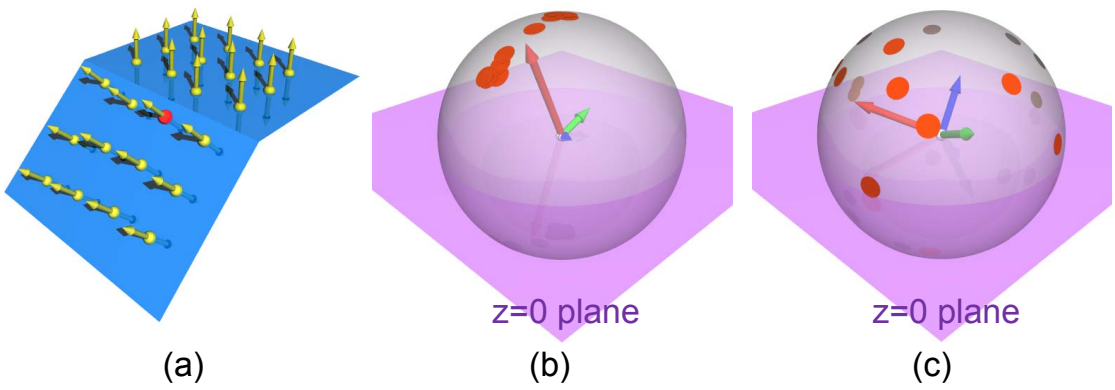


Figure 3.2: Illustration of normal variation measurements: (a) normals of points around a roof ridge; (b) normal distribution on a Gauss sphere, red/green/blue arrows point to the eigenvectors of $\mathbf{C}_p^n$ with lengthes equal to corresponding eigenvalues; (c) normals of points in the neighborhood of a tree point. Both λ_1^n and λ_2^n are large due to the irregularity of normals.

The three eigenvalues are sorted in ascending order: $\lambda_0 \leq \lambda_1 \leq \lambda_2$. The eigenvector $\mathbf{v}_0$ corresponding to the smallest eigenvalue is the estimated normal at point $\mathbf{p}$.

3. **Flatness:** With covariance analysis results, the flatness at this point, also known as *surface variation*[47], could be estimated as:

$$\mathcal{F}_3 = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2}. \tag{3.4}$$

Similar to the previous features, a smaller flatness value indicates more possibility for a point to be a roof point.

4. **Normal distribution:** Once the normal at each point is estimated, another covariance analysis can be further applied over these normals. The solution of the

eigenvector problem for a normal covariance matrix within a larger neighborhood $N_p^n = \{\mathbf{q}|\mathbf{q} \in P, \|\mathbf{p} - \mathbf{q}\| < \eta\}$:

$$\mathbf{C}_p^n = \frac{1}{|N_p^n|} \sum_{\mathbf{q} \in N_p^n} \mathbf{n}_q^T \cdot \mathbf{n}_q, \tag{3.5}$$

results in eigenvalues $\lambda_0^n \leq \lambda_1^n \leq \lambda_2^n$ with corresponding eigenvectors $\mathbf{v}_0^n, \mathbf{v}_1^n, \mathbf{v}_2^n$. As Garland [20] and Pauly [47] point out, λ_1^n measures the maximum variation of these normals on the Gauss sphere, thus could be regarded as a kind of *normal variation*. Therefore, one of the normal distribution features is defined as:

$$\mathcal{F}_4 = \lambda_1^n. \tag{3.6}$$

Apparently, a roof point prefers a smaller normal variation than a tree point.

Moreover, according to principal component analysis [30], the smallest eigenvalue λ_0^n measures the minimum normal variance, with its eigenvector $\mathbf{v}_0^n$ being the direction on Gauss sphere along which normals spread out the least. Specifically, λ_0^n determines whether the normals spread in a narrow band area on the Gauss sphere, or scatters irregularly. This property is particularly useful for sharp roof features at planar roof patch intersections. For example, Figure 3.2 shows a roof ridge broadly existing in urban area. Normals of these points form two clusters on the Gauss sphere. Hence λ_1^n (illustrated as the length of the green vector in Figure 3.2(b)) is large while λ_0^n (length of the blue vector in Figure 3.2(b)) is very small. By contrast, normal distribution around a tree point is fairly irregular and exhibits large λ_0^n and λ_1^n (shown in Figure 3.2(c)). Therefore, the last feature is defined as:

$$\mathcal{F}_5 = \lambda_0^n. \tag{3.7}$$

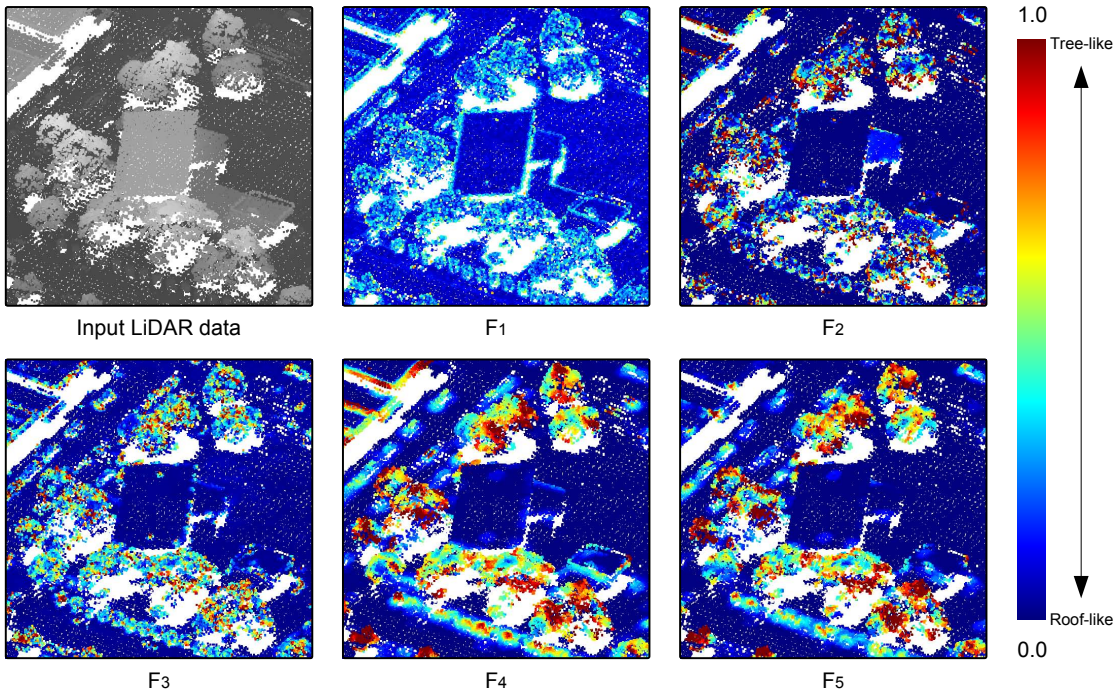


Figure 3.3: Feature values of a small piece of aerial LiDAR point cloud. The top-left sub-figure is rendered with z-values as grey scales at points. Others are rendered with colors representing corresponding feature values of the points. Blue points are more likely to be roof/ground, while red points are more likely to be trees.

Figure 3.3 demonstrates these feature values of an example urban area. The top-left sub-figure shows the input LiDAR data colored with its height value. The rest sub-figures show the five features using point colors respectively. Blue color denotes a small feature value (thus, roof-like); while red color represents a large feature value (tree-like). The five features successfully distinguish vegetation points and building/ground point even when they have similar height.

3.2.2 Classifier

Despite the removed noise points, two classes remains in this classification problem, *i.e.*, trees and building/ground. Thus, a linear classification hyperplane in the 5D feature

space is introduced to split the two classes of point samples. In particular, a discriminative function is defined as the linear combination of these five features:

$$\mathcal{K} = \omega_0 + \omega_1\mathcal{F}_1 + \omega_2\mathcal{F}_2 + \omega_3\mathcal{F}_3 + \omega_4\mathcal{F}_4 + \omega_5\mathcal{F}_5, \tag{3.8}$$

where $\omega_{0,1,2,3,4,5}$ are undetermined parameters. Once these parameters are determined, the classification algorithm simply computes $\mathcal{K}$ for each point and determines its category with $sgn(\mathcal{K})$.

To automatically determine these parameters, supervised machine learning methods are introduced which learn these parameters from a small but typical urban area with manual labeling. In particular, any linear classifier can serve as the learning module, including Linear Discriminant Analysis [15] and Support Vector Machine [2, 8]. For efficiency and accuracy, an unbalanced soft margin Support Vector Machine algorithm is adopted as proposed in [8]. The Support Vector Machine algorithm finds a hyperplane that maximizes the margin between the two classes. Due to the mislabeled examples and noise samples in the classification problem, there may exist no hyperplane that can strictly separates the two classes, thus a soft margin extension is introduced to split the examples as cleanly as possible, while still maximizing the margin distance. The soft margin Support Vector Machine algorithm is detailed in [8] and a third-party library *SVM Light* [29] is employed in the system implementation.

To further improve the classification results, a voting algorithm is introduced as a post-processing step. Intuitively, points of a same category usually occur together in the space. Thus, once all the points are labeled by the classifier, point $\mathbf{p}$'s final label is determined by voting from $\mathbf{p}$'s neighboring points. That is, $\mathbf{p}$ belongs to tree category only if the percentage of tree points in $\mathbf{p}$'s neighborhood N_p is greater than a threshold ω. This threshold is also learnt from the labeled data set.

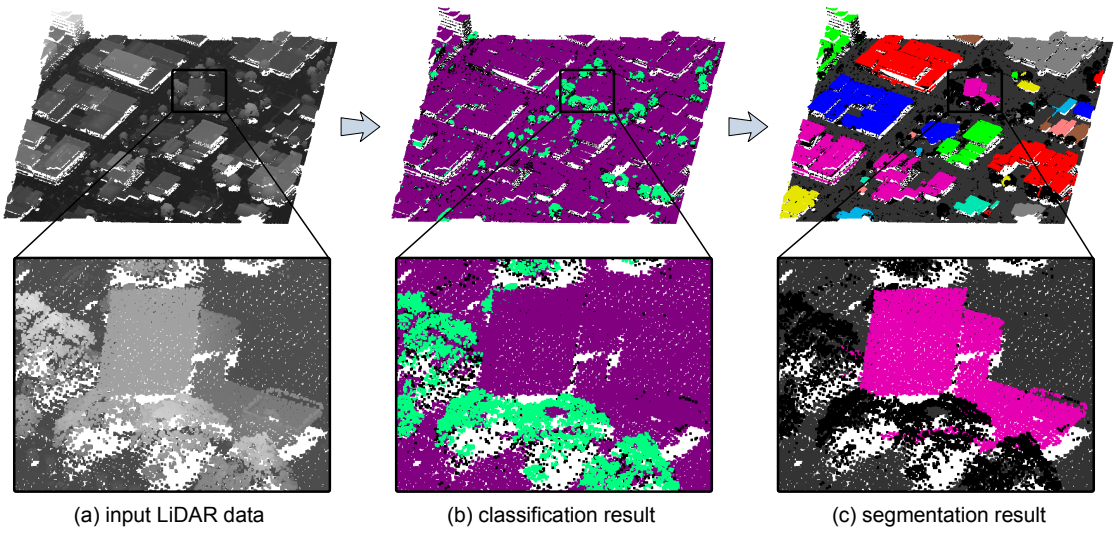


Figure 3.4: (a) Input LiDAR point cloud with color intensity representing the height at each point. (b) Green points are detected as trees. (c) Segmentation result; with trees and noises (black points) removed from the input, the segmentation algorithm splits building roofs from the ground successfully.

3.2.3 Configuration and Results

In this classification algorithm, all the features are designed based on the local differential geometry properties. Without introducing data-set-related variants such as absolute height and intensity, the proposed method achieve significant adaptivity to different data sets. In the experiments, all the seven parameters $\omega_{0,1,2,3,4,5}$ and ω are learnt from a $100m \times 100m$ labeled area from the city of Oakland. Once they are determined, they work for all the testing data sets.

The only parameters that need to be configured for each data set are the sizes of the neighborhood, *i.e.*, δ and η. Since the neighborhood should be large enough to support an effective covariance analysis, δ is set to the value which makes the average point number in N_p between $12 \sim 20$. As for η, since normals are estimated from points in N_p and thus are smoothed in some sense; the size of N_p^n should be larger than that of N_p to gain sufficient geometric significance. Thus, η is two times the value of δ.

The classification algorithm achieves an accuracy above 95% on all testing data sets. Figure 3.4 shows a $300m \times 300m$ urban piece chopped from Oakland data set. Trees and non-trees are correctly classified even at areas where points from both categories have similar heights, as shown in the closeup of Figure 3.4(b).

The weights $\omega_{1,2,3,4,5}$ are learnt with a Support Vector Machine algorithm, and determined once and for all. In the experiments, they are:

$$\omega_{1,2,3,4,5} = \{2.5, \quad 0.1, \quad 1.7, \quad 5.2, \quad 20.4\}. \tag{3.9}$$

First, since all the weights are positive, each feature has a positive contribution to the linear classifier. This observation concurs with human intuition. Second, this result suggests that the normal variations appear to be the most important factor in the classification problem, especially the minimal eigenvalue of the normal covariance matrix, *i.e.*, $\mathcal{F}_5$.

3.3 Segmentation

Once trees and noises are eliminated from the LiDAR data, the next task is to identify ground points and segment different building patches. This is achieved by a distance-based segmentation algorithm based on the following criteria: a pair of points with their distance smaller than a user-given threshold α are assigned to the same segment. With non-overlapping point segments generated from building and ground points, the largest patch is considered to be the *ground* patch. The rest point patches are identified as different roof patches. Roof patches that have neighboring projections on the x-y plane are considered to belong to the same building structure, and thus are merged into one building patch.

An example of the segmentation approach is illustrated in Figure 3.4(c). Tree points and noise points are rendered with black color and removed from the input beforehand. The segmentation algorithm splits the remaining points into several patches, with the largest patch identified as ground and rendered with dark-grey color. The rest patches are grouped together to form individual building roofs and rendered with different bright colors in the figure. Since trees and points on the vertical walls (usually detected as noise) are eliminated previously, there always exists a height gap between building roofs and ground, which leads to a correct result of distance-based segmentation.

3.3.1 Region Growing Implementation

The distance-based segmentation algorithm can be easily realized by a region growing implementation. Starting from an unlabeled seed point $\mathbf{p}$, the region growing algorithm searches its α-neighborhood N_p^α and assigns all the points within N_p^α to the same cluster as $\mathbf{p}$, denoted as C_p. Similar process is applied over all points in C_p repeatedly until the cluster can grow no more. Then the segmentation algorithm shifts to the next unlabeled point $\mathbf{q}$ and grows it into a cluster C_q. This algorithm runs iteratively until every point has a cluster label.

3.3.2 Agglomerative Clustering Algorithm

To improve time efficiency, an agglomerative clustering algorithm is introduced using the a union-find implementation [7]. The agglomerative clustering algorithm executes in a bottom-up manner. It starts with each point as an individual cluster (or segment), then traverses all the point pairs with point distance smaller than α, and merges the clusters to which the two points belong.

The *union-find* implementation utilizes a disjoint-set forests data structure as illustrated in Figure 3.5. Each tree in the forest stores the points in a same cluster. Each point

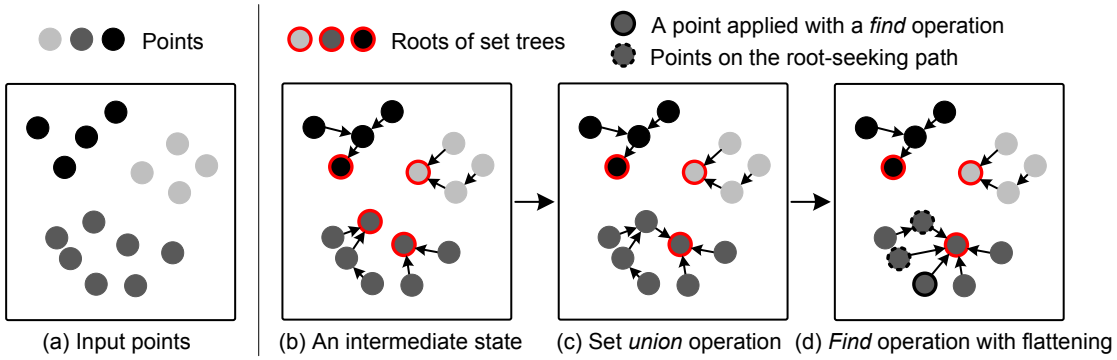


Figure 3.5: Agglomerative segmentation implemented using the union-find algorithm presented in [7]

$\mathbf{p}$ holds a reference $r(\mathbf{p})$ pointing to its parent node in the tree. The segment merging is conveniently implemented by the *union* operation, which simply assigns one root as the parent of another root, as shown in Figure 3.5(c).

To accelerate the process of retrieving root point for each cluster, the *find* operation includes a flatten process which links each point on the root-seeking path directly to the root, as shown in Figure 3.5(d). This flatten process is especially useful in the streaming segmentation to avoid dangling pointers, as detailed in Section 4.3.3.

3.4 Building Modeling

This section presents the building modeling algorithm that generates a simple and elegant mesh model from each individual building point patch. As shown in the bottom sub-figure of Figure 3.1, the building modeling approach executes three steps, namely, planar roof patch extraction, boundary detection, and model creation. The details of these steps will be presented in the following sections. A non-planar roof extension with user interaction is proposed in Section 3.4.4.

3.4.1 Planar Roof Patch Extraction

Given a building point cloud, the first task in building modeling is to extract planar roof patches from the input. As suggested by Verma *et al.* [60], this task can be achieved using a similar distance-based segmentation as that proposed in Section 3.3, with distance defined based on the inner product of point normals instead of Euclidean distance. That is, as point $\mathbf{q}$ is in point $\mathbf{p}$'s neighborhood N_p, they are neighboring only if they satisfy: $1 - |\mathbf{n}_p \cdot \mathbf{n}_q| < \beta$. Empirically, any β between $1 - \cos(5°)$ and $1 - \cos(10°)$ works well.

However, in practice, this method may lose robustness when dealing with large and smooth curved surfaces, such as the roofs of stadiums. In these cases, the whole curved surface is detected as one planar patch, which conflicts with human intuition. Thus, an iterative method is adopted to solve the plane fitting problem robustly. The general idea is similar to the EM algorithm [17] .

Starting from a plane l determined by the position and normal of an unlabeled seed point $\mathbf{p}$, the plane fitting algorithm iteratively executes:

- **E-step:** A point set L is determined as the set of unlabeled points with Euclidean distances to l smaller than a given threshold t, and normal differences less than β;
- **M-step:** Plane l is updated using least squares fitting with respect to point set L.

A planar roof patch is detected when the algorithm converges or has iterated enough times. This algorithm is applied iteratively until each point is assigned to a plane. The labeling of points around boundaries between these plane patches are further refined using a k-means-like algorithm named the k-proxy clustering, as proposed in [6].

With small patches removed as insignificant features, the remaining planar roof patches are considered as important elements in creating building polygons.

3.4.2 Boundary Detection

The next step in the building modeling algorithm is to mark boundary points of each planar roof patch. These boundary points will be later used in footprint generation and polygon production. Some previous work, *e.g.*, [62], finds boundary points by measuring certain characteristics of roof points. These methods, although efficient, cannot guarantee a watertight boundary, thus limit the subsequent processing. Other work uses 2D delaunay triangulation to find polygonal boundaries. *E.g.*, Verma *et al.* [60] introduce a plane ball-pivoting algorithm to triangulate roof points and detect boundaries. These algorithms are able to generate perfect boundaries, but sacrifice time efficiency. In addition, robust implementations for such algorithms are hard to achieve.

This section proposes an algorithm which combines the advantages of these two kinds of algorithm. The proposed method is inspired by part of my early work about a 3D contouring algorithm in [68].

Initially, for each plane roof patch, all the roof points are projected on the x-y plane and embedded into a uniform grid. The grid cells with roof points inside are marked as *object cells*, and illustrated as the grey cells in Figure 3.6. With this uniform grid setup, boundary points and boundary lines are generated respectively:

- **Boundary points:** For each grid edge l separating an object cell c_{in} and a background cell c_{out} (thick red edges in Figure 3.6), the nearest LiDAR point to l in c_{in} is marked as a boundary point $\mathbf{p}(l)$, shown as the red circles in Figure 3.6.

- **Boundary lines:** For each grid corner c (red squares in Figure 3.6) shared by two boundary grid edges l_1 and l_2, a boundary line $(\mathbf{p}(l_1), \mathbf{p}(l_2))$ is created, shown as the thin red lines in Figure 3.6.

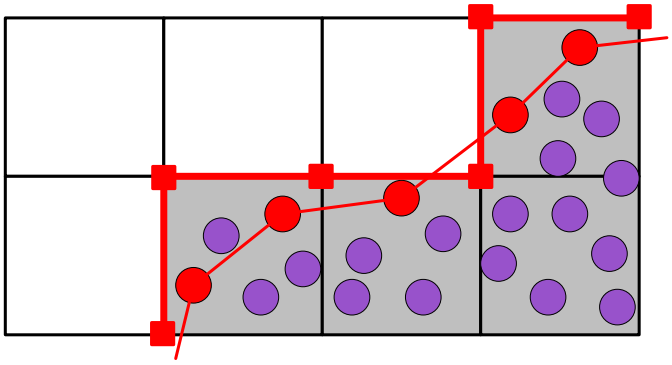


Figure 3.6: An illustration of the boundary detection algorithm. Circles represent the LiDAR points projected on the x-y plane. The boundary of this patch is composed of the boundary points (red circles) connected by boundary lines (thin red lines).

Intuitively, the result of this algorithm is the dual polygon of the object cell boundary along grid edges. Since the latter is a watertight polygon, its dual polygon is also watertight. This property is especially useful in the mesh creation.

The boundary extraction algorithm is efficient and robust. It produces a topology-correct boundary, but cannot guarantee the geometry completeness. For example, in Figure 3.6, there is one point outside the extracted boundary polygon. Since this situation happens rarely and has little effect on the subsequent processing, the resulting boundary is a good approximation.

Morphological operations: Another advantage of this boundary detection algorithm is that it supports morphological operations in an easy manner: after marking object and background cells, one can treat this grid as a monochrome image and apply any morphological operation [54] onto it. Figure 3.7 illustrates an example from part of an industrial site. Directly extracting boundaries from the object patches produces numerous artifacts, shown in (b). Hence, an *opening* morphological operation is applied to remove the insignificant features, shown in (c).

Figure 3.7: Boundary extraction for part of an industrial site: (b) without any morphological operation; (c) with an *opening* morphological operation, most artifacts are removed.

Configuration: The only parameter in the boundary extraction algorithm is the unit length of the uniform grid. This parameter controls the connectivity of the patch. Similar to the ball radius parameter in ball-pivoting algorithm used by [60], a large cell length creates a coarse boundary, while a small cell length keeps details along the boundary as well as noise. Empirically, a cell length that equals to the neighborhood size δ is adopted.

3.4.3 Building Model Creation

So far the building point cloud has been split into several planar roof patches with their boundaries marked out. A Naïve modeling approach takes each boundary as a roof polygon, and connects it to the ground by adding vertical walls. Solid building blocks are created in this way, which are combined together to form 3D building models. A min-max polygon simplification algorithm [32] is introduced to simplify the boundaries and reduce the complexity of each building block. In practice, due to the residual sensor noise, boundaries of neighboring planar roof patches are not strictly aligned together,

thus the Naïve approach may generate overlaps and cracks between neighboring building blocks.

My study observes the fact that most boundary line segments in a local area fall into a couple of directions, known as *principal directions*. Once detected, they are used to align boundaries before polygon simplification. In addition, if two segments from different roof patches are aligned to the same direction and the distance between their projections on x-y plane is small, they are very likely to be the common boundary segment shared by the two roof patches (when their heights are close), or be the two ends of a vertical wall (when there is a height gap). Based on these observations, a four-steps algorithm is designed to improve the modeling quality.

Principal direction detection

Intuitively, the principal directions are the boundary edge directions which most commonly appear in the local area. Thus, the detection algorithm estimates the tangent direction at each boundary point using a 2D covariance analysis, and then employs a histogram to statistically analyze these directions. Principal directions are detected as the peaks of this direction histogram.

For robustness, a Gaussian filter is applied to smooth the histogram before detecting peaks. Specifically, the histogram is modified by convolution with a Gaussian function

$$g(x) = \frac{1}{\sqrt{2\pi} \cdot \sigma} \cdot e^{-\frac{x^2}{2\sigma^2}}. \tag{3.10}$$

The local maxima with sufficient samples of this smoothed histogram are detected as peaks, *i.e.*, principal directions.

Figure 3.8(b) shows the direction histogram of a Denver city piece shown in Figure 3.8(a). Four principal directions are detected: $0°, 43°, 90°, 133°$, which can be separated into two orthogonal pairs, illustrated as the colored arrows in Figure 3.8(a). This

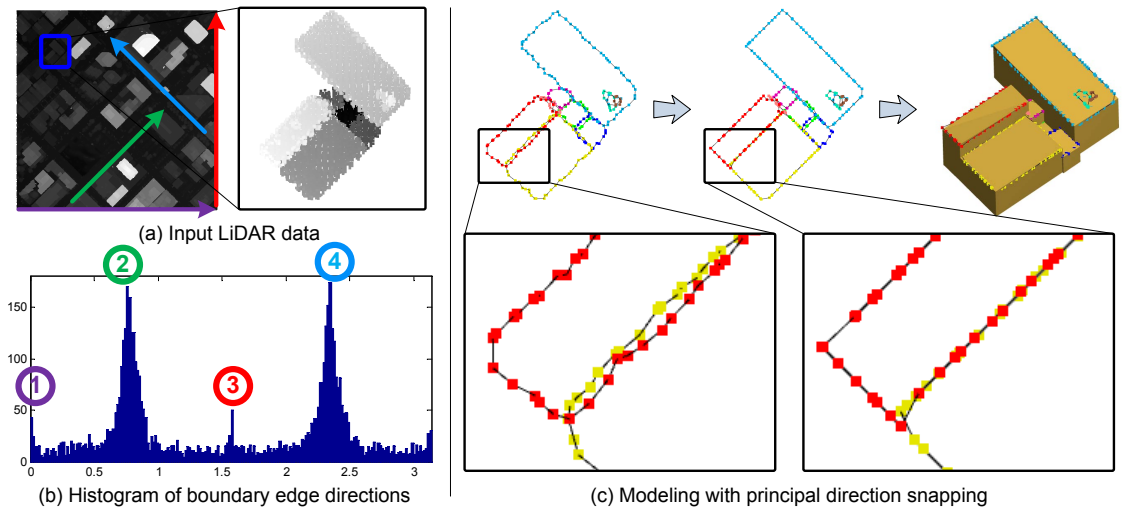


Figure 3.8: Building modeling using principal direction snapping: (a) input LiDAR point cloud chopped from Denver city; (b) histogram of boundary line directions, with four principal directions detected and marked in the figure; (c) boundaries of planar roof patches are snapped onto the principal directions and simple building models are created.

phenomenon concurs with the observation made by the previous work that orthogonal corners are a common pattern in building footprints. In addition, 0° and 90° directions contain less samples than the other two. They represents the chopping boundary of the area.

Principal direction snapping

During this procedure, the boundaries of planar roof patches are snapped to the principal directions as much as possible without exceeding a small error tolerance ϵ. A greedy algorithm is applied over each patch boundary loop $\mathcal{B}$, detailed as follows.

Every boundary point and principal direction pair $(\mathbf{b}_i, \mathbf{d}_j)$ defines a line in 2D space: $L(\mathbf{b}_i, \mathbf{d}_j) = \{\mathbf{p} | (\mathbf{p} - \mathbf{b}_i) \times \mathbf{d}_j = 0\}$. For each such pair, starting from the seed point $\mathbf{b}_i$, the longest continuous point sequence is detected along $\mathcal{B}$, in which all the points

have a distance to L smaller than an error tolerance ϵ. *I.e.*, in this continuous boundary segment $S(\mathbf{b}_i, \mathbf{d}_j)$, all points satisfy:

$$distance(\mathbf{p}, L(\mathbf{b}_i, \mathbf{d}_j)) = \frac{|(\mathbf{p} - \mathbf{b}_i) \times \mathbf{d}_j|}{|\mathbf{d}_j|} < \varepsilon. \tag{3.11}$$

The snapping algorithm then iteratively does the followings: (1) detect the point-direction pair $(\mathbf{b}_i^{max}, \mathbf{d}_j^{max})$ which maximizes the segment length $|S(\mathbf{b}_i, \mathbf{d}_j)|$; (2) project (snap) the points in $S(\mathbf{b}_i^{max}, \mathbf{d}_j^{max})$ onto line $L(\mathbf{b}_i^{max}, \mathbf{d}_j^{max})$; and (3) remove intermediate points in $S(\mathbf{b}_i^{max}, \mathbf{d}_j^{max})$ from $\mathcal{B}$. The last step prevents overlaps between different segments. An exception is the end points of segments, which can belong to two boundary segments. The iterative process stops when the size of the longest boundary segment falls under a certain value (empirically, 10 points).

Neighbor segments snapping

After aligning segments with principal directions, the modeling results can be further improved by snapping neighbor segments to form vertical walls and avoid cracks. Two boundary segments $L(\mathbf{b_1}, \mathbf{d})$ and $L(\mathbf{b_2}, \mathbf{d})$ are neighbors when they have the same direction $\mathbf{d}$ and their distance is smaller than a given threshold, *i.e.*,

$$distance(L(\mathbf{b}_1, \mathbf{d}), L(\mathbf{b}_2, \mathbf{d})) = \frac{|(\mathbf{b}_1 - \mathbf{b}_2) \times \mathbf{d}|}{|\mathbf{d}|} < \epsilon_l. \tag{3.12}$$

For each such neighbor boundary segment pairs.

- If they are from the *same* planar roof patch, and point toward the same direction, these two segments are snapped onto the same line. This operation handles the case that a line segment is broken into several segments due to noise or occlusion.

- If they are from *different* planar roof patches, and point toward opposite directions (*i.e.*, are of the opposite orientation), the snapping algorithm first checks if they

overlap when snapped to the same line. If so, they must be the common boundary segment shared by the two roof patches (when their heights are close), or be the two ends of a vertical wall (when there is a height gap). Thus, they are snapped together to create corresponding features.

Polygonal mesh creation

After boundary edge segments are detected and snapped to the principal directions, a polygon is created for each planar roof patch. Along the boundary loop, if two consequent segments share a common boundary point, the polygon simply connects them together. Otherwise, the min-max polygonal simplification algorithm [32] is adopted to simplify the intermediate point sequence between the two segments.

As mentioned previously, when all these polygons are created, vertical walls are generated to connect two snapped polygons, or a roof polygon and ground. The final model is the composition of all the building blocks.

Figure 3.8 demonstrates a whole pipeline to create a building model. Starting from a building roof patch, plane patches are detected and boundaries are extracted. Then these boundaries are snapped to the principal directions which are detected in Figure 3.8(b). In addition, neighboring segments are aligned together to form vertical walls. The final result shown in Figure 3.8(c) contains 72 triangles, which is simplified from a building patch with 9,778 roof points.

3.4.4 Non-Planar Roof Extension

One limitation of this automatic building modeling algorithm is that it only supports flat roofs. However, in some cases, there are non-planar objects in the data set which need special treatments. For example, Figure 3.9(a) shows an industrial site in which the most interesting objects are the oil tins and tanks represented in cylinder and cone shapes.

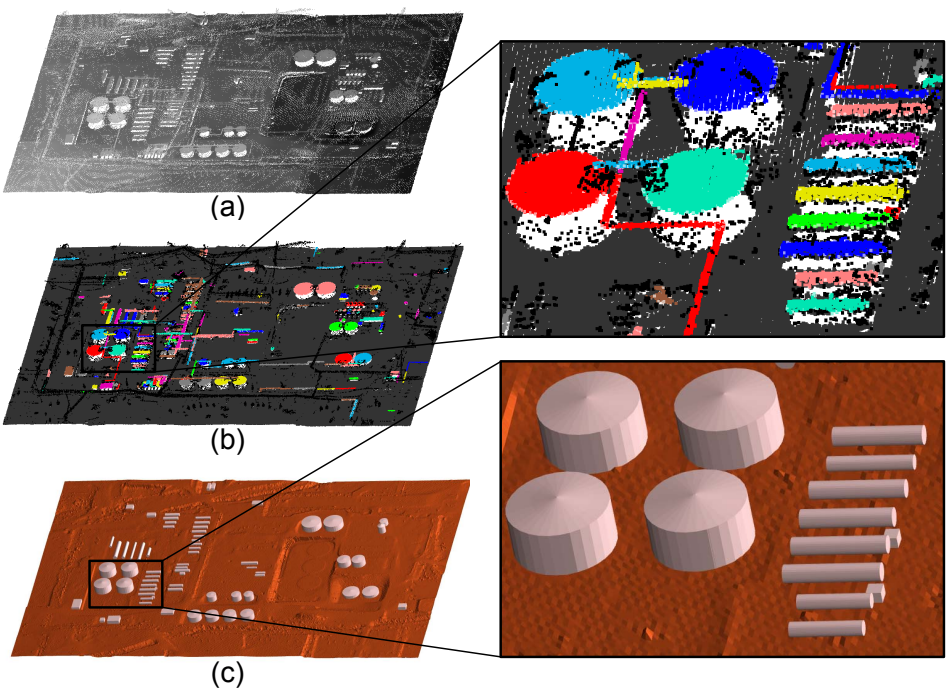


Figure 3.9: Reconstruction of an industrial site using non-planar roof extension: (a) input LiDAR data; (b) detected object patches are rendered with bright colors, while ground and noise are rendered with dark-grey and black; (c) the reconstructed models, with different types of objects generated in different ways.

These shapes increase the complexity of the object reconstruction problem. However, fortunately, these non-planar shapes usually fall into a couple of specific patterns. Therefore, once the pattern types are known, typical pattern recognition algorithms such as RANSAC [14] can be applied to estimate parameters and create 3D models.

In particular, user interaction is introduced to identify the pattern types. The initial LiDAR data is first segmented into different patches and the noise/tree/ground points are removed from the data set. Then the object patches are exhibited to the user as Figure 3.9(b). The user can move his mouse onto any object patch, click to select this patch, and press a key to specify its pattern type, *e.g.*, cone, cylinder, or planar-roof object. The modeling program automatically extracts boundary loops for each object, as presented in the previous sections; and a RANSAC algorithm [14] based on the identified pattern

type is applied to estimate the parameters of the model. The reconstruction result is shown in Figure 3.9(c). Objects of different patterns are created successfully.

3.5 Terrain Modeling

The last part of the general urban modeling pipeline is terrain modeling, which is achieved by rasterizing the detected terrain point patch into a DEM (Digital Elevation Model). Specifically, all the terrain points are embedded into a uniform grid spanning over the entire urban area. One mesh vertex is created at the center of each grid cell with its height being the average height of all terrain points in this grid cell. Due to the occlusion by overground objects such as buildings, trees, and vehicles, the uniform grid may have empty grid cells at the occluded areas. Thus, the terrain modeling module fills holes by solving a Laplace's equation $\nabla^2 z = 0$ taking the heights in all empty grid cells as unknowns and the heights in all non-empty cells as the boundary condition. In particular, for each empty cell at (i, j),

$$4z_{i,j} = z_{i-1,j} + z_{i+1,j} + z_{i,j-1} + z_{i,j+1}. \tag{3.13}$$

The unknown heights are calculated by solving this equation array, and the ground mesh is a DEM connecting all these vertices.

3.6 Experimental Results on Small Sample Data Sets

The general urban modeling system is tested on three different data sets: Denver, Oakland, and an industrial site. The resolution of these data sets varies from 6 samples/m^2 to 17 samples/m^2 In order to process all the data in-core, a 1km-by-1km piece is chopped from Denver as shown in Figure 3.10(a) and a 600m-by-600m piece is chopped from

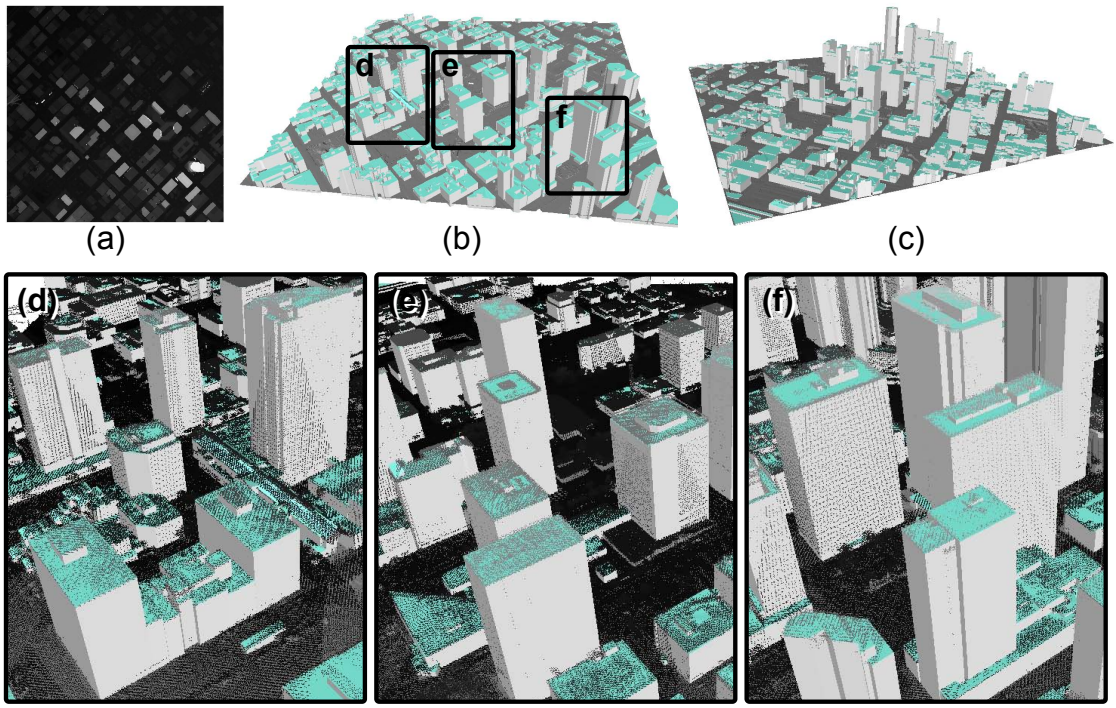


Figure 3.10: Automatic urban modeling of a $1km \times 1km$ piece from the city of Denver: (a) input LiDAR data; (b,c) modeling result viewed from different perspectives; (d,e,f) closeups of the modeling result with initial point cloud overlayed

Oakland as shown in Figure 3.11(a). All the experiments are made on a consumer-level laptop (Intel Core2 1.83GHz CPU, 2G memory) with an external hard disk. The time cost of the entire system is proportional to the number of input points, with a ratio around 8 minutes/million points.

Figure 3.10 shows the reconstruction results of an urban area in the city of Denver. The resulting building models are composed of simple and clean triangular meshes, thus are ideal to some applications such as virtual city tourism. In addition, building blocks are aligned tightly together and form building models of good shapes. As shown in Figure 3.10(d,e,f), these building models fit the raw LiDAR point cloud in an excellent manner.

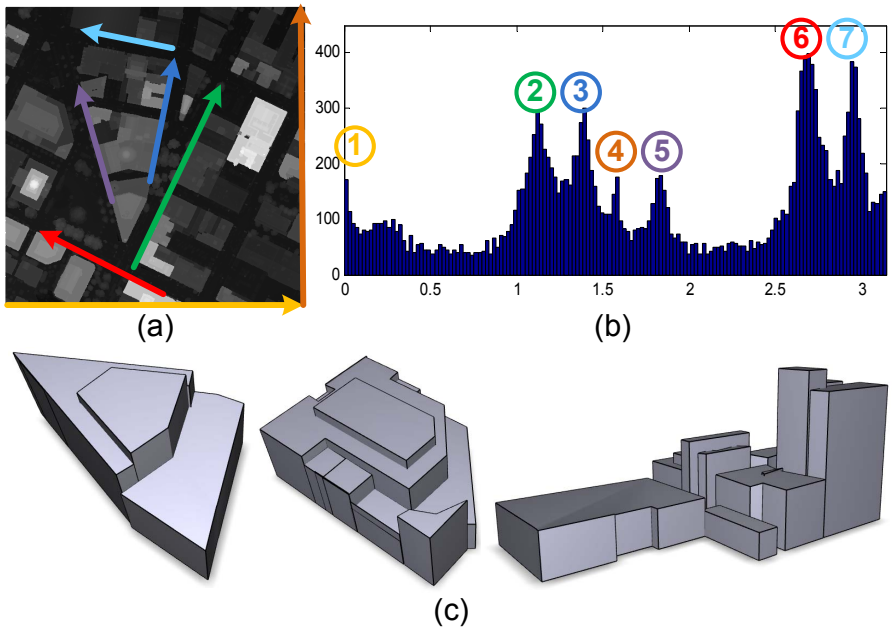


Figure 3.11: Urban modeling of an area in the city of Oakland: (a) input LiDAR point cloud, with intensity representing point height; (b) a histogram with seven principal directions detected and illustrated as colored arrows in (a); (c) reconstructed building models; both orthogonal corners and non-orthogonal corners are reconstructed correctly.

To further demonstrate the ability of supporting multiple principal directions, the system is tested on a challenging area from the city of Oakland, as shown in Figure 3.11. Seven principal directions are automatically detected from the original data sets, which are illustrated as the arrows in Figure 3.11(a). Since the general modeling system has no pre-assumptions on the corner angles, correct models are generated with corners of angles between any two principal directions, which may be $90°$, $45°$, or any other angle; shown in Figure 3.11(c).

The final testing data set is an industrial site shown in Figure 3.9(a). This data set is used to demonstrate the non-planar roof extension to the urban modeling system. Besides plane-shaped roofs, two new types of geometry patterns are involved, namely, a standing cylinder with a cone on top of it (an oil tin) and a lying cylinder (a tank).

Objects of all three patterns are successfully detected and reconstructed by the proposed approach, as shown in Figure 3.9(c).

Chapter 4

Streaming Urban Modeling for Large-Scale Data Sets

Like many previous work, the automatic urban modeling pipeline proposed in Chapter 3 is an in-core method. In other words, it needs to load all the LiDAR data into the memory before processing. Thus, there is a conflict between the increasing size of data sets and the limitation of computer hardware.

A common way to alleviate this problem is to partition the whole data set into small tiles and process them one at a time (*e.g.* [41, 48]). By merging building models generated from different tiles, this approach can produce 3D models from large LiDAR data sets. However, it may introduce artifacts alongside the boundaries between tiles. Although these boundary effects can be moderated by introducing extra processing on boundary regions, the additional processing for tile partitioning, boundary handling and modeling merging is inefficient, tedious and may introduce ambiguity (*e.g.* large buildings that span the intersections of multiple tiles).

This chapter presents a streaming framework to handle extremely large data sets in a *seamless* manner, meaning that the proposed method needs no special treatment for tile-boundaries. By storing data as stream files on hard disks and using main memory as only a temporary storage for ongoing computation, the method achieves efficient out-of-core data management. This gives the urban modeling system the ability to handle data sets with hundreds of millions of points in a uniform manner. For example, by adapting the automatic urban modeling pipeline into this streaming framework, the whole urban

model of Atlanta is reconstructed from 17.7GB LiDAR data with 683M points by under 25 hours of unattended processing using less than 1GB memory. As a comparison, an in-core program would need more than 100GB memory to process this data in one pass.

Section 4.1 briefly reviews the state-of-the-art streaming approaches. Section 4.2 presents a streaming framework. In Section 4.3 the general urban modeling system proposed in Chapter 3 is adapted into the streaming framework. Experiments on different city-scale data sets are given in Section 4.4.

4.1 Review of Streaming Approaches

To solve the conflict between extremely large data sets and computer hardware limitation, *streaming methods* are developed in geometry modeling and computer graphics areas. They have succeeded in a board variety of applications, such as mesh processing [26] and compression [25], tetrahedral mesh simplification [61], level sets methods [45], point cloud processing [46], LiDAR data rasterization [28], dynamic processing [35], Poisson equation solver [53], and delaunay triangulation [27].

Here I highlight [27], [28] and [46]. Isenburg *et al.* [27, 28] reveal the local *spatial coherence* of aerial LiDAR data and propose a grid-based indexing structure and a *spatial finalization* mechanism, which is the basis of the approach proposed in this chapter. Pajarola [46] performs a sequence of operations on a data stream. To allow data blocks to be in different states and deal with transitions between states, he arranges data points into a FIFO queue by sorting the data along one dimension of the largest extent, which is less efficient and general, compared with the state propagation mechanism for solving the same problem, as stated in Section 4.2.

4.2 Streaming Framework

Generally, the streaming framework is an out-of-core architecture. A streaming program acts as a frontier through the data stream - it reads from an input stream, loads necessary data in-core; processes data; and once the data is no longer needed for further processing, it is written into the output stream. To adapt the general modeling pipeline into this out-of-core architecture is particularly difficult, because each urban modeling module in the general pipeline involves several intermediate steps. Each step usually relies on one point's neighboring points being processed by some preceding steps. *E.g.*, in classification, normal covariance analysis over a point p requires position covariance analysis results on all p's neighboring points. Thus, these intermediate steps cannot be batch-processed, and execution order of these steps must be carefully determined.

My research achieves this goal by defining *streaming operators* and *steaming states* formally in Section 4.2.2, and introducing a *state propagation* algorithm to determine the execution order in Section 4.2.3. Section 4.2.1 gives the definition of *point stream* and reviews the *finalization* mechanism as first proposed by Isenburg *et al.* [27].

4.2.1 Point Streams and Finalizer

As observed by Isenburg *et al.* [27] and Pajarola [46], *spatial coherence*, which either appears in the original data set or is the result of a resorting algorithm, can greatly improve the memory efficiency in an out-of-core algorithm. To exploit such spatial coherence, *point stream* is defined as the basic form of data that is processed in a streaming framework:

Definition 4.1 A *point stream* is a FIFO queue composed of point records and *finalization tags*.

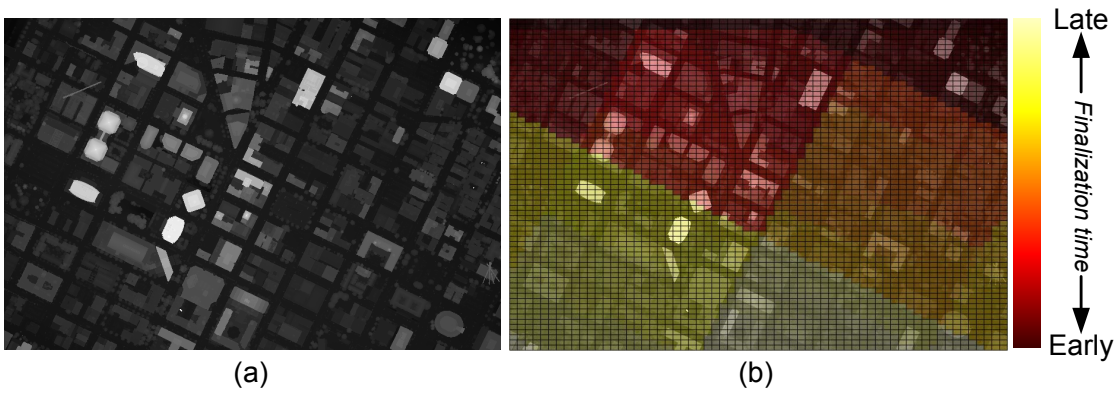


Figure 4.1: Finalization result of the Oakland data set with cell colors in (b) representing the finalization time. Spatial coherence is revealed by the regularity of color distribution.

A point stream is generated by inserting *finalization tags* into original input data, which is done by the pre-processing module called *Finalizer*. A finalization tag $f_{\mathcal{A}}$ is a symbol to indicate that all the point records in a spatial area $\mathcal{A}$ has appeared in the point stream before it. This is necessary because the original data usually is not spatially ordered strictly. When a streaming program meets $f_{\mathcal{A}}$, it gets the guarantee that the information within $\mathcal{A}$ is available and further actions can be taken.

In particular, the input data is partitioned into $2^k \times 2^k$ uniform rectangle grid, so that each cell is the basic unit of spatial area in streaming processing. Therefore, the task of the *Finalizer* is to insert one finalization tag for each such cell into the input data.

Note that the finalization tags only provide a mechanism to reveal the spatial coherence but not to generate it. For example, in the worst case, the last point of each grid cell appears at the very end of the input data; the finalization tags will then all be inserted at the end of the stream and no memory efficiency can be produced. Fortunately, the point sequence in the aerial LiDAR data shows remarkable spatial coherence to make significant memory efficiency [27]. Also, the spatial coherence can be further enhanced by a *chunking* algorithm which partially resort the point records [27].

Figure 4.1 shows the finalization result of the Oakland data set. The colors illustrate the time when a grid cell is finalized in the point stream. The spatial coherence between grid cells is revealed by the regularity of the color distribution.

4.2.2 Streaming Operators and States

The basis of the point stream processing framework is the following observation: most of the complicated local algorithms can be decomposed into a series of *streaming operators*, which is a generalized form of the "stream operators" defined in [46].

Definition 4.2 A *streaming operator* $\Phi_k(\mathbf{p}_i)$ is a local operator which requires all the points in $\mathbf{p}_i$'s local neighborhood $N_k(\mathbf{p}_i)$ to be at *streaming state* s_{k-1} or higher state; $\Phi_k(\mathbf{p}_i)$ takes these points as input and transit point $\mathbf{p}_i$ to *streaming state* s_k.

Here a series of *streaming states* $s_0, s_1, \ldots, s_m$ is defined as follows: the first state s_0 is always *"Unread"* and the last state s_m is always *"Written and released"*. Except for the last state, the information in a point record at a lower state is always a subset of the information at a higher state. In a complete streaming process, each point sequentially experiences states from s_0 to s_m. A state transition from s_{k-1} to s_k can only be invoked by a streaming operator Φ_k.

The streaming classification module is demonstrated here as an example. All the points are in the initial state s_0 =*"Unread"*. The first streaming operator $\Phi_1(\mathbf{p}_i)$ reads $\mathbf{p}_i$ from the point stream into memory and changes its state from s_0 to s_1 =*"Read"*. The second operator $\Phi_2(\mathbf{p}_i)$ collects the positions of points in $\mathbf{p}_i$'s local neighborhood $N_2(\mathbf{p}_i)$, uses these information to estimate $\mathbf{p}_i$'s normal, then transits $\mathbf{p}_i$ into state s_2 =*"With normals"*. During this process, all the points in $N_2(\mathbf{p}_i)$ must be at least at streaming state s_1, *i.e.*, read from the stream and loaded in-core. In a similar manner, the following operators are executed sequentially until point $\mathbf{p}_i$ finally gets into state

s_{m-1}. The last stream operator $\Phi_m(\mathbf{p}_i)$ then writes it to the output stream, releases it from memory and turns its state into s_m =*"Written and released"*. In this whole process, point records which are in the first and last states are stored in disk files, and only a small fraction of points in intermediate states need to be loaded in-core. These intermediate states are called *active states*.

To determine when an operator Φ_k can be invoked, *scope radius* is defined:

Definition 4.3 The *scope radius* $R(\Phi_k)$ is the radius of point $\mathbf{p}_i$'s neighborhood $N_k(\mathbf{p}_i)$ required by $\Phi_k(\mathbf{p}_i)$.

$R(\Phi_k)$ reflects the size of the area affecting Φ_k. For convenience, let $R(\Phi_k) = 0$ when the streaming operator Φ_k does not need information from the local neighborhood, *e.g.*, a *"read from stream"* operator. In other cases, it is determined by the corresponding streaming operator. *E.g.*, the *"normal estimation"* operator of the classification module requires a scope radius equal to the neighborhood size δ defined in Section 3.2. The only exception is the scope radius of the last operator Φ_m, which is forced to be the largest scope radius of all the other streaming operators, *i.e.*,

$$R(\Phi_m) = \max\{R(\Phi_1), R(\Phi_2), \ldots, R(\Phi_{m-1})\}, \tag{4.1}$$

because Φ_m is the only information-subtracting operator – once performed, the point record is no longer available in the memory. By forcing $R(\Phi_m)$ to be the largest scope radius, $\Phi_m(\mathbf{p}_i)$ is applied only when all the points in $N_m(\mathbf{p}_i)$ are at least at state s_{m-1} (written or waiting to be written), so that no point still requires information from $\mathbf{p}_i$ to complete a streaming operator $\Phi_k, k < m$.

The scope radius is also helpful in determining the size of the spatial unit (cell). The side length of a grid cell must be no less than $R(\Phi_m)$, so that the impact of any

streaming operator applied on a cell c is restricted within its 1-ring neighborhood. This is particularly convenient for the state propagation algorithm in the following section.

4.2.3 State Propagation

State propagation is an algorithm which performs the streaming operators in the correct order.

State propagation uses cell as the basic unit to perform an operator, and operator Φ_k performed on cell $c_{i,j}$ is denoted by $\Phi_k(c_{i,j})$. As discussed before, to determine whether $\Phi_k(c_{i,j})$ can be invoked, one only needs to check if all points in $c_{i,j}$'s 1-ring neighborhood are at least in state s_{k-1}. In addition, if a cell reaches a new state s_{k-1} by operator Φ_{k-1}, only its 1-ring neighbor cells may receive the direct impact from this transition, *e.g.*, a neighbor cell may now satisfy the state prerequisite for Φ_k. Based on this observation, the key idea of the state propagation algorithm is to notify all the 1-ring neighbors whenever a cell's state is changed by completing a new operator.

The algorithm starts by reading a cell c_{next} from the input point stream S_{in}. A recursive function *cellAction()* is then called to perform steaming operators in an orderly manner. Taking a cell c and a streaming operator Φ_k as input, *cellAction()* first checks if the state prerequisite for operator $\Phi_k(c)$ is satisfied. If not, it aborts the operation; otherwise, it performs operator $\Phi_k(c)$, transit cell c to state s_k and notifies each cell c^* in c's 1-ring neighborhood by recursively calling *cellAction()* for c^* and operator Φ_{k+1}. In this way, the state change of the initial cell c_{next} is propagated in the grid and operators will be performed once they are ready. The pseudo-code of the algorithm is shown in Table 4.1.

Figure 4.2 shows an example of a state propagation procedure in a 4-states problem. The state of each cell is denoted by the number and the color of the cell. In the first step, a cell $c_{1,1}$ (marked by the black frame) is read from the point stream, and its

/********** **Main program** **********/
Input: a point stream S_{in}; a set of streaming states $\{s_0, \ldots, s_m\}$; and a set of streaming operators $\{\Phi_1, \ldots, \Phi_m\}$.
Output: a point stream S_{out}.
While S_{in} is not empty **do**:

- Read the next cell c_{next} from S_{in};
- Call function ***cellAction***(c_{next}, Φ_1).

End of main program.

/********** **cellAction() function** **********/
Input: a grid cell c at state s_{k-1}; and a streaming operator Φ_k.
For each cell c^* in the 1-ring neighborhood of c **do**:

- **if** the state of c^* is lower than s_{k-1}, **then** return *"not ready"*.

/* If not returned, all cells in the 1-ring neighborhood pass the state test. */
Execute $\Phi_k(c)$. /* take action and change the state of c to s_k */
For each cell c^* in the 1-ring neighborhood of c **do**:

- **if** the state of c^* is s_k, **then** call function ***cellAction***(c^*, Φ_{k+1}).

End of cellAction() function.

Table 4.1: State propagation algorithm

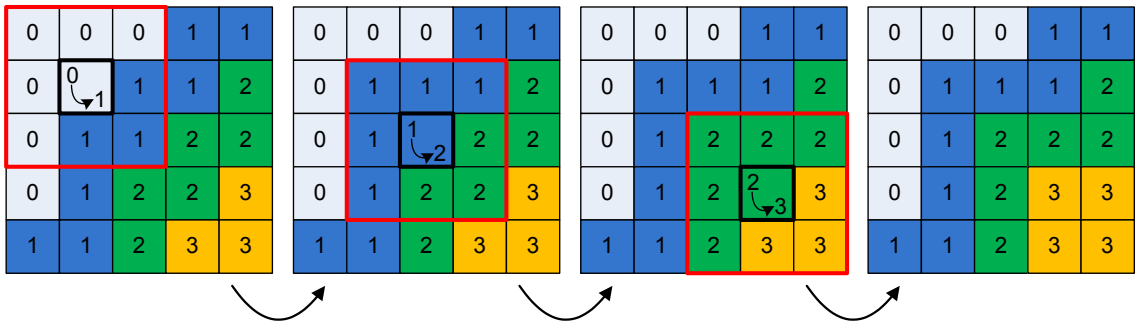


Figure 4.2: An example of the state propagation algorithm. The numbers and colors denote states. When the state of a cell is changed (marked with black frame), it notifies its 1-ring neighborhood (red frame) to check if any of them is ready for the next operator. The whole process is a recursive procedure.

state is transited to s_1 via a call to *cell-Action()* function, which then leads a number of streaming operators performed on other cells and state updates. Cells that reach the final state will be written to the output file. The propagation terminates when no more streaming operator is possible (*i.e.*, none of them fulfill their prerequisites), and the state propagation algorithm will read a new cell into the active set and repeat the propagation until all input data is fully processed.

4.3 Streaming Urban Modeling

This section shows how to adapt the general urban modeling system into this streaming framework. Section 4.3.1 presents the modified pipeline. The subsequent sections then detail modules including streaming classification, streaming segmentation, building modeling, and terrain modeling.

4.3.1 Streaming Modeling Pipeline

An overview of the streaming pipeline is demonstrated in Figure 4.3. The input LiDAR data (usually stored in a list of disk files) is sequentially read by a pre-processing module called *Finalizer*, which inserts finalization tags into the data, and produces a point stream as defined previously.

Taking this point stream as input, two specific streaming modules are performed sequentially: the *Classifier* which classifies vegetation points from building and ground points, and the *Splitter* which segments single building patches from the building and ground points. Both are implemented following the formulation of streaming operators and states in the last section. In the *Classifier* and *Splitter* blocks of Figure 4.3, the solid colored cells denote active (*i.e.*, in-core) data set with different colors corresponding to different streaming states; the dark red region denotes processed and released data; and

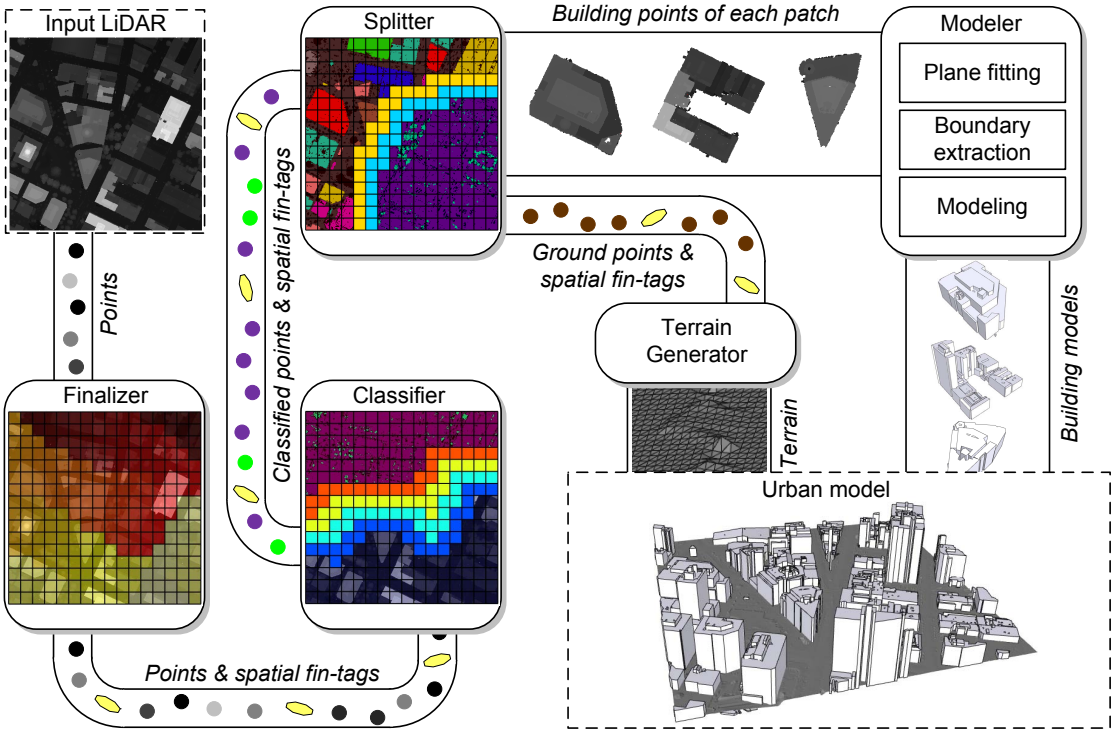


Figure 4.3: An illustration of the streaming building reconstruction pipeline. A pre-processing module (which is called *Finalizer* in [27]) inserts finalization tags (yellow ovals) and generates a point stream which flows over the *Classifier* and the *Splitter* sequentially. Both components introduce a state-propagation mechanism so that only data with active states (solid colored cells in *Classifier* and *Splitter*) are loaded in-core. The *Splitter* finally generates a point stream and a building stream; which are converted into a terrain model and various building models using the *Terrain Generator* and the *Modeler* respectively.

the dark blue denotes the input waiting to be read. The active set progresses as a frontier through the input stream until all data is processed.

The *Splitter* outputs two streams: a point stream with geometry information of all ground points and a building stream which consists of individual building patches. The point stream is converted to a terrain model using a *Terrain Generator*; and a *Modeler* is responsible for turning each building patch into a polygonal building model. The whole urban model is finally created by combining them together.

Streaming operators	Streaming states
Φ_1: Read data from input point stream and allocate memory.	s_0: Unread
Φ_2: Apply covariance analysis on positions to estimate normals; and compute $\mathcal{F}_{1,2,3}$.	s_1: Read
Φ_3: Apply covariance analysis on normals; calculate $\mathcal{F}_{4,5}$; and apply SVM classifier.	s_2: With normals
Φ_4: Refine classification by making points in local neighborhood vote on result.	s_3: Classified
Φ_5: Write point records to output point stream, and release them from memory.	s_4: Refined
	s_5: Written and released

Table 4.2: Streaming operators and states for classification

4.3.2 Streaming Classification

The first step of the classification algorithm proposed in Section 3.2 is a Support Vector Machine training procedure [2, 8] based on local geometric features. This off-line step is processed once and for all, thus needs not to be adapted into the streaming framework.

With the training results, the classification algorithm introduces a linear classifier with five types of features based on differential geometry properties: regularity $\mathcal{F}_1$, horizontality $\mathcal{F}_2$, flatness $\mathcal{F}_3$, and two normal distribution measurements $\mathcal{F}_4$ and $\mathcal{F}_5$. To compute these features for a point $\mathbf{p}$, covariance analysis is performed twice on its local neighborhood. The trained linear classifier then computes the classification result of $\mathbf{p}$ from these features. Finally, the classification results on $\mathbf{p}$'s neighbor points will vote for the final label of $\mathbf{p}$ in a refinement step.

Since each part of this algorithm requires only the information within a local neighborhood of point $\mathbf{p}$, it is easily decomposed into a series of streaming operators and states shown in Table 4.2, which are placed into the streaming framework and form up the *Classifier* module.

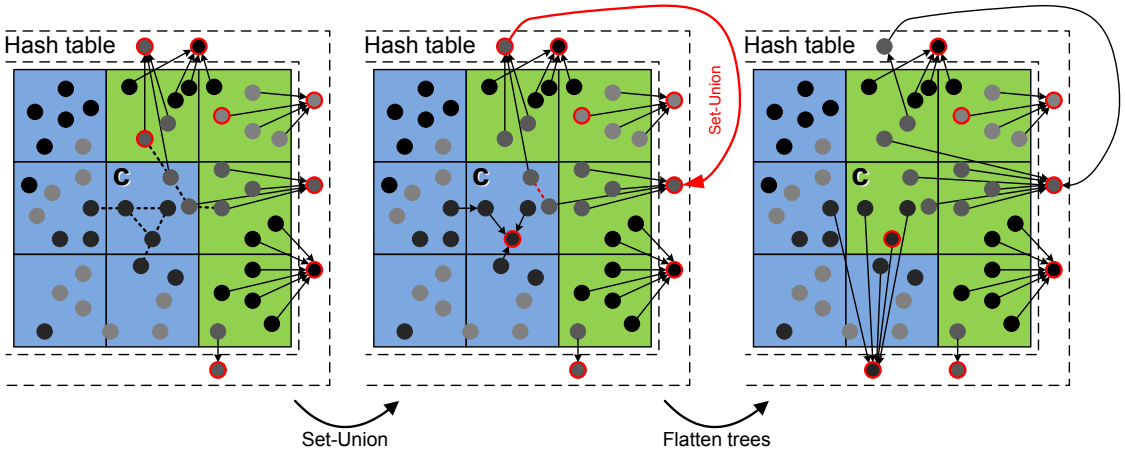


Figure 4.4: The streaming agglomerative clustering algorithm stores set-tree roots (marked with red frame) in a global hash table. It first performs a *union* operation on each neighbor point pair illustrated as the dotted line in the left figure. Then the algorithm flattens the set-trees shown in the middle figure and push new roots into the hash table (right figure).

4.3.3 Streaming Segmentation

The segmentation module aims to divide the input points into a ground patch and individual building patches. When adapting the general segmentation approach (Section 3.3) into the streaming framework, the main difficulty lies in the global characteristic of the segmentation problem. Specifically, patches (*e.g.*, the ground patch) may span over the entire area, and thus one cannot throw away all the information when data is written into the output stream. Instead, a small-but-sufficient global indexing structure is required, which is never released from memory to make the segmentation algorithm execute in a correct manner.

I choose to extend the agglomerative clustering algorithm proposed in Section 3.3.2 into a streaming segmentation approach, since it is a bottom-up algorithm and most of the structural information is stored as parent pointers attached to each point. The remaining structural information which needs to be processed globally is the temporary

roots of each trees, which are stored in a hash table as the global indexing structure. Here I call these roots "temporary" because they may be merged to the same tree in later processing. However, in order to avoid dangling pointers, once a point is pushed into the hash table, it is never released from the memory.

Even with this global indexing structure, there is still a danger of visiting dangling pointers. During streaming process, some points may be outputted to disk and released from the active set. Thus, if a point's root-seeking path contains some released points, invalid pointers will be visited when retrieving the cluster of this point. To solve this problem, *find* operation is introduced to flatten the root-seeking path as proposed in Section 3.3.2. The idea is straightforward: every time the agglomerative clustering algorithm in a cell c finishes, a *find* operator is applied on all points in c's 1-ring neighborhood, to flatten the root-seeking path of these points; after this, the newly created roots are pushed into the hash table. Thus, every point that has been touched in this process is guaranteed to have a parent pointer pointing to a root in the global hash table, which will not be released.

The pseudo-code of this algorithm is shown in Table 4.3. Figure 4.4 shows an example that illustrates the streaming agglomerative clustering process. The algorithm starts with a cell c whose 1-ring neighborhood are all available in memory. The pairs of points involved in c whose distances are small enough for a merging operation are connected in dashed lines. To process c, a *union* operation is performed on each such pair of points and their segments are merged (middle figure). The algorithm then performs a *find* operation over all the points touched in the first step to flatten all the trees which have been changed. Finally, the new roots generated during this process are added into the hash table.

Input: a cell c at state s_1, with its 1-ring neighborhood at state s_1 or higher; root hash table $\mathcal{H}$; and the distance threshold α.
/* Apply *union* operation over cell c */
For each point pair $(\mathbf{p}, \mathbf{q})$ where $\mathbf{p} \in c$ and $||\mathbf{p} - \mathbf{q}|| < \alpha$, **do**:

- Call function ***union(***$\mathbf{p}$***,*** $\mathbf{q}$***)***.

/* Flatten set trees from all touched points */
For each point $\mathbf{p}$ in the 1-ring neighborhood of c **do**:

- Flatten $\mathbf{p}$'s root-seeking path by calling ***find(***$\mathbf{p}$***)***.

/* Put new roots into the hash table */
For each point $\mathbf{p}$ in the 1-ring neighborhood of c **do**:

- **if** $\mathbf{p}$ is root and $\mathbf{p} \notin \mathcal{H}$, **then** push $\mathbf{p}$ into $\mathcal{H}$.

End of union-find algorithm.

Table 4.3: Streaming union-find algorithm (Φ_2)

Streaming operators	Streaming states
Φ_1: Read data from input point stream and allocate memory.	s_0: Unread
Φ_2: Apply streaming union-find algorithm described in Table 4.3.	s_1: Read
Φ_3: Write point records to output point stream, and release them from memory.	s_2: Segmented
	s_3: Written and released

Table 4.4: Streaming operators and states for segmentation

Except the global hash table, this segmentation method is completely local, thus it can be defined as streaming operators. A list of the streaming operators and corresponding states are given in Table 4.4.

As a result, the output point stream is decomposed into segments of points. The largest segment is taken as the ground patch and sent to the *Ground Generator* still in the form of a point stream. The rest segments are sent to the building modeling module.

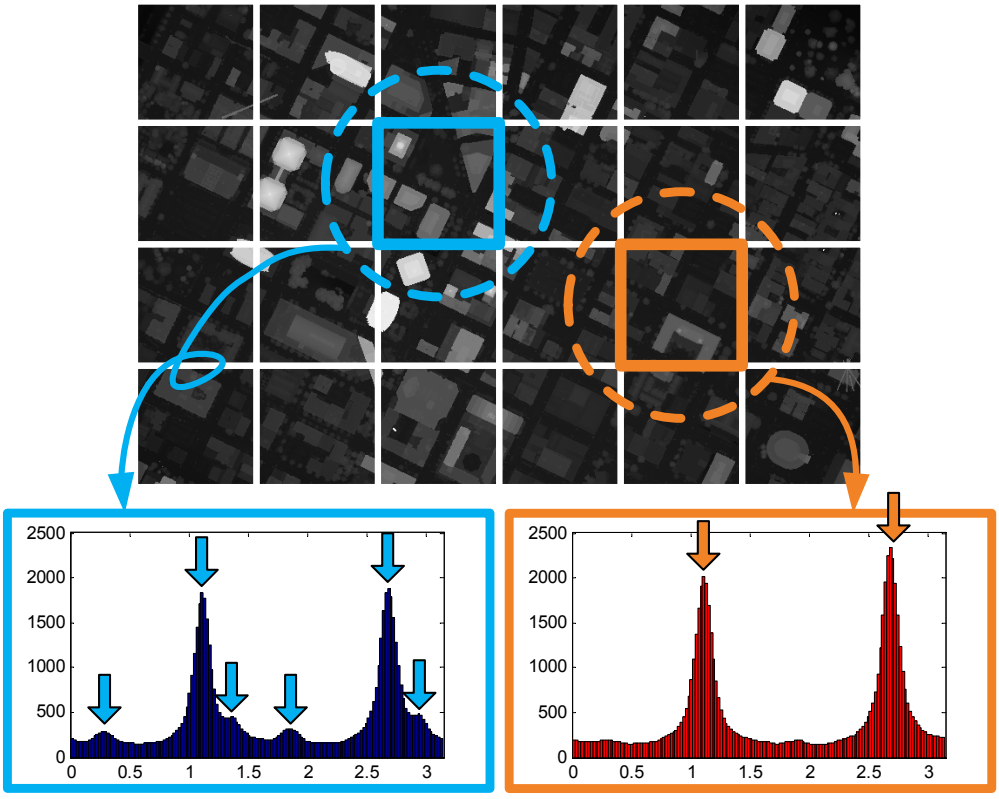


Figure 4.5: With the principal direction grid on the Oakland data set, six principal directions are detected for the blue cell (left), while two principal directions are detected for the orange cell (right).

4.3.4 Building Modeling

The building modeling algorithm now takes over these building patches. Since the number of points contained in a single building patch is small, the patches are loaded into the memory and processed one by one. In the experiments, the largest building patch is the large structure shown in Figure 4.6(a), containing 3.2M points, which takes 332MB of memory to process.

This section extends the automatic building modeling algorithm in Section 3.4 for building model reconstruction. Given a building patch and a set of *principal directions* as input, the algorithm automatically fits planes to the points and snaps the plane boundary segments onto the principal directions.

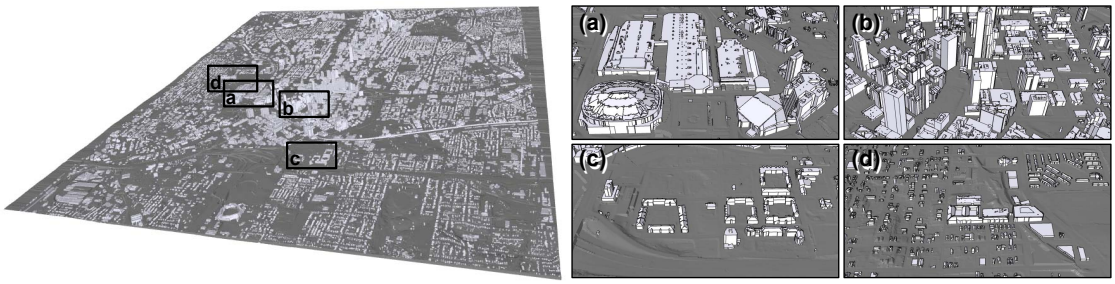


Figure 4.6: Reconstructed urban model of Atlanta city. Closeups of different areas are shown in the right sub-figures.

In Section 3.4.3, the principal directions are extracted over the entire input data, which is problematic in city-scale input. To allow the principal directions to reflect different boundary directions within local regions (such as the downtown area in the city of Atlanta shown in Figure 4.6(b) and the residential area shown in Figure 4.6(d)), a *principal direction grid* (Figure 4.5) is introduced. For each cell in this grid, a histogram of the tangent directions of all boundary points is computed within a local neighborhood and the peaks after Gaussian filtering are found to be principal directions.

4.3.5 Terrain Modeling

The objective of the terrain modeling algorithm is to rasterise the ground point stream into a digital elevation model. Taking the point stream as input, the algorithm builds up a square grid (whose user-selected unit length determines the precision of the terrain mesh); and counts the lowest ground point in each grid cell. These points are later accepted as vertices of the rasterised terrain model. The empty cells can be filled by solving a Laplace's equation as proposed in Section 3.5. However, for performance reason, a linear interpolation is applied to the gaps along each column on the grid. Trivial differences exist between results generated by these two methods; however, the improvement on efficiency is remarkable for city-scale data sets.

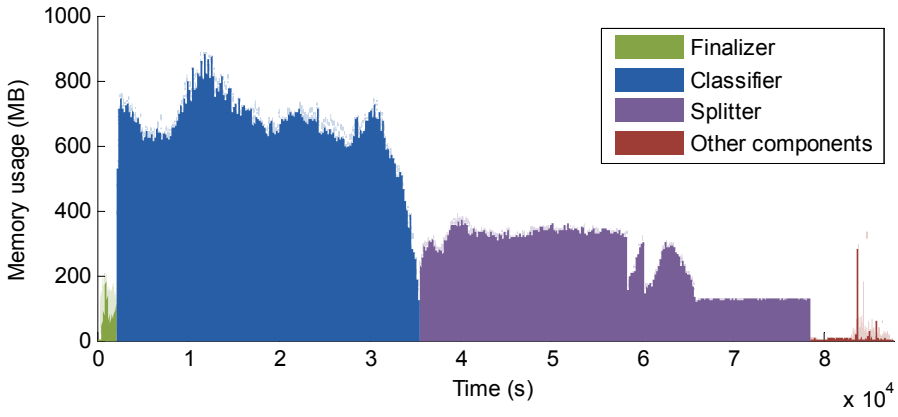


Figure 4.7: Memory usage during experiment on Atlanta data set

4.4 Experimental Results on City-Scale Data Sets

The streaming urban modeling system is tested on three different data sets, namely, Oakland, Denver, and Atlanta. The problem scale varies from 16M points to 683M points. My program generates polygonal urban models for each of them. All the experiments are done on a consumer-level desktop PC (Intel Core2 2.4GHz CPU, with 2G memory and 100GB free hard disk space). The running time and maximum memory usage are reported in Table 4.5. Although the average performance is affected by the characteristics of data sets, the average processing speed is faster than 3 minutes per million points.

Benefiting from the streaming framework, the memory footprint during the experiments is kept at a low level. The whole pipeline consumes no more than 1GB memory at any time to process the largest data sets (Atlanta) in one pass. This memory-saving mechanism can be explained by Figure 4.10, showing the *Classifier* module for Oakland data set. First, the finalization result in Figure 4.10(b) reveals the spatial coherence between streaming grid cells. Second, three snapshots are taken during the streaming

Model		Oakland	Denver	Atlanta
Input LiDAR data	Urban area	1.2km-by-0.8km	4km-by-3km	5.5km-by-7.1km
	Point number	16M	73M	683M
	File size	437MB	1.90GB	17.7GB
	Grid resolution	$64 \times 64 \times 64$	$512 \times 512 \times 512$	$512 \times 512 \times 512$
Time (hh:mm:ss)	Finalizer	24	4:43	34:58
	Classifier	11:10	53:31	9:18:09
	Splitter	3:14	2:03:56	11:54:29
	Others	3:09	15:23	2:33:24
	Total	17:57	3:17:33	24:21:00
Maximum memory usage (MB)	Finalizer	108	16	209
	Classifier	276	156	888
	Splitter	103	72	390
	Others	23	71	332
	Maximum	276	156	888
Output triangles	Building	62K	182K	1.12M
	Terrain	1.92M	10.7M	8.78M

Table 4.5: Three data sets with different sample rates are tested using the streaming building reconstruction system on a consumer-level PC. This table reports the running time and maximum memory usage in each pipeline module, namely, Finalizer, Classifier, Splitter, and other components. The experimental results show the ability of the streaming system to handle extremely large data sets in an efficient manner.

classification processing (Figure 4.10(c,d,e)), showing that only cells at active states (bright solid colored cells) are stored in memory. The remaining cells are either waiting in the input stream or have been written to the output stream and released from memory. With the spatial coherence guaranteed, the active cells are always a small fraction of the data; thus only small amount of memory is required.

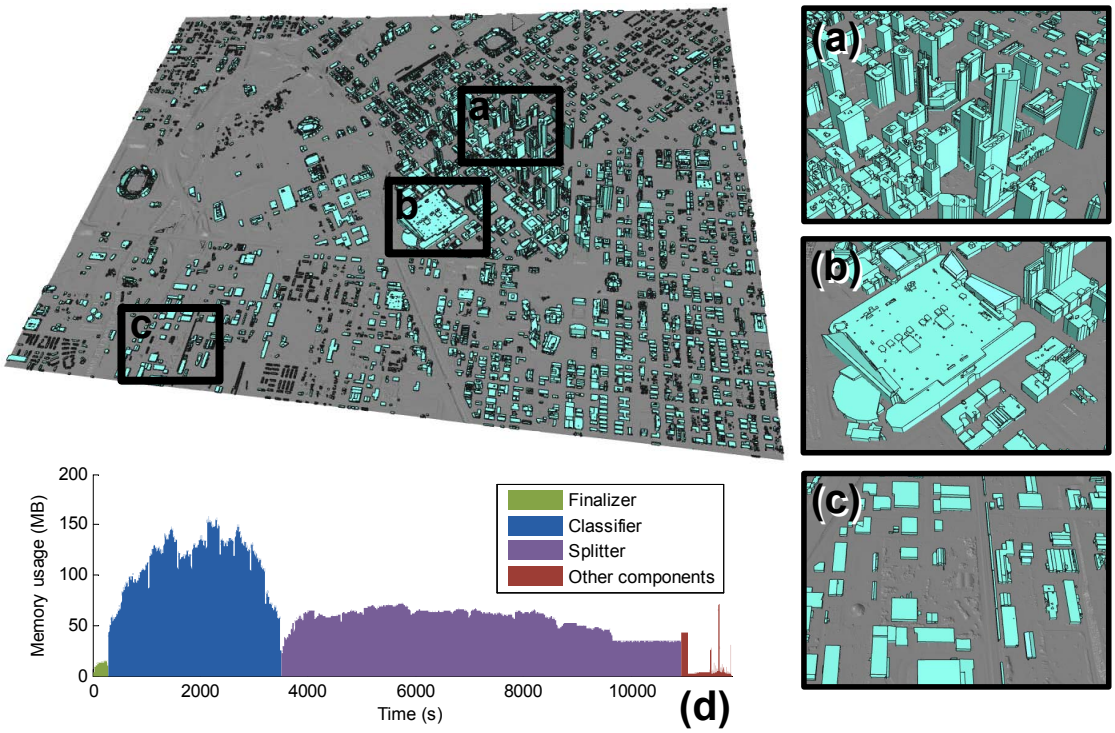


Figure 4.8: Urban model of Denver with three closeups shown in (a,b,c). Although principal directions in these areas are different; with the principal direction grid, correct principal directions are generated for each of them. (d) Memory usage during processing.

Figure 4.6 and Figure 4.8 demonstrate the reconstruction results for Atlanta and Denver respectively. The memory usage in the process of reconstructing the urban model of Atlanta city is plotted in Figure 4.7. The *Classifier* is the most memory consuming module because it has more active states, thus stores more cells in memory; the *Splitter* is the most time consuming module because of the overhead for saving segmented patches into files. For the Denver data set, its urban model and closeups are shown in Figure 4.8. Both data sets exhibit variation of principal directions across the whole city. Nevertheless, it is nicely handled by the grid-based principal direction estimation.

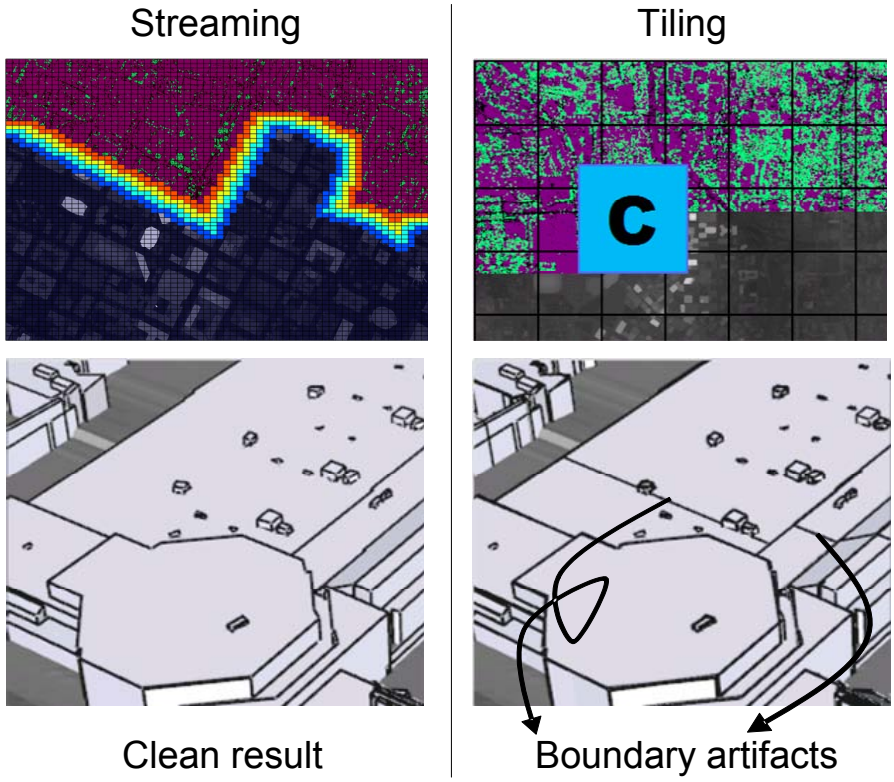


Figure 4.9: Comparison between streaming method (left) and tiling method (right). Even with padding along tile boundaries and tiles batch processed one-by-one using an in-core modeling program (the blue square shown in the top-right sub-figure), artifacts can be generated along tile boundaries as shown in the bottom.

4.4.1 Streaming Versus Tiling

This section makes a comparison between the streaming method and the traditional tiling method [41, 48]. In particular, the tiling method is tested on Atlanta data set following the partition setup described in [41], *i.e.*, $0.6km \times 0.6km$ tiles with $0.2km$ padding at each edge. The tiling grid is shown in the top-right sub-figure of Figure 4.9, with the blue square representing an in-processing tile with its padding. Each tile is processed using the general urban modeling pipeline proposed in Section 3. Following observations are made:

1. Large building structures that are not fully captured by one single tile still exhibit boundary artifacts. These building structures include the largest building shown in Figure 4.6(a). Boundary artifacts are produced by the tiling method as shown in the bottom-right sub-figure of Figure 4.9.

2. Tiling disables global segmentation. In some tiles, ground is segmented into several pieces and some are incorrectly detected as building structures. *E.g.*, with ramps connecting highways and local roads not in the same tile, the highway is often detected as an individual segment and reconstructed as a building; while with the streaming approach highways are appropriately detected as part of the ground.

3. Performance issue: with similar urban modeling technique, the tiling method runs for over 60 hours mainly because of the additional overhead for padding areas (with padding, the data size grows to roughly 3 times to the original size); and requires user interaction (*e.g.*, the user needs to specify which reconstructed building structure is desired when the same building appears partially in neighboring tiles); while the streaming approach runs fully automatically for 24 hours on the same computer.

Moreover, the streaming approach is especially useful under the current trend of rapidly increasing data resolution. *E.g.*, the Atlanta data set is with 17 samples/m^2 resolution, compared with 1 sample/m^2 data set in [41] and 9 samples/m^2 data set in [60].

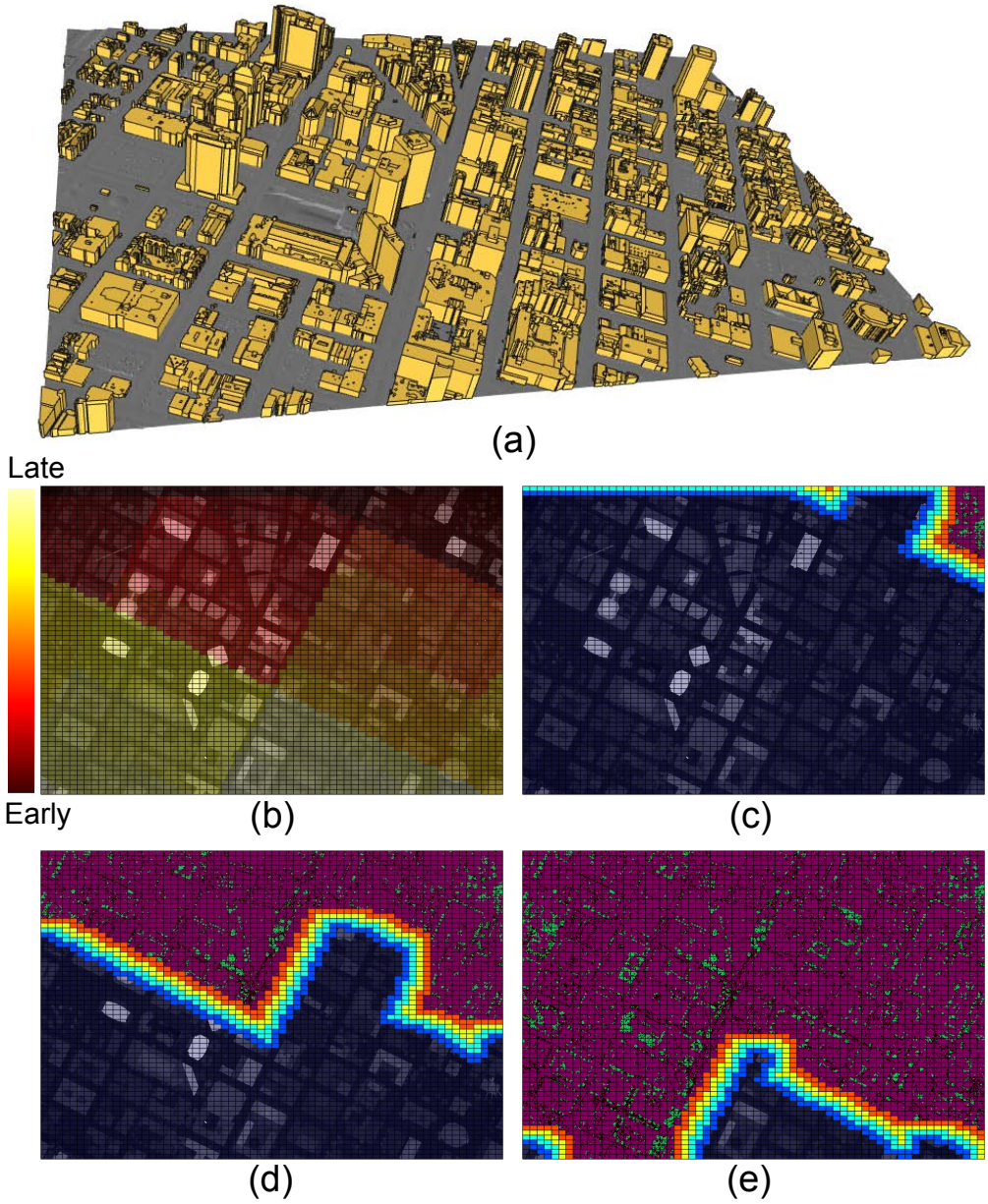


Figure 4.10: (a) Reconstructed urban model of Oakland downtown area. (b) Finalization result reveals the spatial coherence between cells; colors represent the finalization time. (c,d,e) Three snapshots during the streaming classification algorithm; only a small portion of cells are at active states (bright solid colored cells).

Chapter 5

2.5D Dual Contouring

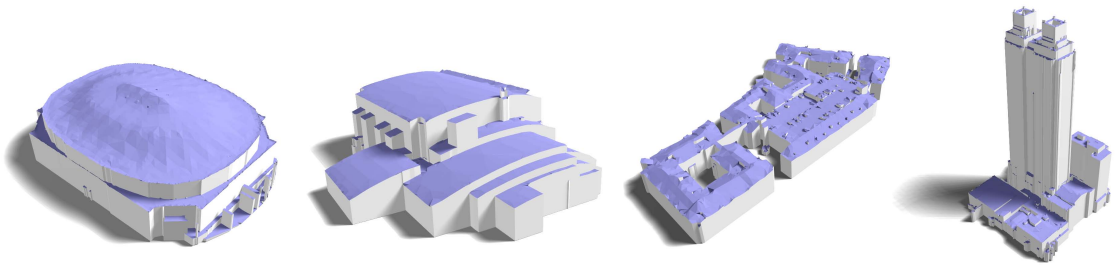


Figure 5.1: Various kinds of building models are created using 2.5D dual contouring. From left to right: two stadium models with different kinds of non-planar roofs; a typical flat building from residential area; and a modern high building from urban area.

This chapter focuses on the complicated problem of reconstructing building models from aerial LiDAR point cloud. The aerial LiDAR point clouds are 2.5D data, *i.e.*, the LiDAR sensor captures the details of roof surfaces, but collects few points on building walls connecting roof boundaries. In addition, manually created building models (Figure 5.2) also show a 2.5D characteristic. Nearly all of them consist of complex roofs (green faces) connected by vertical walls (white faces). Thus, a 2.5D modeling method with the following properties is desired:

- **Accuracy:** The method should produce simple polygonal models fitting the input point clouds in a precise manner.
- **Robustness:** Regardless of the diversity and complexity of building roof shapes, the method should always generate crack-free models, even with the existence of undesired elements such as residual sensor noise and small roof features.

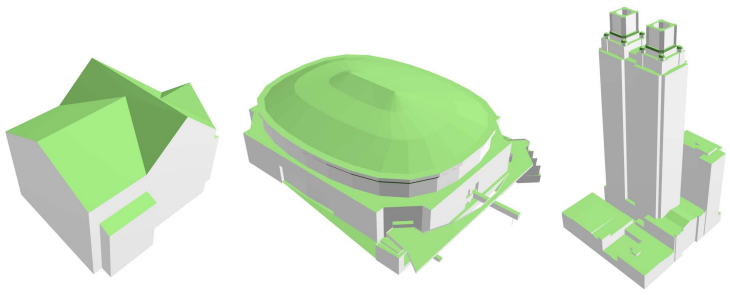


Figure 5.2: Manually created models in Google 3D warehouse [21], showing the 2.5D nature of building structure models

- **2.5D characteristic:** The method should create 2.5D polygonal models composed of detailed roofs and vertical walls connecting roof layers.

In Section 3.4.3, an automatic building reconstruction algorithm is developed based on planar roofs. Its extension supports non-planar roofs with the help of user interactions. However, both methods are limited by the pre-defined roof patterns, *i.e.*, planes or geometry patterns in a primitive library. Thus, they can hardly produce building models with arbitrarily shaped roofs.

This limitation is also a common disadvantage in previous research work. *E.g.*, [41, 48, 60] create planar building roofs and [33, 64, 65] rely on a small set of user-given primitives. These methods work well for buildings composed of pre-defined shapes, but lose accuracy and robustness when dealing with arbitrary roof shapes such as those shown in Figure 5.1.

Another way to attack the building modeling problem is with traditional data-driven approaches. Polygonal models are first generated directly from input data using rasterization or delaunay triangulation, then simplified with general mesh simplification algorithms. The latter step significantly reduces triangle number while preserving a low fitting error. However, since the general simplification algorithms are usually "blind" to the 2.5D nature of the problem, they can hardly produce models satisfying the 2.5D requirement.

This chapter proposes a novel, data-driven approach to solve this problem, named *2.5D dual contouring*. Like the classic dual contouring [31], an adaptive grid is adopted as the supporting data structure. Geometry is reconstructed in each grid cell by minimizing the quadratic error functions known as QEFs. Model simplification is easily achieved by merging grid cells and combining QEFs.

In order to represent the detailed roof surfaces, 2.5D dual contouring works in a 3D space. However, unlike the classic 3D dual contouring, it uses a 2D grid as the supporting data structure. The 2.5D dual contouring approach generates a *hyper-point* in each grid cell, which contains a set of 3D points having the same x-y coordinates, but different z values. They can be regarded as a set of points intersected by a vertical line and multiple roof layers. Hence, the consistency between boundary footprints of different roof layers is guaranteed, and vertical walls are produced by connecting neighboring hyper-points together.

The following part of this chapter is organized as follows: Section 5.1 reviews the classic dual contouring. Section 5.2 presents the 2.5D dual contouring pipeline, followed by sections describing the details of each step. Finally, experimental results are shown in Section 5.8.

5.1 Brief Review of Dual Contouring

Like many volumetric methods [10, 40], dual contouring [31] has proved to be a robust way of generating crack-free 3D models: input points are first scan-converted into a regularized grid; then geometry and topology are created respectively.

In particular, the dual contouring method takes a 3D grid of Hermite data (in the form of point-normal pairs) as input, and creates exactly one mesh vertex in each minimal grid cell by optimizing a quadratic error function (QEF), defined as:

$$E(x) = \sum_i (n_i \cdot (x - p_i))^2, \tag{5.1}$$

where (p_i, n_i) are the Hermite data samples in this cell. Since the error function is defined based on both positions and normals at intersection points of the surface with grid edges, the optimized vertices have a trend to lie on the sharp features such as roof ridges and valleys. Based on this observation, Fiocco *et al.* [13] use classic 3D dual contouring to create 3D building models from both aerial and ground-based LiDAR, preserving sharp building features.

The geometry simplification in dual contouring is achieved in an adaptive manner. Ju *et al.* [31] introduce an octree for 3D geometry simplification. They merge QEFs associated with leaf nodes while collapsing the octree structure. The sub-tree collapsing is controlled by the residual of the merged QEF, which is required to be less than a given tolerance. Finally, dual contouring creates polygons during a traversal over the adaptive grid. A topology safety examination is introduced to avoid possible topology changes during simplification.

Nevertheless, the classic dual contouring approach is designed with regular 2D or 3D grids. It does not satisfy the 2.5D requirement for building models.

5.2 2.5D Dual Contouring Pipeline

Given a building point cloud as input, the 2.5D dual contouring modeling process executes four steps as illustrated in Figure 5.3:

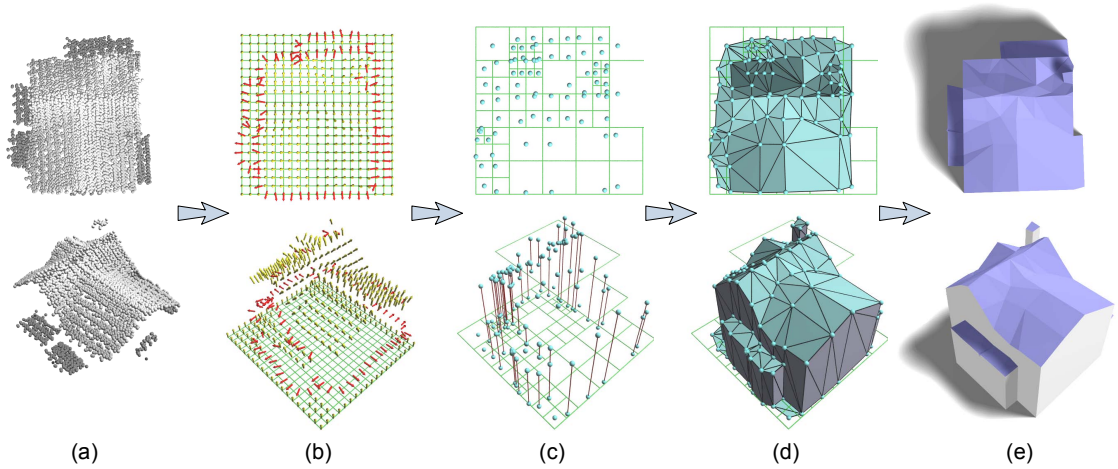


Figure 5.3: Robust building modeling pipeline: (a) the input point cloud; (b) a 2D grid with surface Hermite data (gold arrows) and boundary Hermite data (red arrows) attached; (c) hyper-points (turquoise balls connected by red lines) generated by minimizing QEFs; (d) mesh model reconstructed via 2.5D dual contouring; and (e) final model with boundaries snapped to principal directions.

1. **Scan conversion:** The point cloud is embedded in a uniform 2D grid. *Surface Hermite data* samples (gold arrows) are generated at grid points and *boundary Hermite data* samples (red arrows) are estimated on grid edges connecting different roof layers (Figure 5.3(b)). This 2D grid is also regarded as the finest level of the supporting quadtree.

2. **Adaptive creation of geometry:** In each quadtree cell, a *hyper-point* is estimated by minimizing a 2.5D quadratic error function (2.5D QEF). Geometry simplification is achieved in an adaptive manner by collapsing subtrees and adding QEFs associated with leaf cells (Figure 5.3(c)).

3. **Polygon generation:** A watertight mesh model is created by connecting hyper-points with *surface polygons* (turquoise triangles) and *boundary polygons* (purple triangles), which form building roofs and vertical walls, respectively (Figure 5.3(d)).

4. **Principal direction snapping:** The roof boundaries are refined to follow the principal directions defined in Section 3.4.3 and Section 4.3.4 (Figure 5.3(e)).

5.3 2.5D Scan Conversion

The first step of the 2.5D dual contouring algorithm converts the input point cloud into a volumetric form, by sampling Hermite data (a point-normal pair) over a 2D supporting grid. With elements being considered as their infinite extensions along the vertical direction, this 2D grid has a 3D volumetric connotation. *E.g.*, a grid cell represents an infinite three dimensional volume, while a grid point corresponds to a vertical line containing it.

5.3.1 Surface Hermite Data

Given a 2.5D point cloud as input, the scan conversion algorithm first segments it into multiple roof layers using a local distance-based region growing algorithm[1], as shown in Figure 5.4(a). Ideally, each vertical line passing through a grid point intersects with one and only one roof layer. The intersection point is taken as a *surface Hermite data* sample, and estimated by averaging the heights and normals of its k-nearest input points within the same roof layer, illustrated as points marked with blue or purple outlines (taking $k = 4$) in Figure 5.4(a).

The only difficulty in this process is to robustly detect the right roof layer crossing the vertical line. Intuitively, a roof layer L *covers* a grid point g *iff* each of g's four neighboring cells contains at least one input point p belonging to L or a higher cluster L'. For example, in Figure 5.4(a), point A is covered by no roof layers, and thus is assigned as ground; point B is only covered by and assigned to the dark-grey layer.

[1]The roof layers are always segmented in a local area, as global segmentation may erase local features such as those shown in Figure 5.9(c). Specifically, the segmentation for grid point g is applied to all the input points in g's four neighboring cells.

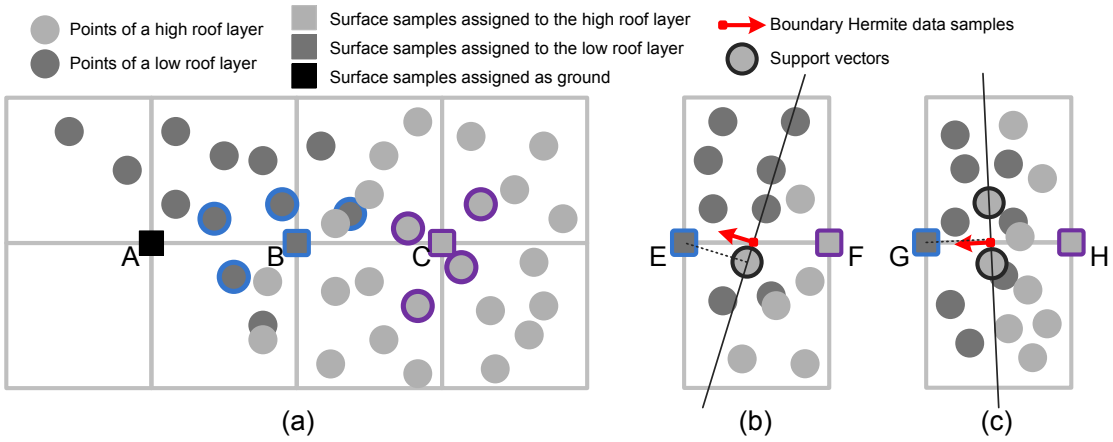


Figure 5.4: Generating (a) surface Hermite data samples on grid points: the sample is assigned to the highest roof layer which *covers* the grid point; (b,c) boundary Hermite data samples on grid edges: the maximum margin line (thin black lines) divides the lower surface Hermite data sample from the higher roof layer.

Note point C is covered by both the dark-grey layer and the light-grey layer. In this case, the scan conversion algorithm takes the highest roof layer covering point C, *i.e.*, the light-grey layer.

5.3.2 Boundary Hermite Data

While surface Hermite data captures the surface geometry of building roofs, the shapes of roof boundaries are represented by the *boundary Hermite data*.

Considering a grid edge e connecting two grid points with surface Hermite data samples $\{s_0, s_1\}$ on different roof layers $s_0 \in L_0, s_1 \in L_1$,[2] the vertical wall connecting L_0 and L_1 should split their projection images on the x-y plane. Inspired by the 2D Support Vector Machine algorithm [2], a maximum-margin line l is detected which separates L_0

[2]To avoid ambiguity, roof layers are determined again by a local segmentation over $\{s_0, s_1\} \cup P$, where P is the input point set within e's two adjacent cells.

and L_1 on the x-y plane. The boundary sample is estimated by intersecting line l and edge e.

In practice, with the existence of residual sensor noise, the projections of different roof layers may overlap on the x-y plane. Since aerial LiDAR data is collected from a top view, more saliency is given to the higher roof layer L_1 (assuming $height(L_0) < height(L_1)$). Thus the scan conversion algorithm takes the maximum-margin line l which separates $\{s_0\}$ and L_1 while maximizing $distance(s_0, l)$, shown as the thin black lines in Figure 5.4(b,c). Empirically, this method is more robust than other methods including that using a maximum-soft-margin line dividing L_0 and L_1.

5.4 Adaptive Creation of Geometry

Given a quadtree cell c (not necessarily being a finest-level leaf cell), the set of surface Hermite data samples on the grid points in c is denoted as S, and the set of boundary Hermite data samples on atomic grid edges in c is denoted as B. The roof layers in c are then determined by segmenting S into k clusters $S = S_1 \cup \cdots \cup S_k$. Intuitively, if an atomic grid edge in c has no boundary sample attached, it connects two surface samples of the same roof layer. Thus, an agglomerative clustering algorithm is adopted to repeatedly combine surface sample sets connected by edges without boundary samples.

Now the task is to generate k vertices for the k roof layers, denoted as a *hyper-point* $\chi = \{x_1, \ldots, x_k\}$. To maintain the consistency of roof layer boundaries, these k vertices are required to have the same projection on the x-y plane, *i.e.*, they should have the same x-y coordinates, but different z values. Thus χ can be expressed as a $k + 2$ dimensional vector $\chi = (x, y, z_1, \ldots, z_k)$. Let $x_0 = (x, y, 0)$ for convenience in following discussions.

5.4.1 2.5D Quadratic Error Function

The hyper-point χ is optimized by minimizing a 2.5D quadratic error function (2.5D QEF) defined as the linear combination of 2D boundary quadratic errors and 3D surface quadratic errors:

$$E(\chi) = \sum_{(p,n)\in B} (\omega n \cdot (x_0 - p))^2 + \sum_{i=1,\ldots,k} \sum_{(p,n)\in S_i} (n \cdot (x_i - p))^2 \tag{5.2}$$

where ω is a user-given weight balancing between boundary samples and surface samples. Empirically, a weight between $1 \sim 4$ satisfies most of the experiments.

Due to the horizontality of boundary sample normals, the third coordinates of p and x_0 do not affect the 2D error term. However, I choose to write all these variables uniformly in 3D, in order to express the energy function in a matrix product form:

$$E(\chi) = (A\chi - b)^T (A\chi - b) \tag{5.3}$$

where A is a matrix whose rows come from normals in $B, S_1, \ldots, S_k$, with those in B multiplied by ω. The x-y values of each normal are placed in the first two columns, while the z values of normals in S_i are placed in the $(i+2)$-th column. The remaining entries in A are padded with zeros. b is a vector composed of corresponding inner products $n \cdot p$ with the first $|B|$ entries multiplied by ω.

The numerical stability during QEF optimization can be improved by employing the QR decomposition as proposed in [31]. *I.e.*,

$$(A \quad b) = Q \begin{pmatrix} \hat{A} & \hat{b} \\ 0 & r \\ 0 & 0 \\ \cdots & \cdots \end{pmatrix} \tag{5.4}$$

where Q is an orthogonal matrix and Equation 5.3 can be rewritten as:

$$E(\chi) = (A\chi - b)^T QQ^T (A\chi - b) = (\hat{A}\chi - \hat{b})^T (\hat{A}\chi - \hat{b}) + r^2. \tag{5.5}$$

Thus, $E(\chi)$ is minimized by solving $\hat{A}\chi - \hat{b} = 0$. To handle the possible singularity of $\hat{A}$, an SVD decomposition is applied, following solutions in previous methods [31, 37]:

$$\hat{A} = U\Sigma V^T. \tag{5.6}$$

Small singular values in Σ with a magnitude of less than 0.1 is truncated, and the pseudo-inverse Σ^+ is adopted to compute the hyper-point χ as:

$$\chi = \bar{\chi} + V\Sigma^+ U^T (\hat{b} - \hat{A}\bar{\chi}) \tag{5.7}$$

where $\bar{\chi}$ is a guessed solution whose first two coordinates come from the centroid of B, and the $(i+2)$-th coordinate is the mean height of samples in S_i. If B is empty, the first two coordinates equal to those of the centroid of S.

5.4.2 Quadtree Simplification with QEFs

Taking a quadtree with QEF matrices pre-computed for all the finest-level cells, the geometry is simplified by collapsing leaf cells into parent cells and combining QEFs in a bottom-up manner. A user-given tolerance δ controls the simplification level by denying sub-tree collapse when the residual is greater than δ.

Combining four regular 3D QEFs can be simply achieved by merging the rows of their upper triangular matrices to form a 16×4 matrix [31]. 2.5D QEF matrices are combined in a similar way, yet with the consideration of association between matrix columns and roof layers: as roof layers in leaf cells merge into one roof layer in the parent cell, corresponding matrix columns are placed in the same column of the combined matrix. Specifically, the roof layer is re-segmented in the parent cell before merging matrices. Assuming the i-th roof layer in a leaf cell belongs to the j-th roof layer in the parent cell, the $(i+2)$-th column of the leaf cell matrix is put into the $(j+2)$-th column of the combined matrix. 0-columns are used to pad the leaf cell matrices where no roof layers belong to certain roof layers in the parent cell.

Once again, the merged matrix is brought to the upper triangular form via a QR decomposition. Due to the orthogonality of involved transformation matrices, it represents the 2.5D QEF in the parent cell.

Figure 5.5 shows a building model without and with geometry simplification. The triangle number decreases from 820 to 252 with an insignificant increase of the fitting error.

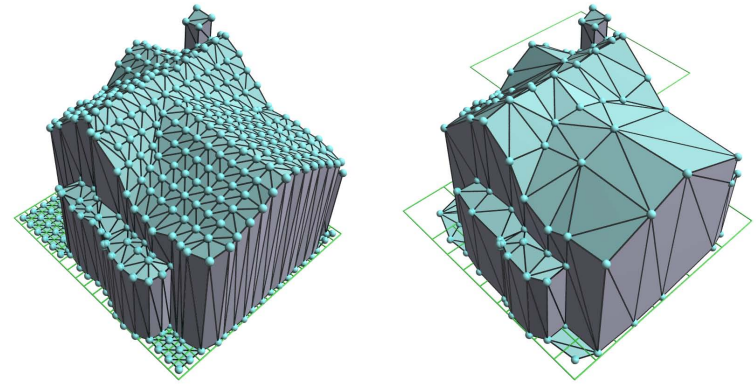


Figure 5.5: 2.5D dual contouring without (left) and with (right) adaptive geometry simplification

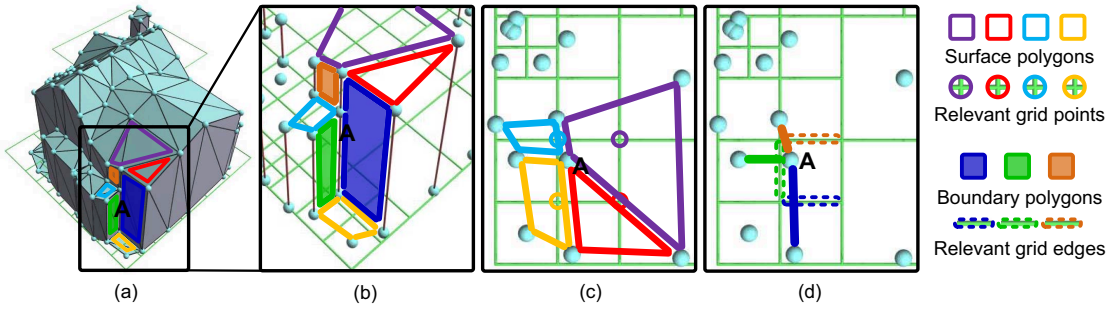


Figure 5.6: (a,b) Creating surface polygons (colored hollow polygons) and boundary polygons (colored semitransparent polygons) around hyper-point A. Viewing from top, (c) surface polygons are generated at grid points, while (d) boundary polygons are produced for grid edges which exhibit a roof layer gap.

5.5 Polygon Generation

Given the simplified quadtree with hyper-points estimated in each leaf cell, the next task is to create polygons connecting these hyper-points into a mesh. In particular, two kinds of polygons are generated to satisfy the 2.5D characteristic.

1. **Surface polygons:** At each grid point p, a surface polygon is created by connecting vertices in the hyper-points on the same roof layer as p in its neighboring cells.

2. **Boundary polygons:** At each minimal quadtree edge e, a boundary polygon is generated connecting two hyper-point segments in the adjacent cells.

Figure 5.6 shows an example of polygon generation around a hyper-point A. The surface polygons and boundary polygons are highlighted with colored outlines and colored semitransparent polygons respectively. To avoid cracks generated within a hyper-point, a boundary polygon sequentially passes through the vertices in a hyper-point segment in height ascending or descending order. *E.g.*, the dark-blue boundary polygon in Figure 5.6 goes through all the three vertices in hyper-point A, from the top vertex to the bottom vertex.

This polygon generation method is guaranteed to produce crack-free models, which can be derived from the fact that except for the border edges created around the entire grid, the other mesh edges are contained by an even number of polygons. Proof is straightforward: a non-vertical mesh edge is either contained by two surface polygons, or by one surface polygon and one boundary polygon. As for the vertical mesh edges created within a hyper-point, considering all the boundary polygons around this hyper-point (*e.g.*, the colored semitransparent polygons shown in Figure 5.6(a,b)): they go up and down though this hyper-point and finally return to the start vertex, forming up a closed edge loop. Thus, each vertical mesh edge in this hyper-point appears even times.

5.5.1 Sharp Feature Preserving Triangulation

By minimizing QEFs, 2.5D dual contouring has the ability to produce vertices lying on sharp features, which are a common pattern in building roofs. However, a poor triangulation of surface polygons can spoil this advantage, as shown in Figure 5.7 left. To solve this problem, an efficient sharp feature detection algorithm is proposed to preserve these features once detected.

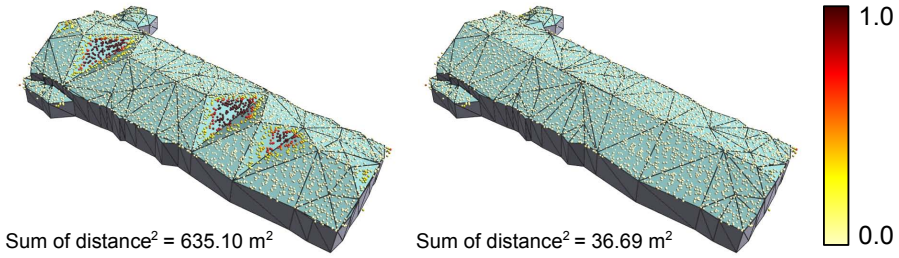


Figure 5.7: Triangulation without (left) and with (right) the sharp feature preserving algorithm. The colors of input points represent the squared distances from the mesh.

In a grid cell c containing only one roof layer, covariance analysis is applied over the normals of all surface samples, *i.e.*, to get the eigenvalues of matrix:

$$C = \frac{1}{N} \sum_{i} n_i \cdot n_i^T. \tag{5.8}$$

Since c has no boundary Hermite data samples, Equation 5.4 and 5.6 can be used to simplify this matrix as:

$$C = \frac{1}{N} A^T A = \frac{1}{N} \hat{A}^T \hat{A} = \frac{1}{N} V \Sigma^T \Sigma V^T. \tag{5.9}$$

Thus, the diagonal of matrix $\frac{1}{N}\Sigma^T\Sigma$ gives the eigenvalues of C, while the columns of V are corresponding eigenvectors. As Pauly [47] suggests, the smallest eigenvalue λ_0 and the middle eigenvalue λ_1 estimate the minimal and maximal curvatures, as the corresponding eigenvectors v_0, v_1 point to the curvature directions. Therefore, ridges and valleys are detected by finding vertices with small λ_0 and fairly large λ_1. v_0 is then adopted as the feature direction. Since the involved matrices have all been computed in previous steps, the additional overhead of this algorithm is trivial.

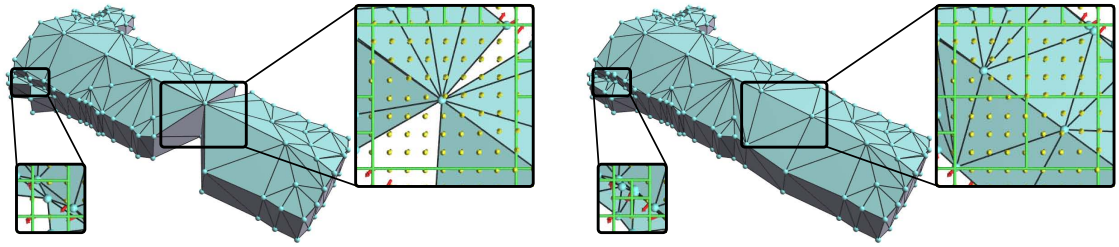


Figure 5.8: Comparison between topology-unsafe simplification (left) and topology-safe simplification (right). Undesired features can be created by merging leaf cells in a topology-unsafe manner.

Specifically, for each diagonal e of a surface quad, the triangulation algorithm calculates:

$$\sum_{p \in e \text{ and } \lambda_0(p) < \tau} \lambda_1(p) \cdot |v_0(p) \cdot e| \tag{5.10}$$

and chooses the diagonal e^* which maximizes this value to split the quad into two triangles. Here τ is a user given threshold. Empirically, $\tau = 0.01$.

5.6 Topology-Safe Simplification

So far the quadtree simplification is completely built on QEFs, and the topology of output models may change during this process. Undesired features can be generated as shown in Figure 5.8 left. To solve this problem, an additional topology test is introduced right before sub-tree collapse happens. It rejects collapse if there is a danger of topology change. Regarding multiple roof layers as multiple materials, topology test algorithm in [31] is extended with an additional test (step 3) which prevents different roof layers in one leaf cell (top-left cell in Figure 5.9(a)) from merging into a same roof layer in the coarse cell (Figure 5.9(b)). This situation may cause removal of small vertical wall features (*e.g.*, Figure 5.9(c)).

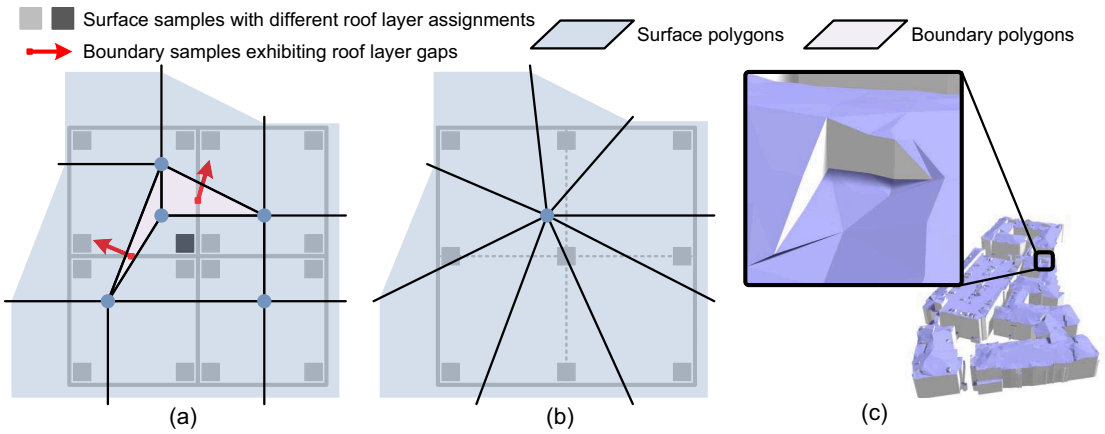


Figure 5.9: An unsafe simplification case denied by the topology safety test step 3. Since the center grid point has different roof layer assignments in these leaf cells, two different layers in the top-left leaf cell (a) belong to the same roof layer in the coarse cell (b). Unsafe merging may erase wall features such as the one shown in (c).

1. Test whether each leaf cell creates a manifold; if not, stop.

2. Test whether the coarse cell creates a manifold; if not, stop.

3. Test whether any two roof layers in a same leaf cell belong to two different roof layers in the coarse cell; if not, stop.

4. Test whether the topology of the dual contour is preserved using following criteria; if not, stop; otherwise, collapse.

   (a) Test whether the roof layer on the middle point of each coarse edge agrees with the roof layer on at least one of the two edge endpoints.

   (b) Test whether the roof layer on the middle point of the coarse cell agrees with the roof layer on at least one of the four cell corners.

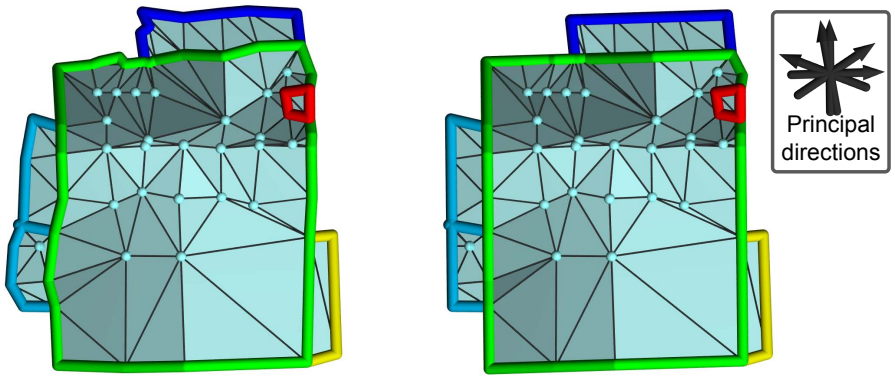


Figure 5.10: Roof layer boundaries (thick colored lines) are regularized using principal direction snapping algorithm.

5.7 Principal Direction Snapping

The 2.5D dual contouring algorithm is completely data-driven, *i.e.*, no pre-assumptions about the roof shapes have been made. Thus this algorithm can handle complex roofs in a robust manner. On the other hand, in some cases, prior knowledge of the urban area is given and it is a desire to have building models concurring with such knowledge. This section shows a post-processing refinement to the modeling results using the prior knowledge of principal directions as detected in Section 3.4.3 and Section 4.3.4.

The idea is straightforward: once the boundaries of individual roof layers are extracted, they can be snapped to the principal directions as much as possible without exceeding a small error tolerance. In order to maintain the consistency between boundaries of different layers, the boundaries are handled one by one in height-descending order. *I.e.*, when a roof layer boundary has been processed, the x-y coordinates of the touched hyper-points are fixed, which are then considered as constraints during the subsequent processing of lower roof layers. Figure 5.10 shows clean and simple roof boundaries generated by the principal direction refinement.

5.8 Experimental Results of 2.5D Dual Contouring

Figure 5.11 shows an urban area of Los Angeles reconstructed from 26M LiDAR points with 7 samples/m^2 resolution. The streaming urban modeling system presented in Chapter 4 is first adopted to remove irrelevant parts such as noises, trees, vehicles and even ground. The 2.5D dual contouring algorithm is then tested on point clouds of individual buildings. This algorithm successfully creates 1,879 building models consisting of 857K triangles within 6 minutes on a consumer-level laptop (Intel Core 2 1.8GHz CPU with 2GB memory). 2.5D building models with complex roofs are robustly generated in the entire area.

To further demonstrate the ability of handling various kinds of building models, the 2.5D dual contouring method is tested on a set of buildings from the city of Atlanta, as illustrated in Figure 5.1. Figure 5.12 shows a comparison between this method and the state-of-the-arts. In particular, I compare the average squared distance from input point sets to the generated models, and the ratio of points with squared distances greater than 1m^2 In Figure 5.12, point colors denote the squared distances, and the colored bars show the percentage of points at different squared distance levels. As the quantitative results in Table 5.1 illustrate, the 2.5D dual contouring method (first column) is the most accurate algorithm to produce 2.5D models. Plane-based approaches such as the one in Section 3.4.3 (second column) are unable to handle non-flat roofs (a,d) and small roof features (b,e). Cracks often exist when fitting is unsuccessful (c,d). A general mesh simplification over the DEM (third column) is competitive in the sense of fitting quality. However, it cannot produce 2.5D models composed of roofs and vertical walls. In addition, the fitting quality on roof boundaries is unsatisfactory (f,g,h). The last column demonstrates point clouds aligning with manually created models. Designed without knowledge from real-world data, they often lack of accuracy even after registration to the input points.

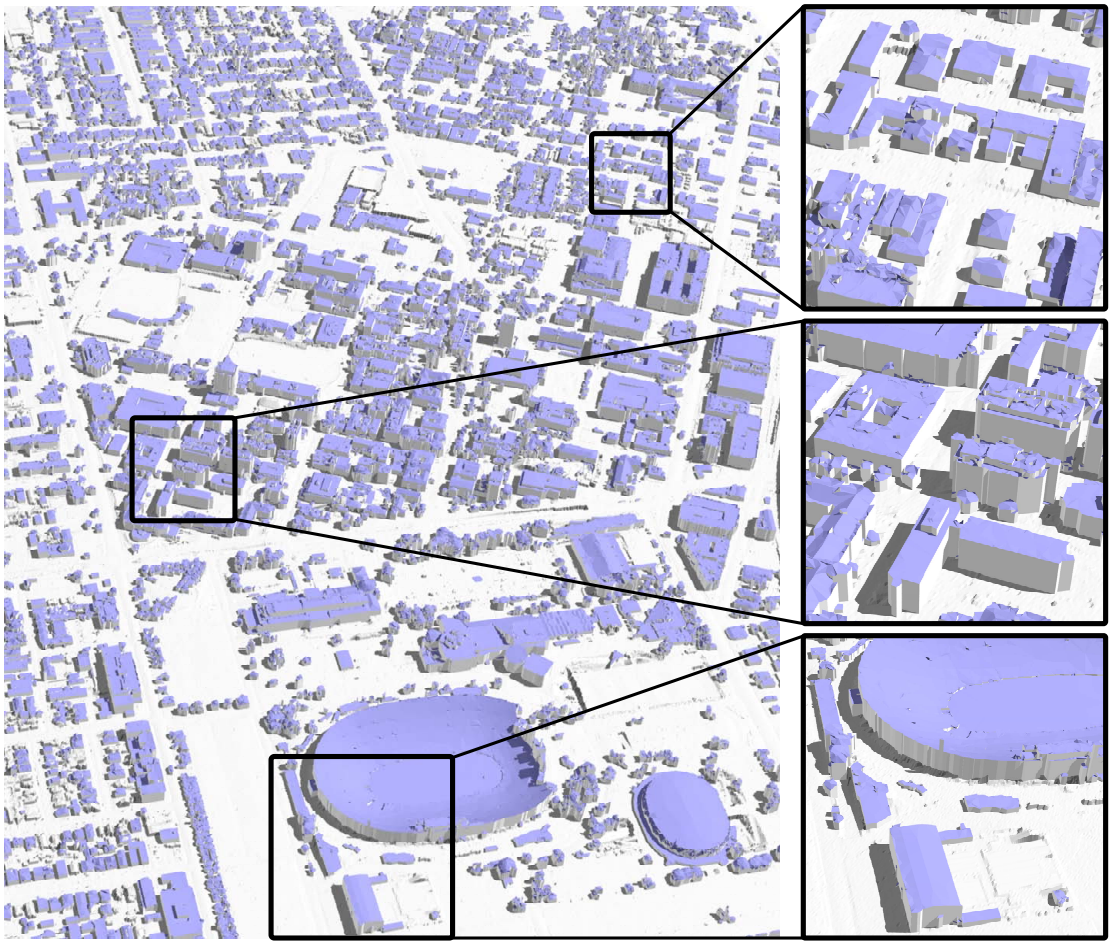


Figure 5.11: Building reconstruction for a 2KM-by-2.5KM urban area of Los Angeles

Figure 5.13 finally demonstrates the influence of grid configuration. As an adaptive approach, 2.5D dual contouring is insensitive to the grid size (top row). In addition, it has the ability to place vertices at optimal positions, thus grid orientation affects the results insignificantly (bottom row).

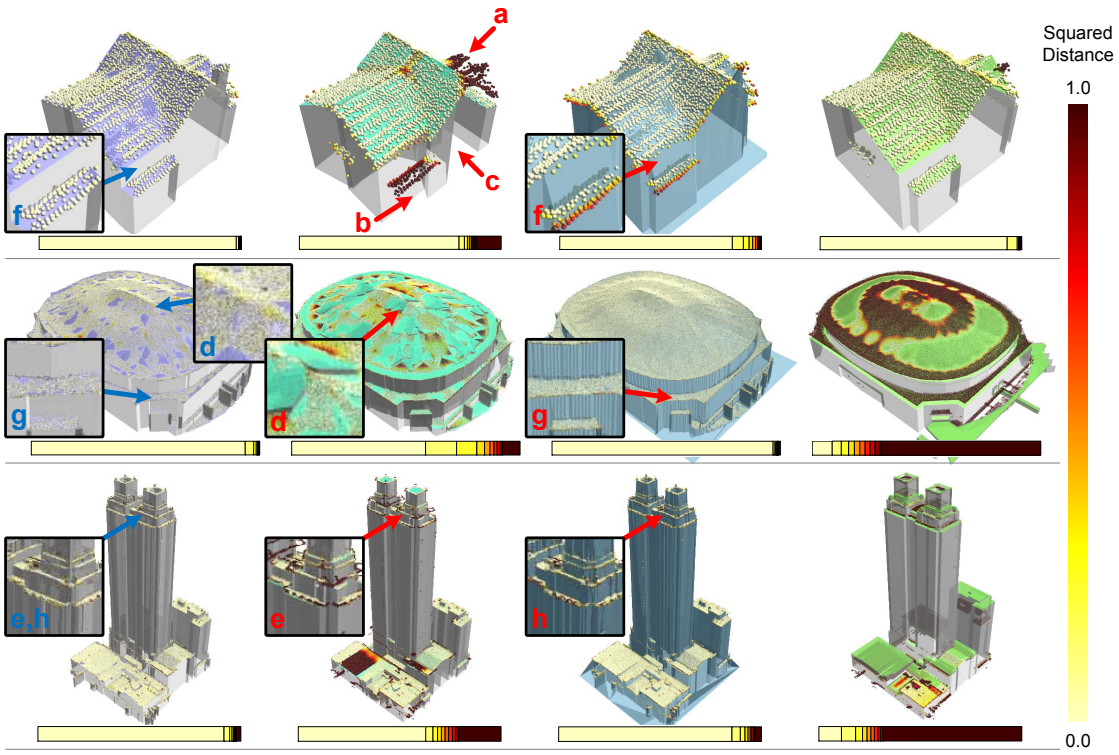


Figure 5.12: Building models created using different approaches (from left to right): 2.5D dual contouring, plane-based method proposed in Section 3.4.3, general mesh simplification over a rasterized DEM, and manual creation. Point colors denote the squared distances between points and generated models. Color bars under the models show the ratio of points at different squared distance level.

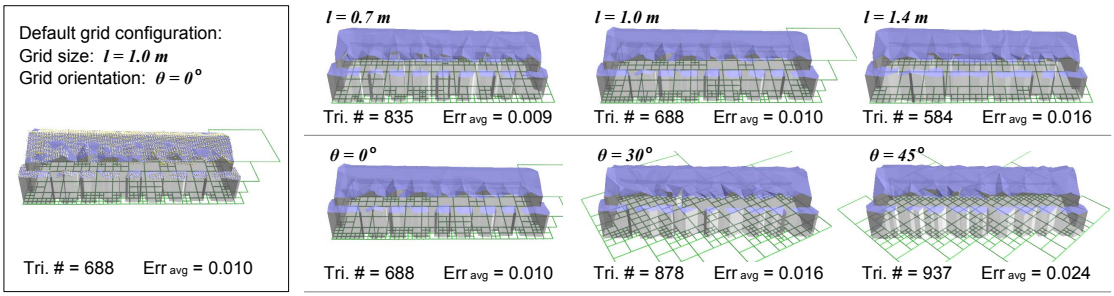


Figure 5.13: Models of similar quality are generated with the same point cloud embedded into grids of different sizes or different orientations.

Models in Figure 5.12		2.5D dual contouring	Plane-based method	DEM simplification	Manual creation [21]
First row (4679 points)	Triangle number	214	76	198	78
	Avg. distance^2	0.016	0.599	0.061	0.058
	Outlier ratio	0.06%	12.37%	0.53%	0.83%
Second row (684907 points)	Triangle number	8009	6262	8000	1227
	Avg. distance^2	0.037	0.465	0.035	7.780
	Outlier ratio	0.44%	7.93%	0.87%	70.38%
Third row (198551 points)	Triangle number	12688	1619	12999	1558
	Avg. distance^2	0.203	1.610	0.264	16.220
	Outlier ratio	2.03%	21.15%	3.08%	68.28%

Table 5.1: Quantitative evaluation of four modeling approaches over models shown in Figure 5.12

Chapter 6

2.5D Building Topology

This chapter studies 2.5D building topology, and extends 2.5D dual contouring into a 2.5D building modeling method with topology control.

The major contribution of this chapter is based on the observation that human vision tend to be more sensitive to building topology rather than building geometry. Intuitively, building topology determines the existence of structural pieces and the connections between them; while building geometry describes where these structural pieces appear in the three dimensional space. Humans tend to be more aware of changes in topology even if the related structural pieces are small. For example, Figure 6.1(c,d) demonstrate two building models created targeting to achieve more precise geometry and more precise topology respectively. Although the left model fits the input point cloud better under typical geometrical error measurements (*e.g.*, average quadratic distance), it is visually less convincing than the right one because a roof piece (the chimney) is missing.

In 2.5D dual contouring, the topology issue is alleviated by introducing a topology test presented in Section 5.6. The adaptive simplification process first collapses quadtree cells and optimizes an anchor point completely based on geometric errors without considering building topology; then rewinds the collapse operation if the topology test reveals a possible topology change. This strategy performs well under strong geometric control (*i.e.*, with a small geometry error tolerance). However, in cases where simpler models are desired thus looser geometric control is given, the number of topology test failures increases rapidly and they become the dominant factor in preventing quadtree

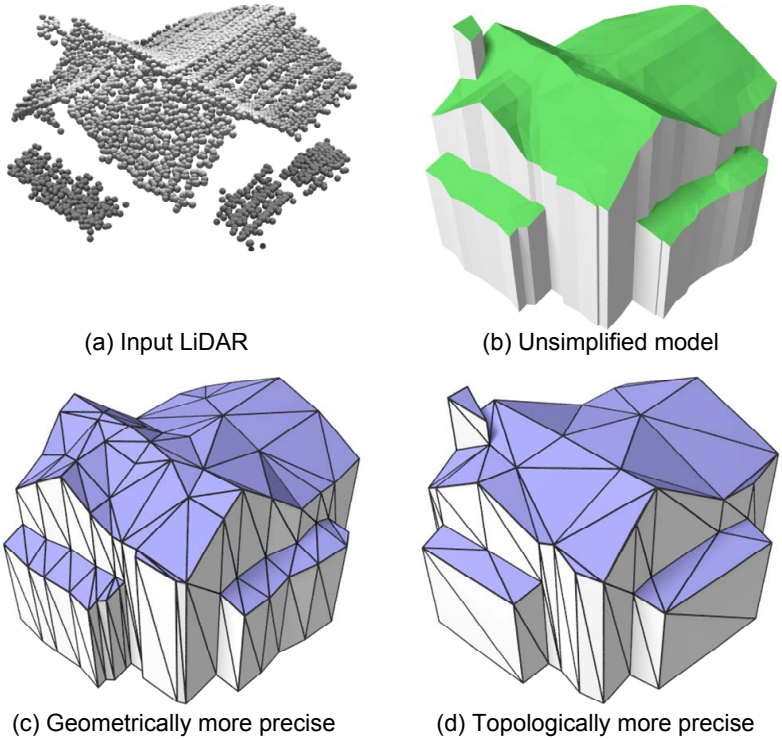


Figure 6.1: Building models reconstructed targeting to obtain (c) more precise geometry and (d) more precise topology respectively. Compared with (a) the input LiDAR and (b) unsimplified building model, the missing of the chimney makes the former one visually less convincing than the latter one.

collapse. Figure 6.2(a) shows such an example in which topology test frequently detects possible roof layer cracks and denies the cell collapse; therefore, numerous insignificant triangles are produced along the thin long roof features as shown in the closeup. The deep reason behind this problem is that the optimization process is completely unaware of building topology. It produces exact one hyper-point[1] per quadtree cell without discrimination. Hence, the most complicated topological structure that can exist in one cell is a conjunction hyper-point with star-shaped roof boundaries, as shown in Figure 6.2(a) bottom right. The adaptive simplification becomes problematic in producing building

[1]A hyper-point is defined as a series of 3D points having the same x-y coordinates but different z values, see Chapter 5.

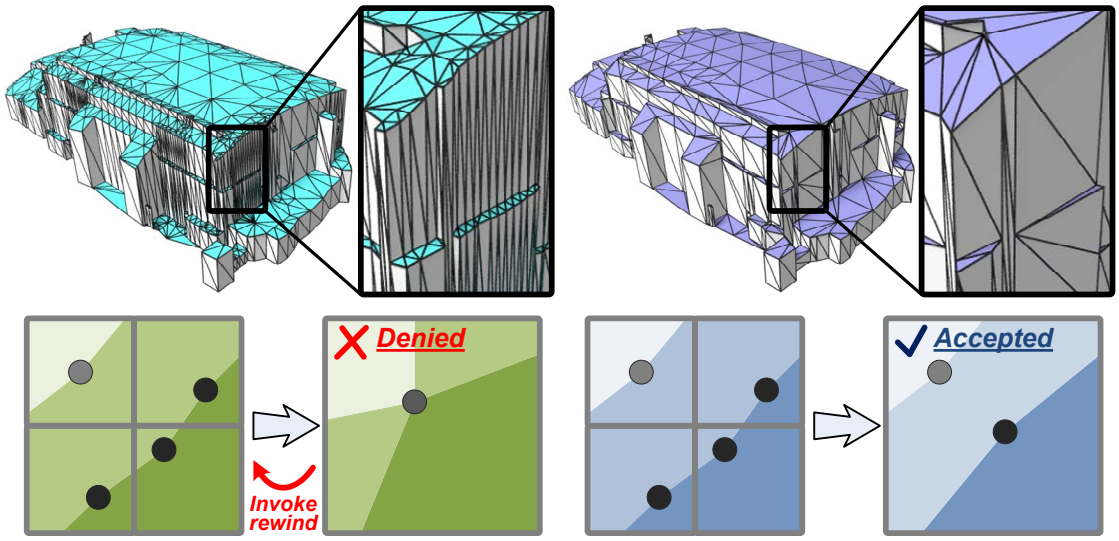


Figure 6.2: Comparison between (a) 2.5D dual contouring and (b) 2.5D building modeling with topology control. While the uniqueness of hyper-point in one cell prevents a flexible simplification in dual contouring, the new method detects and controls building topology beyond the rigid quadtree structure.

structures with topology that is more complex than a conjunction hyper-point. Collapse rewind is invoked frequently.

This chapter proposes an extension to the 2.5D dual contouring method to enable building topology control. The key idea is to maintain multiple hyper-points in one quadtree cell. Therefore complicated in-cell building topology is allowed. With this extension, the adaptive model creation procedure becomes less restrictive, and thus generates simpler building models in a flexible manner, *e.g.*, Figure 6.2(b). In particular, without changing building topology, the modeling method can produce building models with triangles as few as manually created models or primitive-based models; while it still provides a similar geometric optimization scheme as the data-driven modeling approaches.

Section 6.1 reviews topology control approaches in volumetric modeling. Section 6.2 formally defines 2.5D building topology which is the basis of this chapter. Section 6.3 then extends 2.5D dual contouring into a building modeling method with topology control. Experimental results are shown in Section 6.4.

6.1 Review of Topology Control in Volumetric Modeling

In classic 2D and 3D volumetric modeling methods, the topology issue is first noticed by Ju *et al.* [31]. They propose a topology test mechanism to reject simplification operations yielding possible topology changes. This mechanism is later extended to the 2.5D dual contouring method as discussed in Section 5.6.

The drawback of creating one vertex per octree/quadtree cell is noticed by researchers and different approaches have been proposed to solve this problem, *e.g.*, [51, 59, 66]. Generally, they all allow one grid cell to have more than one vertices, in order to track contour components which are topologically more complicated than a disk (or in 2D, a line segment) in each cell. Although these methods share some similarities with the approach presented in this chapter, two key differences are noted: first, this research aims at 2.5D building modeling involving hyper-points that cannot be handled in classic 2D or 3D manner; second, various building topology features are defined and processed which are more complicated than disk-like features in classic 2D or 3D space.

6.2 2.5D Building Topology

Considering a 2.5D dual contouring process without adaptive simplification: taking aerial LiDAR data as input, the contouring method builds up a uniform grid with Hermite data attached; creates one hyper-point (a series of points that are consistent on the x-y plane) in each cell by optimizing a quadratic error function; generates surface

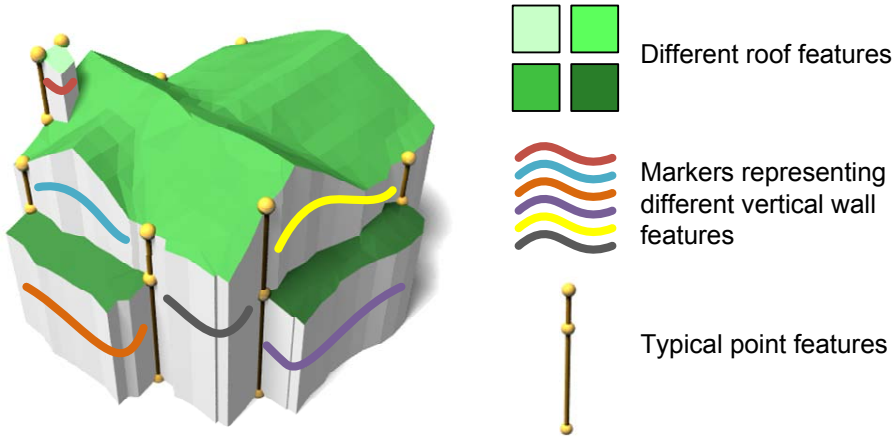


Figure 6.3: Topological features in an unsimplified 2.5D building model

polygons and vertical boundary polygons; and produces an unsimplified 2.5D building model such as the one shown in Figure 6.3 left.

6.2.1 Topological Feature Definitions

The first observation about 2.5D building topology is that a typical building structural piece (*e.g.*, a chimney) is usually composed of one unique roof patch and its surrounding walls. Hence,

Definition 6.1 A *roof feature* R is a connected component composed of non-vertical surface polygons.

Roof features are the key in determining 2.5D building structures. Figure 6.3 utilizes green mesh pieces rendered with different color intensities to represent multiple roof features. In particular, neighboring roof patches exhibit a height gap along their common boundary, which is sealed up by vertical boundary polygons (grey vertical polygons). These polygons form wall features that are marked in Figure 6.3 by curves with various colors.

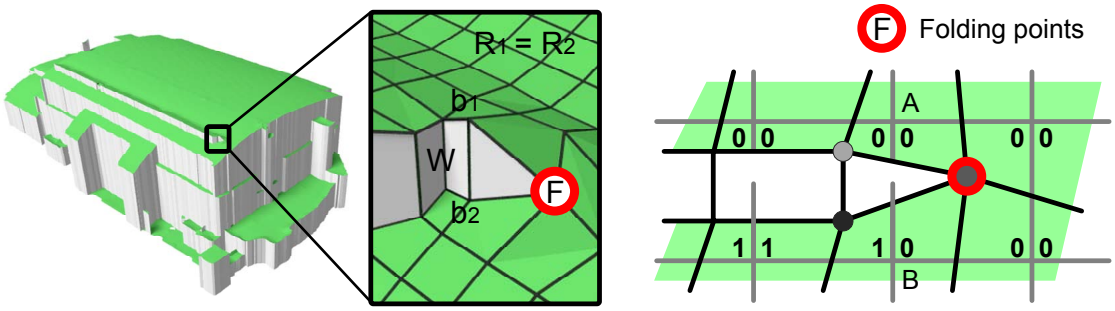


Figure 6.4: 2.5D building models may contain point features involving only one wall feature (left). These points are produced around grid edges (*e.g.*, AB) which detect inconsistent roof layer assignments in two adjacent cells (right).

Definition 6.2 Given two roof features R_1 and R_2, the corresponding *wall feature* W is defined as the connected component composed of vertical boundary polygons adjacent to R_1 and R_2 simultaneously. W intersects with R_1 and R_2 via non-identical roof boundary polylines.

Here a slight modification is made to 2.5D dual contouring, that the triangulation is disabled. Instead, surface polygons are treated as quads connecting all four vertices around a grid corner, while boundary polygons are produced with two non-vertical edges linking a pair of neighboring hyper-points. Therefore, a vertical boundary polygon is "adjacent" to a roof feature as long as they share a non-vertical edge. By tracking the consecutive non-vertical edges, one roof boundary polyline is created regarding each adjacent wall-roof feature pair $(W, R_i), i = 1, 2$, denoted as $b_i = W \cap R_i, i = 1, 2$. According to definition 6.2, W is a valid feature when b_1 and b_2 are not identical, even if R_1 and R_2 refer to the same roof patch.

Definition 6.3 Given a wall feature W sharing two consecutive roof boundary polylines b_1 and b_2 with R_1 and R_2 respectively, *point features* are defined as hyper-points which contain b_1 and b_2's end points if there is any.

Typically, a hyper-point shared by two neighboring wall features is a point feature. They are rendered in Figure 6.3 as golden balls connected by vertical lines. Note that definition 6.3 supports these common hyper-points shared by neighboring wall features; but is not limited to them. In a special case shown in Figure 6.4, the corner points of grid edge AB have the same roof layer assignment in one adjacent cell (right cell) but different assignments in another (left cell). Thus, a vertical boundary polygon is produced to reflect this significant topology feature. Specifically, R_1 and R_2 denote the same roof patch, which is adjacent with W along b_1 and b_2 representing the upper roof boundary polyline and the bottom roof boundary polyline respectively. Since b_1 and b_2 are non-identical, W is a valid wall feature. Point feature F acts as a "folding point" which connects b_1 and b_2 together and folds up the boundary of W.

6.2.2 Connections between Topological Features

By projecting the 2.5D building model onto the x-y plane, the building topology can be viewed with a 2D cell complex representation[2]. In particular, roof features, wall features, and point features are projected onto the 2D x-y plane as 2-cells (regions), 1-cells (polylines), and 0-cells (points) respectively. High dimensional cells are always bounded by a set of low dimensional cells.

Given a projection operator $\mathbb{P}(\cdot)$ which projects a set of 2.5D objects onto the x-y plane and a boundary extraction operator $\partial(\cdot)$, the connections between roof feature set $\mathcal{R}$, wall feature set $\mathcal{W}$, and point feature set $\mathcal{P}$ are revealed with following equations:

$$\mathbb{P}(\partial R) \subseteq \mathbb{P}(\mathcal{W}), \qquad \text{for any } R \in \mathcal{R}, \tag{6.1}$$

$$\partial\mathbb{P}(W) \subseteq \mathbb{P}(\mathcal{P}), \qquad \text{for any } W \in \mathcal{W}. \tag{6.2}$$

[2]Cell complexes are the basic concepts in algebraic topology, see [23] for detailed discussion.

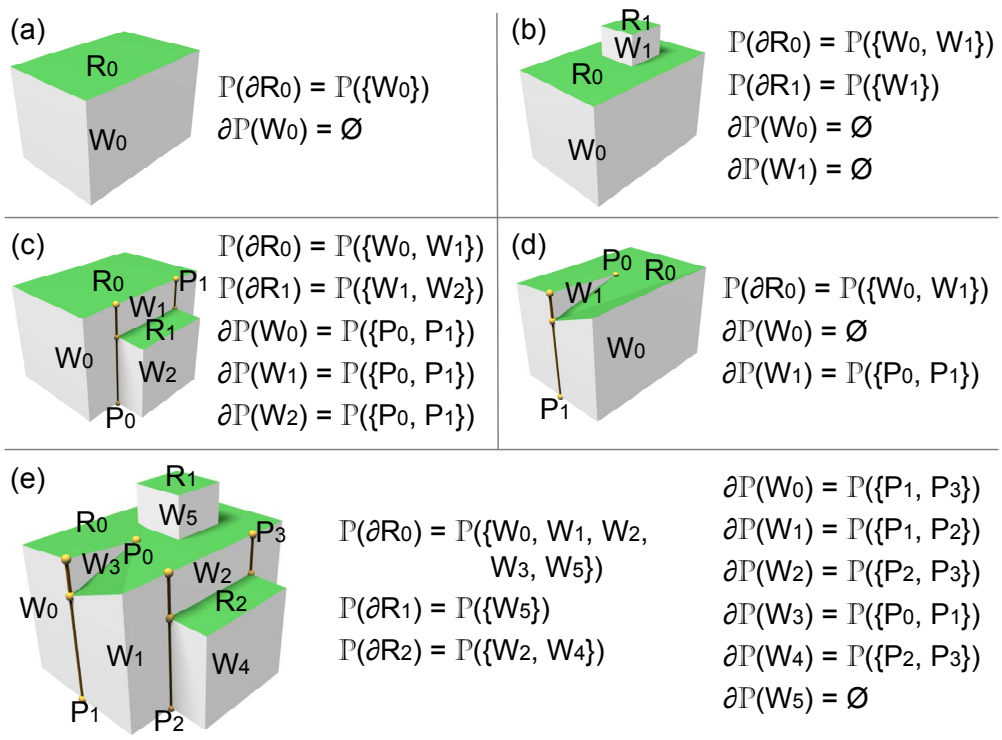


Figure 6.5: 2.5D building topology is represented by topological features, and the associations between them in form of Equation (6.1) and (6.2). Examples include typical building structures such as (a) individual building blocks, (b) blocks with top attachments, (c) blocks with side attachments, (d) stair-shaped structures, and (e) combinations of these patterns.

These equations can be straightforwardly derived from the definitions of roof, wall, and point features. On the other hand, once topological features and their associations expressed in form of Equation (6.1) and (6.2) are fixed, the 2.5D building topology is determined accordingly.

Figure 6.5 demonstrates typical building structures including standing-alone building blocks, vertically attached blocks, horizontally attached blocks, stair shapes, and the combinations of these patterns. Nevertheless, the 2.5D building topology representation describes them in a deterministic and differentiable manner.

In addition, the difference between 2.5D topology representation and classic 2D topology representation is noted as follows. The latter one can be achieved by projecting all building elements onto the x-y plane at first, and treating different roof layers as multiple region materials. This representation, however, is problematic in handling wall features connecting the gap within one roof layer (*e.g.*, W_1 in Figure 6.5(d)). It eliminates such wall features together with the folding point (*e.g.*, P_0). In contrast, the topology representation presented in this section faithfully preserves all significant 2.5D topology features which are the basis of the topology control method [3].

6.3 Contouring with Topology Control

So far the 2.5D topological features are formally defined and the associations between them are well explored. These mechanisms can be naturally expanded from a uniform grid to a quadtree. Thus, the quadtree-based simplification of 2.5D dual contouring can be extended with a topology control method to maintain the 2.5D building topology.

6.3.1 Hyper-Points Clustering

The core of the simplification algorithm is to optimize the geometry of hyper-points based on a 2.5D quadratic error function (see Section 5.4). These hyper-points can be categorized by the number of their layers.

1. **1-layer points**: A hyper-point containing one vertex is optimized targeting the disk-like geometry in a grid cell. In most cases, it is connected to vertices created in its neighboring cells only by surface polygons. Such a hyper-point is an inner

[3]This problem can also be regarded as the result of changing the order in applying boundary extraction operator and projection operator. As 2D topology representation projects all elements onto the x-y plane at first, it attempts to replace $\mathbb{P}(\partial R)$ in Equation (6.1) with $\partial\mathbb{P}(R)$. This attempt is problematic because $\partial\mathbb{P}(R) \not\equiv \mathbb{P}(\partial R)$ as $\mathbb{P}(\cdot)$ can absorb wall features such as W_1 in Figure 6.5(d).

vertex of a roof feature. Thus it can be safely merged into a neighboring vertex without changing the 2.5D building topology. The accepting vertex can be either another 1-layer point or one vertex in a hyper-point with more than one layers. The only exception is a 1-layer folding point, which is connected to two different layers of a neighboring hyper-point by a boundary polygon, as shown in Figure 6.4 and Figure 6.6 left. In this case the 1-layer point is a point feature, thus should not be merged into other points.

2. **2-layer points**: A hyper-point containing two layers is a typical roof boundary anchor point, which is optimized with surface geometry and boundary geometry simultaneously. Typically, it is an inner element of a wall feature, thus can be merged into another hyper-point that is connected to it by a vertical boundary polygon. The accepting hyper-point can be either another typical 2-layer point or a multi-layer point. Similar to 1-layer points, folding points can exist within 2-layer points. Figure 6.6 middle shows such an example where the 2-layer point is a point feature. It cannot be merged in typical manner.

3. **Multi-layer points**: A hyper-point with more than two layers can be regarded as a conjunction point of more than two regions, if the problem is viewed on the 2D projection plane where roof patches are treated as regions with different materials. I find that any multi-layer point is a point feature. Proof is straightforward as they cannot be the inner elements of wall features; thus always stand at boundaries of wall features' 2D projections. Therefore, multi-layer points can only accept 1-layer points and 2-layer points joining it, but cannot be merged with other point features.

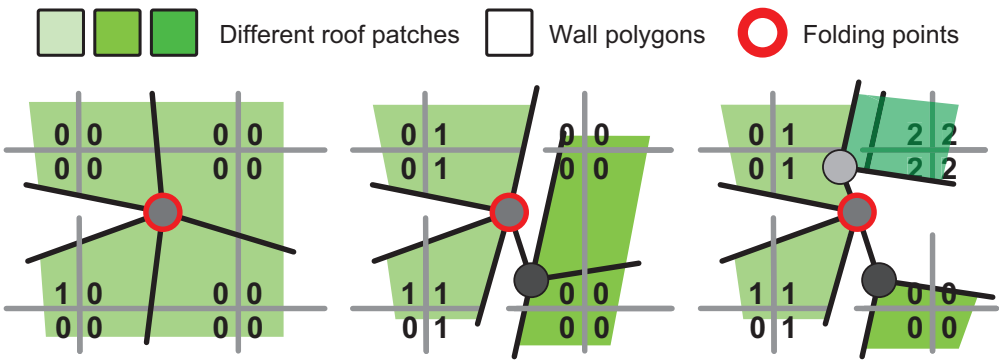


Figure 6.6: Folding points can be part of a 1-layer hyper-point, a 2-layer hyper-point, or a multi-layer hyper-point (from left to right). They are detected and marked as point features before adaptive simplification starts.

Since folding points can exist in 1-layer points, 2-layer points, and multi-layer points, as demonstrated in Figure 6.6, they are detected and marked as point features during pre-processing. The detection algorithm is implemented by uncovering grid edges with inconsistent roof layer assignments in adjacent cells.

Apart from folding points, typical 1-layer points, typical 2-layer points, and multi-layer points are denoted as p^1, p^2, and p^m respectively. A set of hyper-point clustering operations are introduced to merge components connected by surface polygons, denoted as $\Phi_{\mathrm{S}}^{1\to1}$, $\Phi_{\mathrm{S}}^{1\to2}$, and $\Phi_{\mathrm{S}}^{1\to m}$, with following functions:

$$\Phi_{\mathrm{S}}^{1\to1}: \quad \{p_1^1, p_2^1, \ldots, p_n^1\} \quad \Rightarrow p^{1*}, \tag{6.3}$$

$$\Phi_{\mathrm{S}}^{1\to2}: \quad \{p_1^1, p_2^1, \ldots, p_n^1\}, p^2 \quad \Rightarrow p^{2*}, \tag{6.4}$$

$$\Phi_{\mathrm{S}}^{1\to m}: \quad \{p_1^1, p_2^1, \ldots, p_n^1\}, p^m \quad \Rightarrow p^{m*}. \tag{6.5}$$

Each operation merges a connected component within a roof feature, and produces one hyper-point (*e.g.*, p^{1*}, p^{2*}, or p^{m*}) per cluster. In particular, the geometric coordinates of output hyper-points are obtained by optimizing a 2.5D QEF matrix which is the combination of QEF matrices from input hyper-points. The third column from p_i^1's matrices

are placed with corresponding matrix column from p^2 or p^m. Details of QEF matrices combination can refer to Section 5.4.2. Similarly, the clustering operations for components that are connected by vertical boundary polygons are:

$$\Phi_{\mathrm{B}}^{2\to 2}: \quad \{p_1^2, p_2^2, \ldots, p_n^2\} \quad \Rightarrow p^{2*}, \tag{6.6}$$

$$\Phi_{\mathrm{B}}^{2\to m}: \quad \{p_1^2, p_2^2, \ldots, p_n^2\}, p^m \quad \Rightarrow p^{m*}. \tag{6.7}$$

These operations merge connected components within a wall feature.

6.3.2 Handling Degenerate Cases

Although these clustering operations aim at simplifying continuous roof and boundary features, they risk in making topological features degenerate. For example, with intense geometry simplification, the building structure in Figure 6.5(a) may degenerate into a single vertical line with its top vertex collapsed from R_0.

To address this problem, degenerate tests are adopted after each clustering operation. Given $f(C)$ and $e(C)$ as the number of faces and edges in a cell complex C, the following requirements must be satisfied:

$$f(R) \geq 1, \qquad \text{for any } R \in \mathcal{R}, \tag{6.8}$$

$$e(\mathbb{P}(W)) \geq 1, \qquad \text{for any } W \in \mathcal{W}. \tag{6.9}$$

Initially, these two criteria are fulfilled due to Definition 6.1 and 6.2. In adaptive simplification phase, for each possible hyper-point clustering operation, a degenerate test is applied to all the modified roof features and wall features, to check if these two criteria are still satisfied. If any of them is violated, the clustering operation is rewound.

In practice, I found this degenerate test inefficient because it involves polygon recreation and topological feature detection after every clustering operation. Thus an equivalent criterion is proposed to replace the former two, which is much easier to be implemented:

Degenerate test A hyper-point clustering operation passes the degenerate test if the following criterion stays true:

$$e(\partial\mathbb{P}(R)) \geq 3, \qquad \text{for any } R \in \mathcal{R}. \tag{6.10}$$

The equivalence between this degenerate test and the one based on Equation (6.8) and (6.9) is proved in the Appendix of this chapter.

Since the degenerate test is irrelevant to typical 1-layer points, clustering operations based on surface components are always allowed (*i.e.*, $\Phi_{\mathrm{S}}^{1\to1}$, $\Phi_{\mathrm{S}}^{1\to2}$, $\Phi_{\mathrm{S}}^{1\to m}$). For clustering among 2-layer points and multi-layer points, the simplification algorithm precomputes boundaries of roof feature projections, *i.e.*, $\partial\mathbb{P}(R)$, and keeps tracking of the edge numbers $e(\partial\mathbb{P}(R))$ during the complete simplification process. Boundary component clustering operations (*i.e.*, $\Phi_{\mathrm{B}}^{2\to2}$ and $\Phi_{\mathrm{B}}^{2\to2}$) decrease the corresponding edge numbers. Once a boundary edge number is less than 3, the latest operation is rewound.

6.3.3 Adaptive Contouring

2.5D dual contouring is extended by allowing multiple hyper-points in each grid cell of the quadtree Q. In addition to the adaptive structure of quadtree, a hyper-point forest is maintained to allow topology-preserving clustering operations which are detailed in previous sections. As illustrated in Figure 6.7, trees in the forest are connected components that can be simplified via a series of clustering operations, and each of them is finally

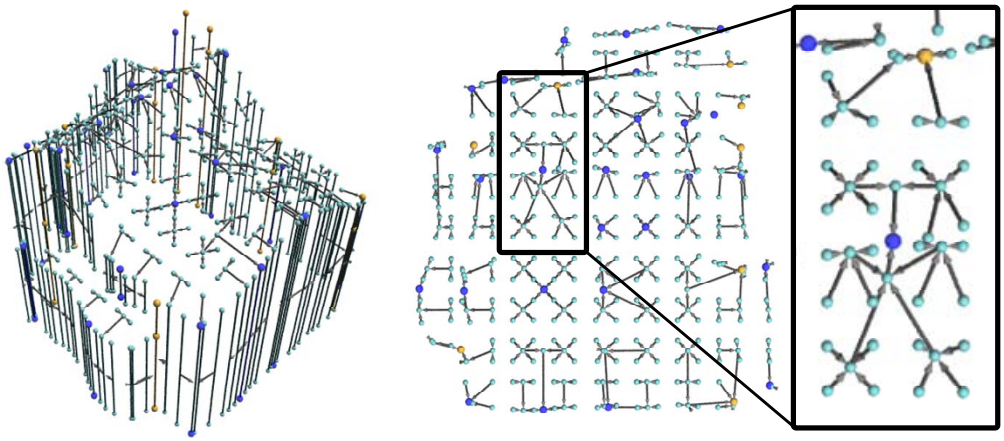


Figure 6.7: Hyper-point cluster forest viewed from oblique and orthogonal perspectives

represented by its root (blue and gold hyper-points) whose coordinates are determined by optimizing a 2.5D QEF combining geometric information from leaf points.

This hyper-point cluster forest is built in a bottom-up manner. In a quadtree cell c composed of four leaf cells $c_{0,0}, c_{0,1}, c_{1,0}, c_{1,1}$, assume each leaf cell has a set of cluster roots that are available for further clustering (*i.e.*, without exceeding the geometry error tolerance or violating degenerate test). The simplification algorithm first traverses all the corner points in c that are shared by two of the four leaf cells. At each grid corner, four vertices in adjacent cells are connected via a surface polygon. The simplification algorithm retrieves the roots of these vertices and detects possible clustering operations based on surface component, *i.e.*, $\Phi_{\mathrm{S}}^{1\to1}$, $\Phi_{\mathrm{S}}^{1\to2}$, or $\Phi_{\mathrm{S}}^{1\to m}$. Similar approaches can be applied to minimal grid edges which exhibit roof layer gaps. They imply boundary-neighborships leading to possible operations $\Phi_{\mathrm{B}}^{2\to2}$ and $\Phi_{\mathrm{B}}^{2\to m}$.

With possible clustering operation detected, they are sequentially tested against the geometry error tolerance and the degenerate test. Since the test sequence may affect the modeling quality, each clustering operation is assigned with a priority. In particular, high priority is assigned to $\Phi_{\mathrm{S}}^{1\to1}$ and $\Phi_{\mathrm{B}}^{2\to2}$; medium priority is assigned to $\Phi_{\mathrm{S}}^{1\to2}$ and $\Phi_{\mathrm{S}}^{1\to m}$; and low priority is assigned to $\Phi_{\mathrm{B}}^{2\to m}$. The reason behind this priority assignment

is that 1-hyper points and 2-hyper points are expected to be first clustered together to form meaningful geometric patterns (*e.g.*, roof ridges and straight vertical walls), before they are merged into key features with higher dimensional topology. As for clustering operations with same priority, the test sequence is determined by the addition to quadratic errors in ascending order.

Polygon generation of 2.5D dual contouring is adapted to the hyper-point cluster forest in a straightforward manner. Considering the unsimplified polygonal model created from the uniform grid, each point in the model is replaced by the root of its cluster (or the corresponding portion of the root if that has more layers than the leaf point). Numerous polygons become degenerate and are removed automatically, *e.g.*, a triangle whose three vertices belong to the same cluster and thus map to the same root point. A simple polygonal model with small amount of triangles is produced which has the same 2.5D building topology as the unsimplified model.

6.4 Experimental Results

Figure 6.8 shows a stadium model reconstructed using different approaches, namely, 2.5D dual contouring (a,b), the modeling method presented in this chapter (c,d), manual creation (e), and the plane-based method presented in Section 3.4. In particular, the geometry error tolerance δ is varied in order to make trade-offs between model scale and fitting quality. The relation curve between δ and the number of triangles produced by different approaches is illustrated in Figure 6.8(g). Quantitative measurements are given in Table 6.1. With error tolerance δ increasing, the new method constantly decreases the triangle number of reconstructed models. Reasonable cost is paid in fitting quality as the trade-off. On the contrary, 2.5D dual contouring reaches the simplification barrier around 3,000 triangles. This barrier can be explained by the last column of Table 6.1,

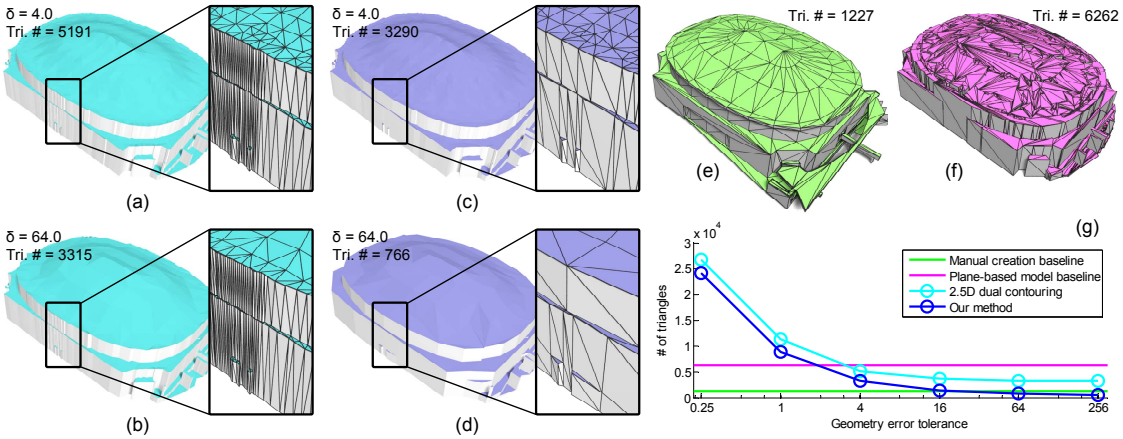


Figure 6.8: A stadium model created using (a,b) 2.5D dual contouring, (c,d) the modeling method presented in this chapter, (e) manual creation, and (f) the plane-based approach in Section 3.4. The relation curve between error tolerance and the triangle number of reconstructed models is illustrated in (g). With larger geometry error tolerance given, the new method can always produce simpler models with less triangles; while the overstrict topology test in 2.5D dual contouring creates numerous trivial triangles along thin roof features shown in closeups of (a,b).

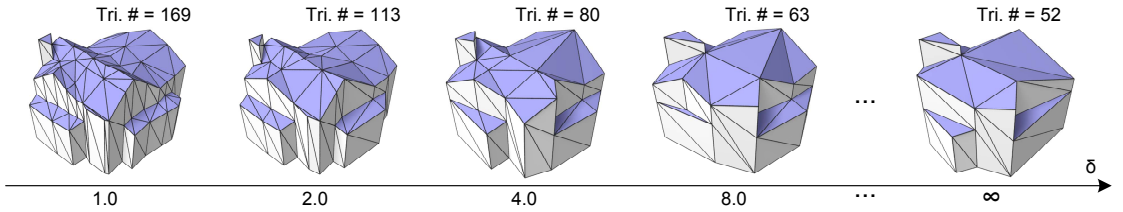


Figure 6.9: Model evolution with error tolerance growing from 1.0 to infinite

showing the percentage of unsuccessful collapses caused by topology test among all unsuccessful collapses. Many of them happen in small cells that create trivial triangles as shown in Figure 6.8(a,b) closeups.

Figure 6.10 shows the building reconstruction for a 5km-by-7km urban area of Denver, from 73M input aerial LiDAR points with 6 samples/sq.m. resolution. The urban modeling pipeline in Section 4 is adopted to extract individual building patches, and

test the modeling method with topology control, 2.5D dual contouring, and a plane-based method independently. A fairly large error tolerance is utilized for both the new method and 2.5D dual contouring. The new modeling method successfully reconstructs 2,099 2.5D building models within 5 minutes on a consumer level laptop (Intel i-7 CPU 1.60GHz with 6GB memory). 227,566 triangles are produced for the building models which are rendered in the top rows of Figure 6.10. The output triangle number is comparable to plane-based results (181,752 triangles rendered in the bottom row). However, unlike the plane-based method, the modeling approach presented in this chapter detects and preserves 2.5D building topology, thus avoids producing cracks and inconsistencies between building blocks. *E.g.*, the roof of the large structure shown in the bottom left closeup intersects with small features on top of it; while the new method does not have such problem. The middle row of Figure 6.10 shows 2.5D dual contouring result, it produces twice as many triangles (551,341 triangles) as the other two approaches.

Since the scale of the result is inversely proportional to geometry error tolerance δ. It is beneficial to study the evolution of building model with respect to δ. In particular, 2.5D building models are created for the same LiDAR point cloud using an exponentially increasing δ, shown in Figure 6.9. Although the model geometry constantly becomes simpler, the building topology is faithfully preserved. Even in the extreme simplification case where $\delta = \infty$, a model with 32 vertices and 52 triangles is created, which contains the smallest amount of vertices and triangles that can represent the building topology of this model.

6.5 Appendix

Proof of equivalence between (6.8)+(6.9) and (6.10):

First, all three equations stand true in the initial unsimplified model.

Geometry error tolerance	The new method		2.5D dual contouring		
	Triangle #	Avg. distance^2	Triangle #	Avg. distance^2	Topology test failure rate
$\delta = 0.25$	24161	0.0157	26776	0.0156	3.02% (82 out of 2713)
$\delta = 1.00$	8864	0.0202	11280	0.0182	11.83% (109 out of 921)
$\delta = 4.00$	3290	0.0849	5191	0.0316	39.10% (149 out of 381)
$\delta = 16.00$	1374	0.1259	3644	0.0988	76.30% (161 out of 211)
$\delta = 64.00$	766	0.4812	3315	0.2104	90.45% (161 out of 178)

Table 6.1: Quantitative comparison between modeling with topology control and 2.5D dual contouring using the experiment shown in Figure 6.8. The last column reports the percentage of cell collapses rejected by topology test among all rejected collapses. The topology test becomes dominant in 2.5D dual contouring with large error tolerance.

After a clustering operation, if both equation (6.8) and (6.9) are true, for each $R \in \mathcal{R}$, there is $f(\mathbb{P}(R)) = f(R) \geq 1$. ***I.e.***, $\mathbb{P}(R)$ contains at least one 2-cell. Thus, the boundary of $\mathbb{P}(R)$ contains at least 3 edges, $e(\partial\mathbb{P}(R)) \geq 3$.

Conversely, when Equation (6.10) is true, $\mathbb{P}(R)$ has at least one 2-cell. Therefore, $f(R) = f(\mathbb{P}(R)) \geq 1$, ***i.e.***, (6.8). As for a wall feature W with $|\partial\mathbb{P}(W)| \geq 2$, it is bounded by two point features, and (6.9) is true by definitions of hyper-point clustering operations. Now considering a wall feature W with $|\partial\mathbb{P}(W)| \leq 1$ (***e.g.***, W_0 in Figure 6.5(d) and W_1 in Figure 6.5(b)), $\mathbb{P}(W)$ is a close loop on 2D space. It divides $\mathcal{R}$ into partition $\mathcal{R}_{in}$ and $\mathcal{R}_{out}$, where $\mathcal{R}_{in} \neq \emptyset$. Therefore:

$$\mathbb{P}(W) = \partial \sum_{R \in \mathcal{R}_{in}} \mathbb{P}(R). \tag{6.11}$$

Since the right part sums at least one 2-cell before boundary extraction, the boundary of the cell complex contains at least 3 edges, ***i.e.***, $e(\mathbb{P}(W)) \geq 3$. □

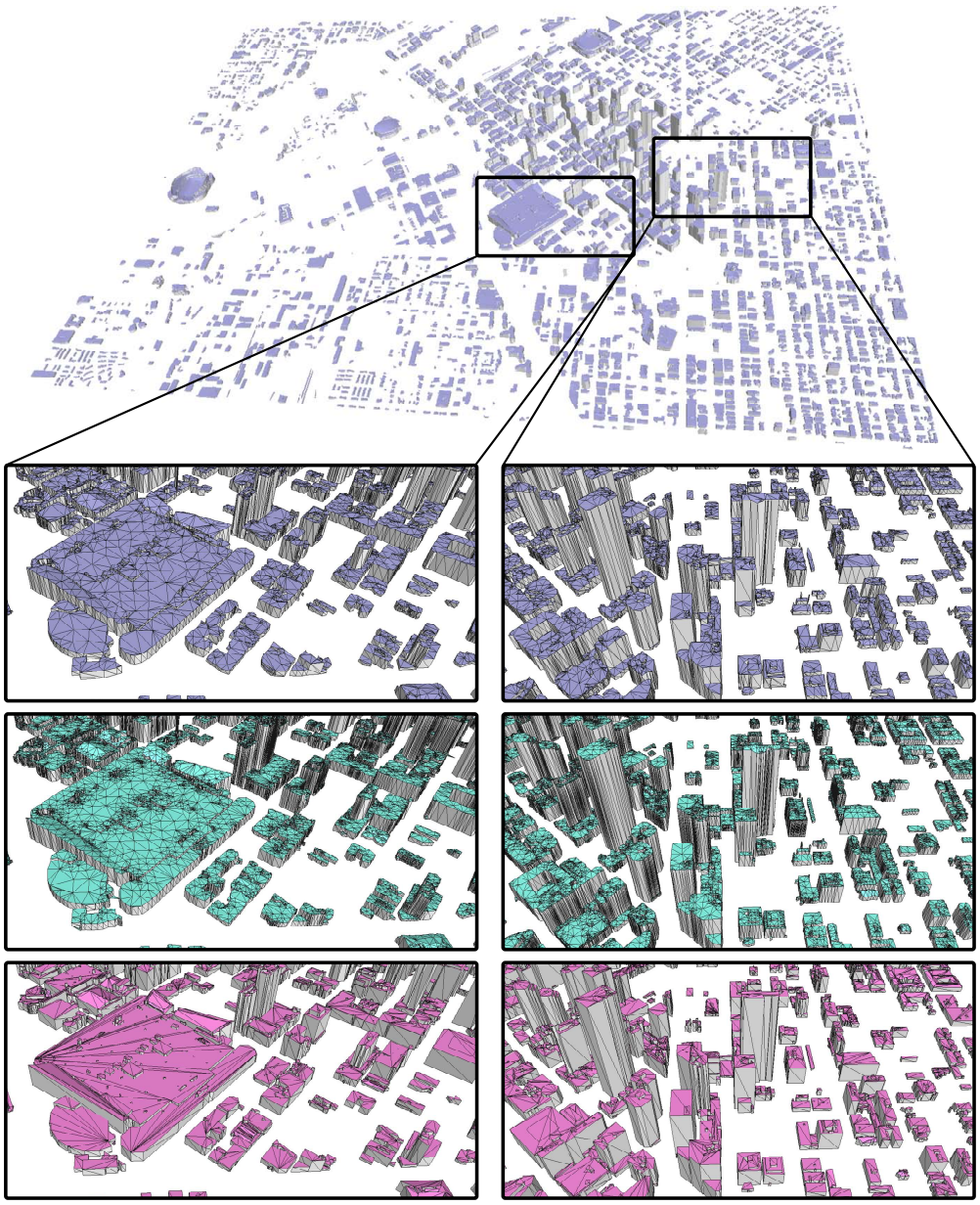


Figure 6.10: 2,099 building models are created for an urban area in Denver using (top) building modeling with topology control, (middle) 2.5D dual contouring, and (bottom) plane-based method. The new method produces as few triangles as the plane-based method while recovering and preserving the topological features in each building structure.

Chapter 7

2.5D Building Modeling by Discovering Global Regularities

Besides data-driven building modeling approaches such as those proposed in Chapter 5 and Chapter 6, another popular strategy in attacking the 2.5D building modeling problem is to introduce primitives (*e.g.*, planes, spheres, cones) to represent building shapes. In particular, planes receive the most attention since they are the commonest structures in man-made objects, especially in buildings. Planar roof patches are locally fitted from raw points, and are later combined with vertical facades aligning with roof boundaries, to construct a compact mesh model while maintaining low geometric fitting error rate. The main difficulty of this strategy is that local plane fitting can become unstable when dealing with noisy or incomplete point clouds. Artifacts are inevitably created from unreliable plane primitives. To alleviate this problem, existing methods typically introduce strong urban priors to prune the fitted planes, such as roof topology [60], Manhattan-world grammars [41, 48], and principal directions introduced in Section 3.4.3 and Section 4.3.4. While prior knowledge successfully increases the robustness of these methods, it tends to be overstrict and thus limits their applicability when dealing with moderately complex building structures such as the one shown in Figure 7.1.

This chapter proposes *global regularities*, a moderate yet informative structure to organize planar roof patches and roof boundary segments. As illustrated in Figure 7.1 bottom, I explore both orientation and placement regularities between planar roof patch

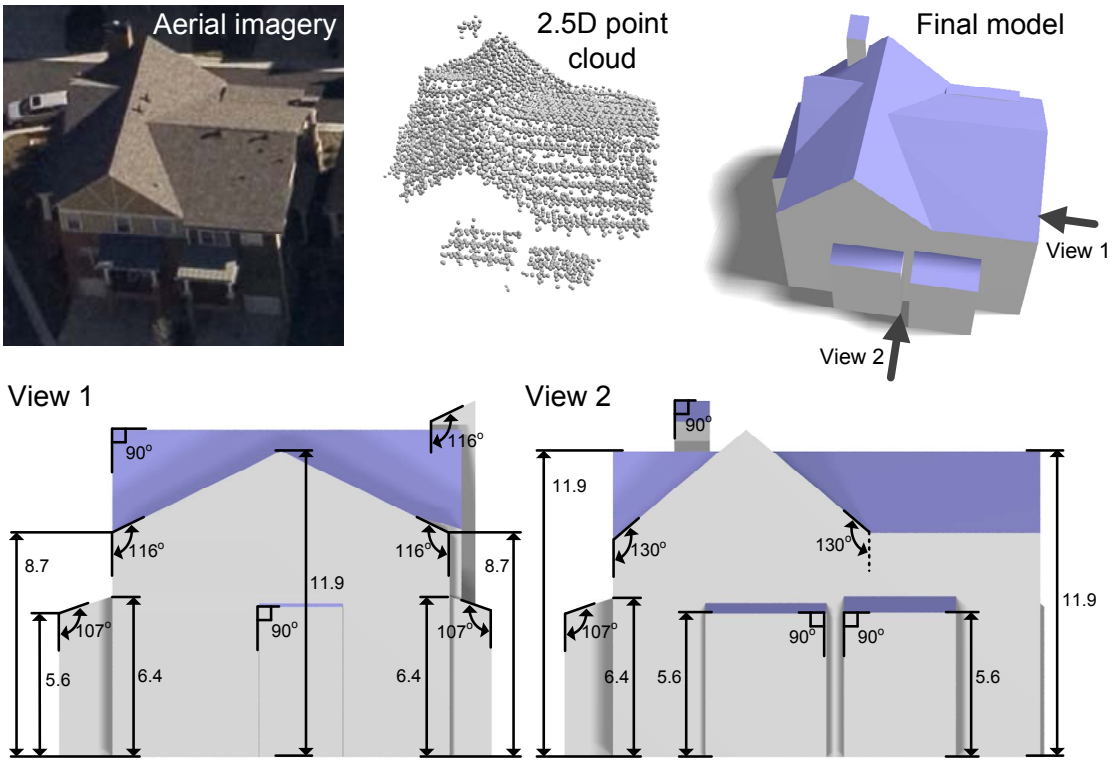


Figure 7.1: The method presented in this chapter automatically discovers global regularities from a noisy 2.5D point cloud, and uses them to create a convincing 2.5D building model. Two orthogonal projections illustrate a subset of the global regularities in this model (lengths in meters).

pairs (*e.g.*, slope angle equality), between roof boundary segment pairs (*e.g.*, segment height equality), and between a planar roof patch and its boundary segments (*e.g.*, orthogonality between their orientations). These global regularity patterns reveal the inter-element similarities and relations that intrinsically exist in building models because of human design and construction. With these patterns, the complexity of the building modeling problem can be significantly reduced for complicated building models such as the one in Figure 7.1.

This chapter presents an automatic algorithm which detects global regularities and utilizes them to calibrate plane primitives. Unlike the strong priors introduced by previous methods, global regularities offer a more flexible representation of the global knowledge in 2.5D building models, and thus enable the modeling algorithm to handle more complicated building shapes.

Another significant advantage of global regularities is that they characterize the intrinsic structures of building models, to which human vision is sensitive. For instance, Figure 7.2 right shows two models created from plane primitives. Without comparing model geometry with input points, human vision immediately finds the top-right model more convincing since it conforms to more global regularities.

In this chapter, Section 7.1 summarizes shape-from-symmetry approaches. Section 7.2 explores various global regularity patterns in 2.5D building structures. Section 7.3 presents an automatic algorithm to discover and enforce global regularities through a series of alignment steps. Finally, Section 7.4 shows 2.5D modeling results with high quality in terms of both geometry and human judgement.

7.1 Review of Shape-from-Symmetry Approaches

In both computer vision and computer graphics, symmetry has been identified as reliable global knowledge in 3D reconstruction. For instance, Fisher [16] introduces domain knowledge of standard shapes and relationships into reverse engineering problems. Thrun and Wegbreit [57] detect symmetries and utilize them to extend partial 3D models into occluded space. Gal *et al.* [19] adopt 1D wires to carry both local geometry information and global mutual relationships in man-made objects. Li *et al.* [36] extract relationship graphs among primitives to encode intra-part relations and use them to further improve the reconstruction quality.

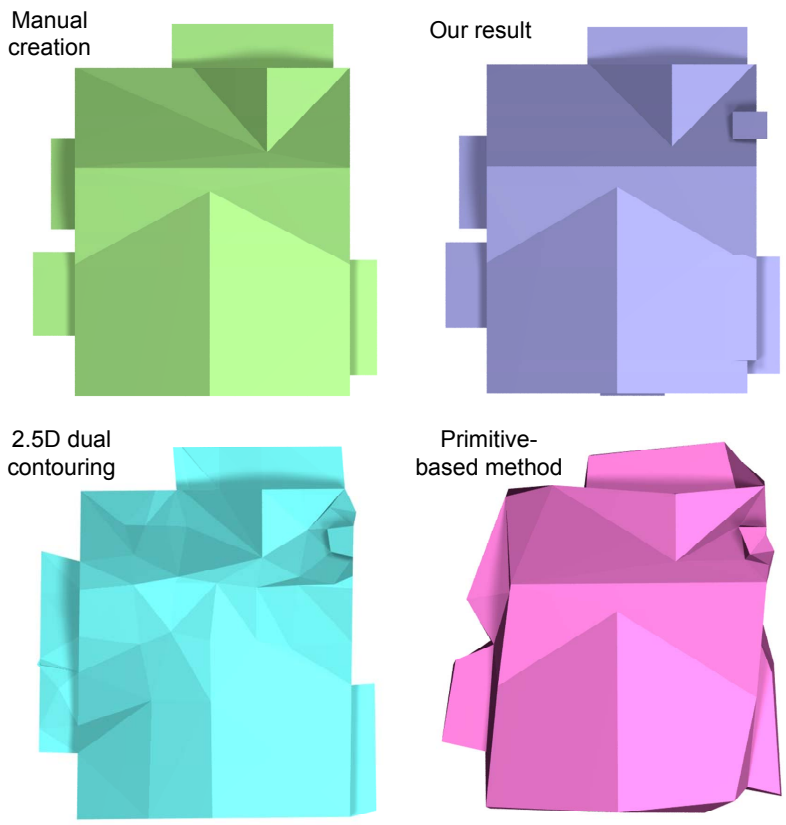


Figure 7.2: Modeling results generated from the same input point cloud by manual creation, the modeling method proposed in this chapter, 2.5D dual contouring with principal direction snapping, and a primitive-based method [34]. The new modeling method creates the most visually convincing result among all three automatic methods since its resulting building model conforms to the most global regularities.

These methods are similar in spirit to the method presented in this chapter. While previous research work focuses on 3D man-made objects, this research is the first to explore global regularities in 2.5D building models composed of roof patches and vertical walls.

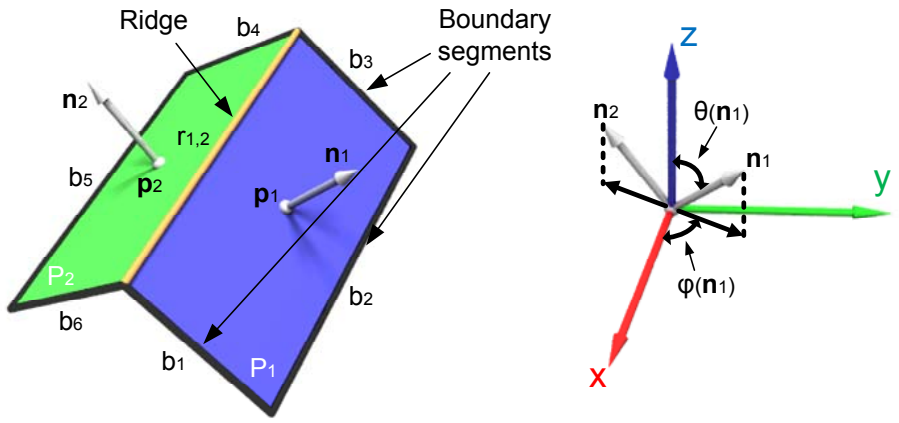


Figure 7.3: A typical gable-shaped building roof containing a set of 2.5D elements (*e.g.*,plane primitive, boundary segments, and ridges)

7.2 Global Regularities

In 2.5D building models, global regularities characterize the inter-element similarities and relations arising from human design and construction. They are particularly useful in correcting plane primitives and creating more visually convincing building models. This section explores various global regularity patterns that commonly exist in 2.5D building models, and demonstrates them using a typical gable-shaped building roof shown in Figure 7.3.

Considering a 2.5D building model composed of plane primitives including planar roof patches and planar facade patches, it can be equivalently represented by a set of planar roof patches together with their boundary segments; because given the 2.5D constraints that roof surfaces are always bounded by vertical facades, planar facade patches and linear roof boundary segments have the same projection on the x-y plane. In particular, the planar roof patch set is denoted as $\mathcal{P} = \{P_i : (\mathbf{v} - \mathbf{p}_i) \cdot \mathbf{n}_i = 0\}$ in which each plane P_i is determined by a normal-position pair $(\mathbf{n}_i, \mathbf{p}_i)$. A boundary segment set for each planar roof patch is collected by intersecting P_i with its surrounding planar facade patches, denoted as $\mathcal{B}_i$ (*e.g.*, in Figure 7.3, $\mathcal{B}_1 = \{b_1, b_2, b_3\}$). The following sections

explore global regularities among the 2.5D element set $\mathcal{P} \cup (\cup_i \mathcal{B}_i)$ from three aspects: roof-roof regularities, roof-boundary regularities, and boundary-boundary regularities.

7.2.1 Roof-Roof Regularities

This section focuses on two classes of commonly encountered regularities between roof plane pair (P_i, P_j) as follows.

7.2.1.1 Orientation regularities

In 3D models, the orientation regularities are usually defined as the orthogonality or parallelism between plane normals (*e.g.*, [36]). This definition, however, cannot be directly applied to 2.5D building models for two reasons: first, roof plane normals rarely show orthogonality or parallelism; second, roof inclination and direction are of more interest in building modeling. In 2.5D models, orientation regularities are not determined by the angle between plane normals, but by the projections of normals on either the x-y plane or the z-axis. For instance, although $\mathbf{n}_1$ and $\mathbf{n}_2$ in Figure 7.3 do not exhibit orthogonality or parallelism, they show strong orientation regularities since their projections on the x-y plane are opposite. Therefore, I choose to write plane normals in spherical coordinates $(\theta(\mathbf{n}), \varphi(\mathbf{n}))$:

$$\theta(\mathbf{n}) = \arccos(n_z), \tag{7.1}$$

$$\varphi(\mathbf{n}) = \arctan(n_y, n_x), \tag{7.2}$$

where $\theta(\mathbf{n}) \in [0, \pi/2)$ and $\varphi(\mathbf{n}) \in [0, 2\pi)$ (Figure 7.3 right).

Intuitively, $\theta(\mathbf{n})$ determines the inclination of the planar roof patch, and $\varphi(\mathbf{n})$ indicates the direction of the slope. As humans are particularly interested in roof patches having the same inclination and roof patches exhibiting regularized slope directions

(either orthogonal or parallel), four typical roof-roof orientation regularities are defined accordingly:

- θ**-equality** when $\theta(\mathbf{n}_i) = \theta(\mathbf{n}_j)$,
- φ**-equality** when $\varphi(\mathbf{n}_i) = \varphi(\mathbf{n}_j)$,
- φ**-opposite** when $\varphi(\mathbf{n}_i) = \varphi(\mathbf{n}_j) \pm \pi$,
- φ**-orthogonality** when $\varphi(\mathbf{n}_i) = \varphi(\mathbf{n}_j) \pm \frac{\pi}{2}, \frac{3\pi}{2}$.

For example, plane pair (P_1, P_2) in Figure 7.3 exhibits the same inclination and the opposite slope direction. Using the above formulation, these characteristics are denoted as θ-equality and φ-opposite respectively.

7.2.1.2 Placement regularities

Placement of roof planes (*i.e.*, roof positions) by themselves do not contain much regularity information. However, the placement of intersections between roof plane pairs may carry meaningful structural information about the building. In particular, *ridges* are defined to reveal the regularities of roof plane placements.

Ridge definition: for a neighboring plane pair (P_i, P_j) satisfying both θ-equality and φ-opposite, the intersection of P_i and P_j is defined as a *ridge*, denoted as $r_{i,j}$.

The direction of ridge $r_{i,j}$ is uniquely determined as

$$\mathbf{d}(r_{i,j}) = (\sin(\varphi(\mathbf{n}_i)), -\cos(\varphi(\mathbf{n}_i)), 0)^T. \tag{7.3}$$

Since $\mathbf{d}(r_{i,j})$ is parallel to the x-y plane, the placement of $r_{i,j}$ can be parameterized by a pair of real numbers $(h(r_{i,j}), p(r_{i,j}))$, denoting the height of $r_{i,j}$ and the distance from origin to $r_{i,j}$'s projection on the x-y plane respectively. They can be calculated by

solving an equation system with plane equations regarding $(\mathbf{n}_i, \mathbf{p}_i)$ and $(\mathbf{n}_j, \mathbf{p}_j)$. Two types of placement regularities for ridge pair $(r_{i,j}, r_{k,l})$ are defined as:

- **Ridge-height-equality** when $h(r_{i,j}) = h(r_{k,l})$,
- **Ridge-position-equality** when $\mathbf{d}(r_{i,j}) \parallel \mathbf{d}(r_{k,l})$ and $p(r_{i,j}) = p(r_{k,l})$.

7.2.2 Roof-Boundary Regularities

I observe that the majority of boundary segments are aligned either orthogonally (*e.g.*, b_2 in Figure 7.3) or parallel (*e.g.*, b_1, b_3) to the normals of their owner planes (*e.g.*, P_1), when projected on the x-y plane. Therefore, this section focuses on roof-boundary regularities between plane P_i and its boundary segments $\mathcal{B}_i$. The direction of P_i on the x-y plane is denoted by a 2D vector $\mathbf{o}_i = (\cos(\varphi(\mathbf{n}_i)), \sin(\varphi(\mathbf{n}_i)))^T$, and the direction of a boundary segment b_j's x-y projection is denoted as $\mathbf{o}(b_j) = \mathbb{P}(\mathbf{d}(b_j)), b_j \in \mathcal{B}_i$, where $\mathbb{P}$ is the projection operator and $\mathbf{d}(b_j)$ is the b_i's direction in 3D space. There are:

- **o-parallelism** when $\mathbf{o}_i \parallel \mathbf{o}(b_j)$,
- **o-orthogonality** when $\mathbf{o}_i \perp \mathbf{o}(b_j)$,

7.2.3 Boundary-Boundary Regularities

As the orientation regularities among boundary segments can be implied from roof-boundary regularities and roof-roof orientation regularities, this section focuses on placement regularities between boundary segment pairs. In particular, two significant regularity patterns are noted: first, boundary segments that are parallel to the x-y plane may have similar heights (*e.g.*, (b_2, b_5) in Figure 7.3); second, when projected onto the x-y plane, boundary segments with the same direction may align to the same line (*e.g.*, (b_1, b_6) and (b_3, b_4)). Thus, boundary-boundary regularities are defined as follows,

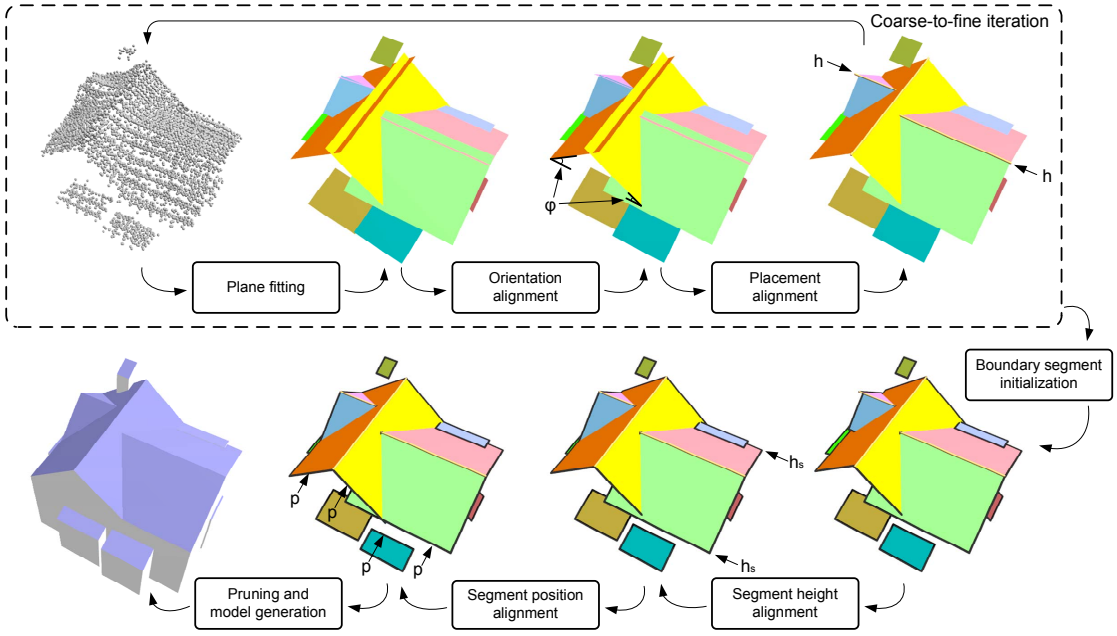


Figure 7.4: Pipeline of the modeling approach: a 2.5D point cloud (top left) is transformed to a building model (bottom left) through a series of steps. Global regularities are discovered and enforced in each alignment step.

where $h(b_i)$ is the height of boundary segment b_i, and $p(b_i)$ is the distance from origin to b_i's projection on the x-y plane:

- **Segment-height-equality** when $h(b_i) = h(b_j)$, and both $\mathbf{d}(b_i)$ and $\mathbf{d}(b_j)$ are parallel to the x-y plane,
- **Segment-position-equality** when $\mathbf{o}(b_i) \parallel \mathbf{o}(b_j)$ and $p(b_i) = p(b_j)$.

7.3 Modeling with Global Regularities

Given a noisy 2.5D point cloud as input, this section presents an automatic method to simultaneously detect locally fitted plane primitives and global regularities. In general, a discover-then-align strategy is adopted: once initial plane primitives are identified, the modeling algorithm discovers global regularities from them, and then immediately

refines these initial primitives by aligning them to the global regularities. This optimization strategy is applied individually to each type of global regularities defined in Section 7.2. It effectively corrects the geometric errors raised by local fitting approaches, and thus significantly improves the model quality.

An overview of this approach is shown in Figure 7.4. The building modeling system contains three main modules to create a 2.5D building model (bottom left) from a noisy aerial scan (top left):

1. **Planar roof patch extraction:** As shown in Figure 7.4 top, with plane primitives detected via local fitting, two discover-and-align steps are sequentially executed to detect the roof-roof regularities and refine the planar roof patch, namely, *orientation alignment* and *placement alignment*. Both planar roof patches and the roof-roof regularities are iteratively generated in a coarse-to-fine manner.

2. **Boundary segment production:** The modeling algorithm immediately enforces the roof-boundary regularities by creating a rectangular bounding box for each planar roof patch, and identifying boundary segments from bounding box edges, shown as the black lines in Figure 7.4 bottom. These boundary segments are further refined by discovering and enforcing boundary-boundary regularities.

3. **Model generation:** Vertical facades are automatically generated from boundary segments to connect roof patches and the ground. Rectangular roof patches are pruned by neighboring elements. A 2.5D building model is produced by combining both planar roof patches and vertical facades as shown in Figure 7.4 bottom left.

7.3.1 Planar Roof Patch Extraction

Given a set of points equipped with normals[1], a popular plane detection algorithm is adopted for aerial LiDAR scans [34, 60] to find plane primitives: a region-growing procedure is applied to find spatially connected point clusters with similar normals; then plane primitives are locally fitted to individual point clusters. The detected plane primitive set is denoted as $\mathcal{P} = \{P_i\}$. Orientation alignment and placement alignment are applied to $\mathcal{P} = \{P_i\}$ sequentially.

7.3.1.1 Orientation alignment

By expressing plane normals in spherical coordinates, the orientation regularities can be categorized into two classes: θ-equality finds planes with similar slope angles, while φ-equality, φ-opposite and φ-orthogonality show regularized roof patch directions. These orientation regularities can be discovered by detecting clusters of $\Theta = \{\theta(\mathbf{n}_i)\}$ and clusters of $\Phi = \{\varphi(\mathbf{n}_i) \mod (\pi/2)\}$ respectively. Each angle cluster implies a set of corresponding orientation regularities while the center of each cluster predicts the best alignment. In particular, the complete-linkage clustering algorithm [12] is adopted to identify clusters in Θ and Φ. Cluster center sets $\mathcal{C}_\Theta$ and $\mathcal{C}_\Phi$ are taken as constraints in the subsequent alignment stage, in which $\theta(\mathbf{n}_i)$ and $\varphi(\mathbf{n}_i)$ are snapped to the corresponding cluster centers in $\mathcal{C}_\Theta$ and $\mathcal{C}_\Phi$.[2]

[1] Normals can be effectively estimated via covariance analysis [47].

[2] In singular cases where $\theta(\mathbf{n}_i) \approx 0$, $\varphi(\mathbf{n}_i)$ becomes unstable. Thus, $\theta(\mathbf{n}_i)$ is snapped to 0 and $\varphi(\mathbf{n}_i)$ is assigned to the most popular $\varphi \in \mathcal{C}_\Phi$.

7.3.1.2 Placement alignment

To effectively deal with placement regularities, ridges are detected from neighboring plane pairs. Similar to orientation alignment, the placement alignment algorithm decouples the placement alignment into two independent sub-problems: aligning ridge heights towards ridge-height-equality and aligning ridge positions towards ridge-position-equality. These placement regularities can be discovered by finding clusters of ridge height set $\mathcal{H} = \{h(r_{i,j})\}$; and clusters of ridge position set $\mathcal{S}(\mathbf{d}) = \{p(r_{i,j})|\mathbf{d}(r_{i,j}) \parallel \mathbf{d}\}$, regarding each ridge direction $\mathbf{d}$. Cluster center sets are denoted as $\mathcal{C}_{\mathcal{H}}$ and $\mathcal{C}_{\mathcal{S}}$ respectively, and used as regularity constraints henceforth. In the alignment stage, ridge height $h(r_{i,j})$ and position $p(r_{i,j})$ are both aligned to their cluster centers, resulting in modifications on plane position $\mathbf{p}_i$ and $\mathbf{p}_j$. In order to avoid conflicts between ridge height alignment and ridge position alignment, the former only affects the z values of position vectors, while the latter makes modifications to the x and y coordinates. Therefore, the only conflict source lies in planes that have multiple ridges, where the ridges compete in modifying the plane's position. In this case, only the longest ridge is allowed to modify the position. The effects from other ridges are ignored.

7.3.1.3 Coarse-to-fine iteration

The planar roof patch extraction executes in a coarse-to-fine manner, as demonstrated in Figure 7.5. In particular, the modeling system fits planes to the input points, makes orientation alignment and placement alignment to the plane primitives, discards points already associated with existing plane primitives, and then iterates through these steps with three modifications until no more plane primitives can be found by planing fitting:

1. Plane-fitting criterion is loosened to accept smaller plane patches. In particular, normal variance allowance α is increased and the minimum number of points

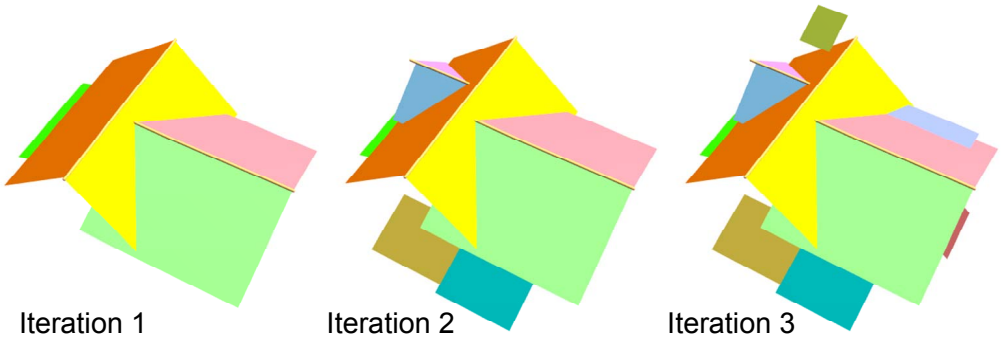


Figure 7.5: Coarse-to-fine planar roof patch extraction

required to validate a plane primitive K is reduced. Empirically, α remains $pi/12$ while K is reduced by half in each iteration, from 200 to 25.

2. In plane-fitting, instead of randomly picking region-growing seed, the plane detection algorithm starts from points with normals that are close to normal $\mathbf{n}(\theta, \varphi), \forall(\theta, \varphi) \in \mathcal{C}_\Theta \times \mathcal{C}_\Phi$. These points have great potential to grow into plane primitives conforming to existing orientation regularity constraints.

3. In alignment steps, the new plane primitives first attempt to snap to existing constraint sets $\{\mathcal{C}_\Theta, \mathcal{C}_\Phi, \mathcal{C}_\mathcal{H}, \mathcal{C}_\mathcal{S}\}$. Clustering is performed only for primitives that cannot align to the existing constraints given a distance threshold. New cluster centers are combined with existing centers to form regularity constraints for the next iteration.

These modifications are driven by two observations: first, large plane primitives are more reliable in producing global regularity constraints, and thus the iterative algorithm begins with large primitives and requires small primitive to be snapped to large primitives if possible (modification 3); second, the regularity constraints detected from large primitives can greatly improve the robustness of plane fitting for small patches (modification 2).

7.3.2 Boundary segment production

Given a set of plane primitives as shown in Figure 7.4 top right, initial boundary segments are created for each planar roof patch with help of roof-boundary regularities. Segment height alignment and segment position alignment are performed based on boundary-boundary regularities.

7.3.2.1 Boundary segment initialization

As discussed in Section 7.2.2, most boundary segments in 2.5D building models conform to the roof-boundary regularities, *i.e.*, when projected on the x-y plane, boundary segments are either orthogonal or parallel to the normals of their owner planes. On the other hand, boundary segments represent the borders of roof patches, and thus bound the patch content, *i.e.*, points associated with the roof plane. Considering a planar roof patch P_i and the set of points associated with it denoted as $\mathcal{V}_i$; a boundary extraction algorithm computes a rectangular bounding box R_i of point projections $\mathbb{P}(\mathcal{V}_i)$ on the x-y plane, with R_i's orientation following $\mathbf{o}_i = (\cos(\varphi(\mathbf{n}_i)), \sin(\varphi(\mathbf{n}_i)))^T$. The segment set $\mathcal{B}_i$ is initialized by back projecting R_i's edges onto P_i.

Given that the plane normals are already aligned to a small set of orientation constraints $\mathcal{C}_\Phi$, the x-y directions of boundary segments fall into a few 2D directions, which can be regarded as building-scale *principal directions* (as discussed in Section 3.4.3). In the special case where $|\mathcal{C}_\Phi| = 1$, the boundary segments are in two orthogonal x-y directions, forming *rectilinear contours* for building models [41].

7.3.2.2 Segment height alignment

The modeling system now applies segment height alignment to horizontal boundary segments. Segment-height-equality is discovered and enforced by a clustering method similar to previous alignment steps with two additional rules:

1. Heights of boundary segments from the same plane patch cannot belong to the same cluster, since snapping them together risks making the roof patch degenerate.

2. For each plane pair (P_i, P_j) that creates a ridge $r_{i,j}$, the boundary segments opposite to $r_{i,j}$ (*e.g.*, b_2 and b_5 in Figure 7.3) are tested with a relaxed criteria, because each ridge indicates a high probability of a reflection-symmetry.

A boundary segment is marked as "fixed" if its height is snapped to a cluster with more than one element. The position of each fixed segment is determined accordingly by substituting the modified height value into the plane equation. These positions act as placement constraints $\mathcal{C}_p$ in the next step.

7.3.2.3 Segment position alignment

Segment position alignment is applied to the remaining boundary segments including both non-horizontal segments and horizontal segments that are not marked as "fixed" in the previous step. Positions of these segments first attempt to snap to elements in $\mathcal{C}_p$, and only join the position clustering procedure when the snapping attempt fails.

7.3.3 Model Generation

With planar roof patches and their boundary segments generated and aligned to the global regularities, a 2.5D building model is reconstructed by combining roof patches and vertical facades, as shown in Figure 7.4 bottom left. The facades are produced from boundary segments connecting roof patches to the ground, while rectangular roof patches are pruned by neighboring elements including both roof patches and facades.

Metric	This method	2.5D dual contouring	Primitive-based method [34]	Manual creation
Triangle #	130	214	89	78
Avg. dis^2	0.012	0.016	0.018	0.058
Outlier ratio	0.9‰	10.0‰	11.1‰	17.3‰

Table 7.1: Quantitative results for the comparison in Figure 7.6

7.4 Experimental Results

This section first compares the modeling method proposed in this chapter with two existing approaches (*i.e.*, 2.5D dual contouring and a primitive-based method [34]). Figure 7.2 shows a qualitative comparison between these methods. The modeling result with global regularities has the most similar visual appearance to manual creation, because it conforms to the most global regularities that characterize the intrinsic structure of building models. In contrast, 2.5D dual contouring only considers boundary direction similarities by introducing the *principal direction snapping* algorithm, while [34] detects primitives but does not deal with the relations between them. In addition, quantitative comparison between the three methods are made using metrics shown in Figure 7.6 and Table 7.1. While the modeling result with global regularities shows comparable triangle number and average squared distance compared with previous approaches, it significantly reduces the outlier ratio (*i.e.*, the ratio of points with squared distances greater than 0.25m^2), because the modeling method can robustly fits small plane primitives through iterations with global regularities detected and updated progressively.

The modeling method with global regularities is further tested on several LiDAR scans of buildings in the city of Atlanta, as illustrated in Figure 7.7. The input contains aerial LiDAR point cloud with 17 samples/m^2 resolution (first column). Planar roof patches and their boundary segments are detected and aligned with global regularities

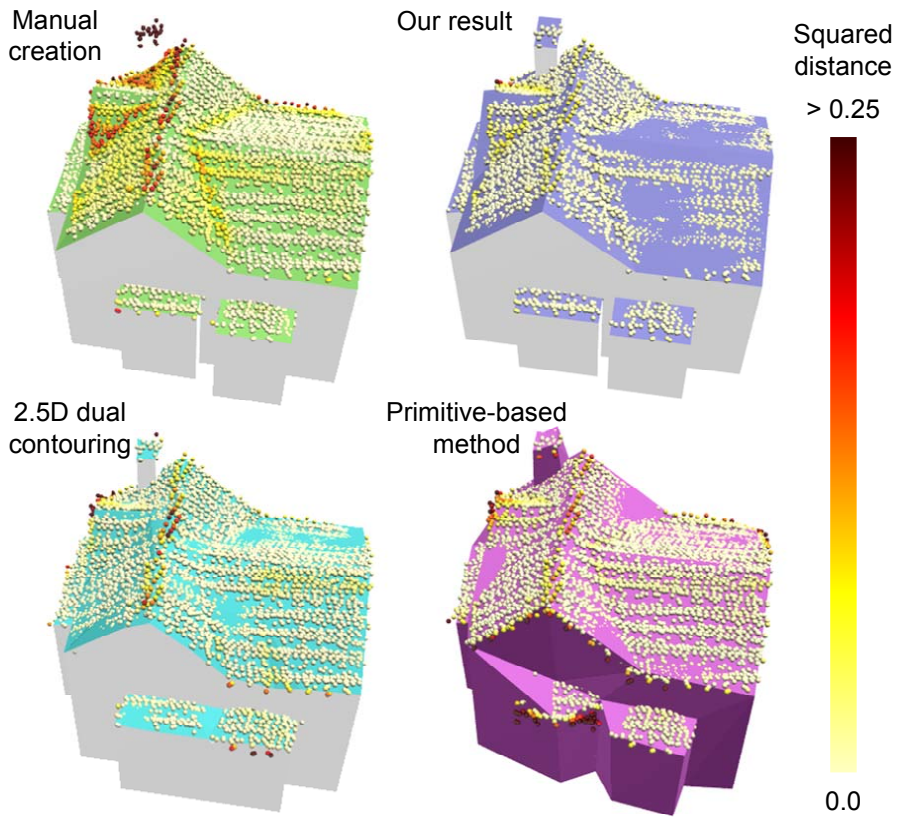


Figure 7.6: A comparison of geometric fitting errors (*i.e.*, squared distances from input points to the model surfaces) between models created by four different approaches

learnt from them (second column). 2.5D building models are reconstructed from these primitives by pruning the roof patches (see examples in the last three rows) and creating vertical facades from boundary segments (third column). Given the aerial imagery as reference (fourth column), the new results are more "realistic" than the 2.5D dual contouring results (last column). Table 7.2 shows the statistics of the experiments in Figure 7.7. Computation time is measured on a laptop with Intel i-7 CPU 1.60GHz and 6GB memory.

Model	Point #	**Plane #**	$\lvert\mathcal{C}_\Theta\rvert$	$\lvert\mathcal{C}_\Phi\rvert$	**Ridge #**	$\lvert\mathcal{C}_\mathcal{H}\rvert$	$\lvert\mathcal{C}_\mathcal{P}\rvert$	**Hor. segment #**	Seg. h-cluster #	**Non-hor. segment #**	Seg. p-cluster #	Tri. #	time (s)
1st row	3349	**6**	1	1	**3**	3	2	**6**	1	**12**	5	48	7.1
2nd row	5603	**6**	2	1	**3**	3	2	**6**	2	**12**	6	48	6.8
3rd row	4549	**4**	1	2	**2**	1	2	**4**	1	**8**	4	28	3.6
4th row	6143	**13**	3	1	**5**	3	3	**18**	7	**24**	13	110	16.6
5th row	6920	**14**	3	1	**7**	3	3	**16**	2	**28**	10	112	39.6

Table 7.2: Statistics of the experiments in Figure 7.7. Column 3 - 12 show statistics of the intermediate results. Since the size of constraint sets is considerably smaller than the number of primitives (in bold), the solution space (and thus the complexity) of the modeling problem is reduced significantly.

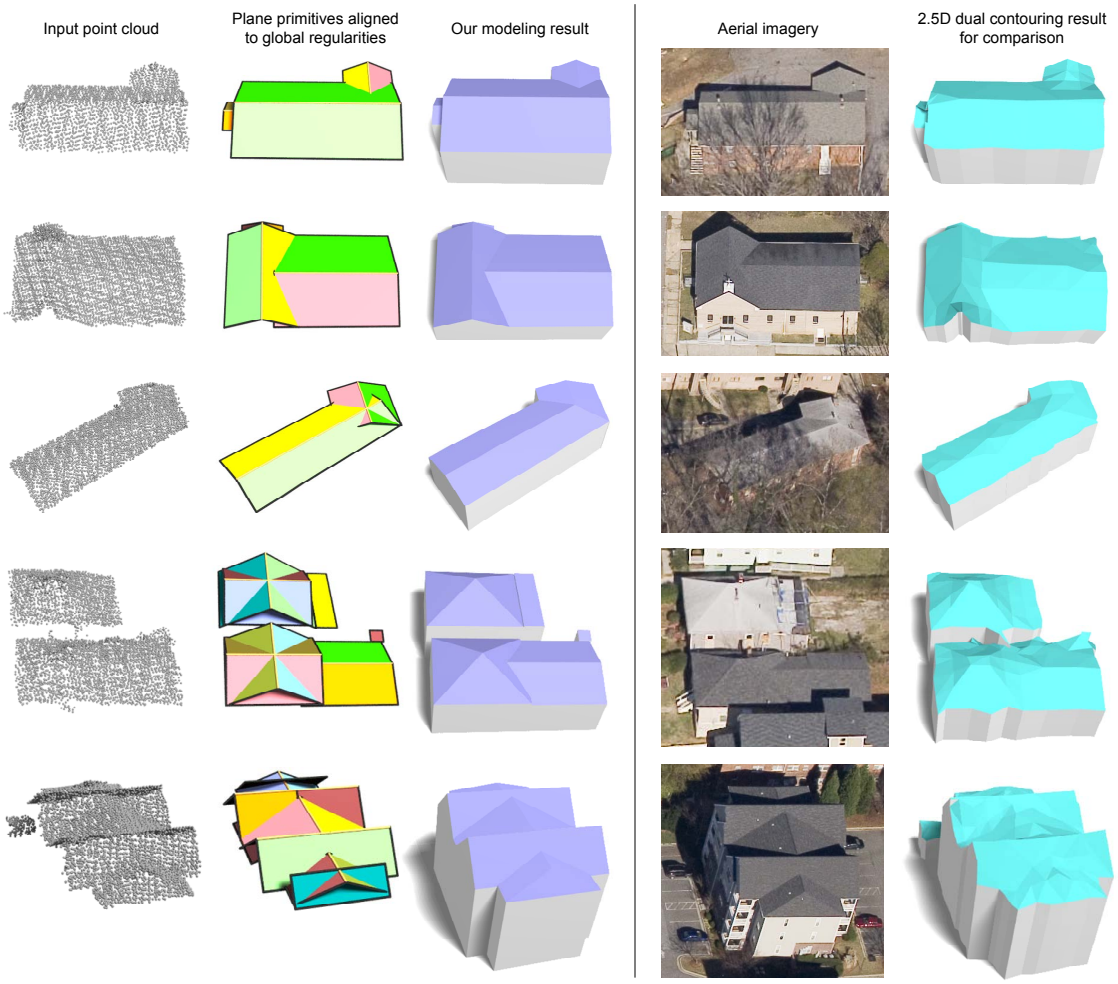
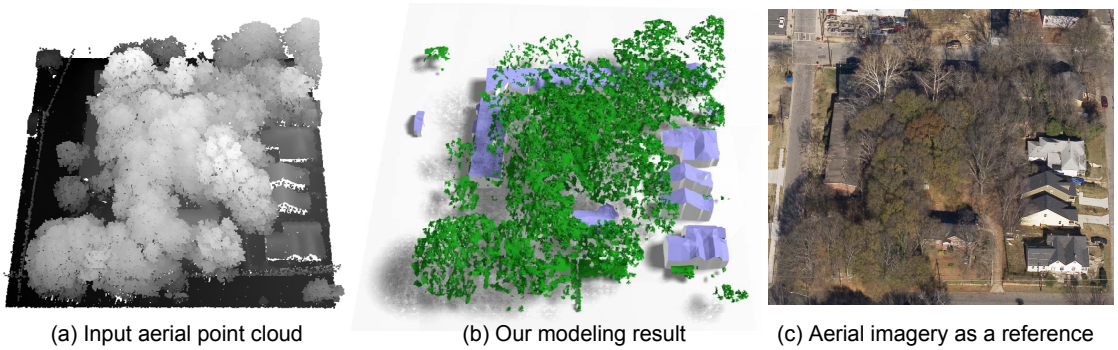


Figure 7.7: Experiments on several building scans. By discovering global regularities and enforcing them on the planar roof patches and their boundary segments (second column), the modeling algorithm creates visually convincing 2.5D building modelings (third column). Aerial imagery and 2.5D dual contouring results are included as reference.

Chapter 8

Modeling Residential Urban Areas



(a) Input aerial point cloud (b) Our modeling result (c) Aerial imagery as a reference

Figure 8.1: Given (a) a dense aerial LiDAR scan of a residential area (point intensities represent heights), the residential urban modeling system reconstructs (b) 3D geometry for buildings and trees respectively. (c) Aerial imagery is shown as a reference.

Two new challenges emerge when the urban modeling problem extends to residential areas. First, as shown in Figure 8.1(a), vegetation is a major component of urban reality in residential areas. An urban modeling method for residential areas should detect and reconstruct both buildings and trees, *e.g.*, as in Figure 8.1(b). The second challenge lies in the classification method: dense LiDAR scans capture the detailed geometry of tree crowns, which may have similar height and local geometry features as rooftops of residential buildings. Figure 8.2 shows such an example where part of the tree crown shows similar or even better *planarity* than part of the rooftop (see closeups illustrating local points as spheres together with the optimal plane fitted to them). Classification algorithms based on local geometry features may fail and produce significant modeling errors. *E.g.*, Figure 8.2 right.

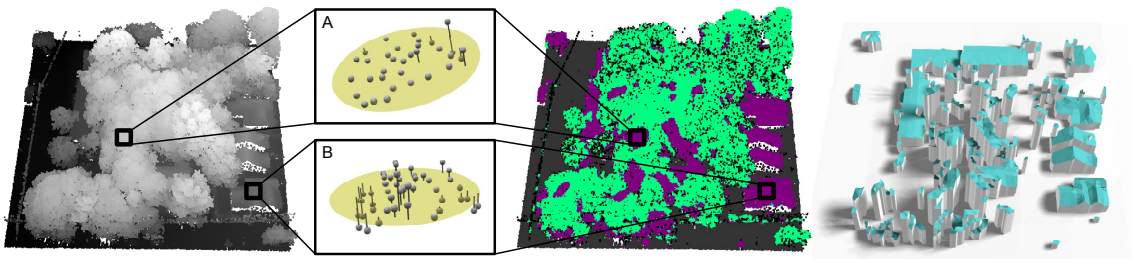


Figure 8.2: Local geometry features become unreliable when dealing with residential areas with rich vegetation. In closeups of (A) a tree crown region and (B) a rooftop, points are rendered as spheres while a locally fitted plane is rendered in yellow. Middle: classification results from the general urban modeling system in Chapter 3, trees in green, buildings in purple, and ground in dark grey. Right: modeling artifacts are created because of classification errors.

To address these two challenges, this chapter presents a robust classification method to classify input points into trees, buildings, and ground. Building models and trees are created from these points using 2.5D dual contouring and a novel leaf-based tree modeling approach, respectively. The heart of the classification method is a simple, intuitive, but extremely effective measurement. In particular, I observe that residential buildings usually show a strong 2.5D characteristic, *i.e.*, they are composed of skywards roofs and vertical walls; both are opaque and thus prevent the laser beams from penetrating the building structure. Therefore, there is no point sample inside the building structure. The rooftops (or ground) become the lowest visible surface at a certain x-y position, as illustrated in Figure 8.3 left. In contrast, trees, composed of branches and leaves, do not have this 2.5D structure. With multiple passes of scanning from different angles, the point cloud captures not only the top surface of the tree crown, but also surfaces inside and underneath the crown, as shown in Figure 8.3 right.

In this chapter, Section 8.1 reviews tree detection and tree modeling approaches respectively. Section 8.2 proposes an effective algorithm to classify trees, building roofs, and ground. In Section 8.3, a hybrid model containing both 2.5D building models and

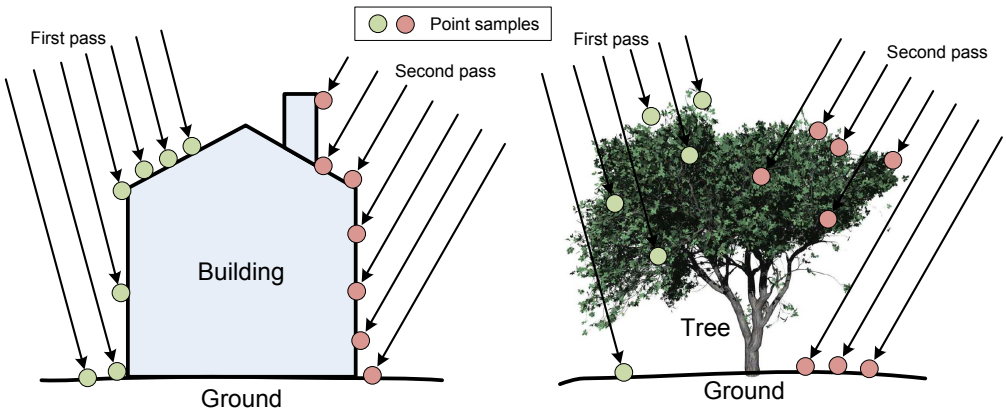


Figure 8.3: While building structures have a 2.5D characteristic, trees do not possess such property. Dense laser scans may capture surface points under the tree crown (right).

leaf-based tree models is generated in an automatic and robust manner. Experimental results are shown in Section 8.4.

8.1 Related Work

8.1.1 Tree Detection in LiDAR

In urban modeling systems, trees are often recognized as outliers and thus are classified and removed in the first step. Most of the classification algorithms rely on point-wise features including height [34, 39, 52, 58] and its variation [5, 39, 52], intensity [39, 52], and local geometry information such as *planarity* [34, 60], *scatter* [34, 58], and other local geometry features. Heuristics or machine learning algorithms are introduced as classifiers based on the defined feature set. To further identify individual building roof patches, segmentation is either introduced in a post-classification step, or combined with classification in the form of energy minimization such as [34].

Nevertheless, the method proposed in this chapter is the first to introduce the 2.5D characteristic of building structures into the classification problem. A simple, efficient

and effective classification algorithm is presented, which gains great accuracy in residential areas with rich vegetation.

8.1.2 Tree Modeling

Tree modeling is a missing part in most of the aerial LiDAR based urban modeling approaches. To the best of my knowledge, Lafarge and Mallet [34] is the only research work which addresses the tree modeling problem by matching simple ellipsoidal template to tree clusters. This method, however, is problematic when dealing with complicated tree structures in residential areas, *e.g.*, Figure 8.1(a).

Computer graphics and remote sensing communities have made great efforts in modeling trees from ground LiDAR and imagery, such as [9, 38, 44, 55, 56, 63]. A general tree model is broadly adopted in these literatures, composed of skeletal branches and leaves attached to them. Inspired by these efforts, this chapter proposes leaf-based tree modeling from aerial LiDAR scans.

8.2 Point Cloud Classification

Given an aerial LiDAR point cloud of a residential area as input, the objective of classification is to classify points into three categories: trees, buildings, and ground. As mentioned previously and illustrated in Figure 8.3, the 2.5D characteristic is the key difference between trees and buildings (or ground). In order to formulate this concept, the point cloud is first embedded into a uniform 2D grid G. In each grid cell c, the point set $P(c)$ is segmented into multiple *layer fragments* $L(c)$, using local distance-based region growing. Ideally, a layer fragment $l_{building} \in L(c)$ lying on a 2.5D object (rooftop or ground) must have the lowest height among all layer fragments in $L(c)$, because the

rooftop (or ground) is always the lowest visible surface to laser beams at a certain x-y position, as analyzed previously. On the other hand, a tree layer fragment l_{tree} can exhibit any height. However, as there is usually a ground or rooftop surface underneath tree samples, l_{tree} is not expected to be the lowest layer fragment in $L(c)$. From an energy minimization perspective, these features can be described by a data energy term $E_d(x_l)$ for each $l \in L(c)$ as:

$$E_d(x_l) = \begin{cases} \alpha & \text{if } x_l = \textit{building} \text{ or } \textit{ground}, \text{ and } l \text{ is not the lowest in } L(c) \\ \beta & \text{if } x_l = \textit{tree}, \text{ and } l \text{ is the lowest layer fragment in } L(c) \\ 0 & \text{otherwise} \end{cases} \tag{8.1}$$

where x_l is the label of layer fragment l.

To further discriminate building and ground in the energy minimization framework, *elevation* of layer fragment $e(l)$ is introduced and defined as the height difference between l and the ground elevation at c's center [34, 58][1]. Another data energy term $E_g(x_l)$ is defined accordingly:

$$E_g(x_l) = \begin{cases} \gamma \cdot max(1 - \frac{e(l)}{\sigma}, 0) & \text{if } x_l = \textit{building} \\ \gamma \cdot min(\frac{e(l)}{\sigma}, 1) & \text{if } x_l = \textit{ground} \\ 0 & \text{if } x_l = \textit{tree} \end{cases} \tag{8.2}$$

where σ is the normalization factor. Empirically, $\sigma = 6m$, as suggested in [34].

[1]The ground elevation map can be easily estimated by assigning a 20m-by-20m coarse grid, estimating ground height with the lowest point in each cell, and applying linear interpolation across the entire coarse grid.

With a smooth energy $E_s(x_{l_1}, x_{l_2})$ defined over all neighboring layer fragment pairs (*i.e.*, layer fragments belonging to neighboring cells and satisfying certain distance criteria), a Markov Random Field can be built, which leads to an energy minimization problem over the labeling x of the entire layer fragment set $\mathcal{L}$:

$$E(x) = \sum_{l \in \mathcal{L}} (E_d(x_l) + E_g(x_l)) + \lambda \sum_{(l_1, l_2) \in \mathcal{N}} E_s(x_{l_1}, x_{l_2}) \tag{8.3}$$

where $\mathcal{N}$ is the set of neighboring layer fragment pairs, and smooth energy $E_s(x_{l_1}, x_{l_2})$ is defined as characteristic function $\mathbf{1}_{x_{l_1} \neq x_{l_2}}$.

With the energy minimization problem being solved using the well-known graph-cut method [3], point labels are determined as the label of the corresponding layer fragment. To further construct roof patches from building points, a region growing algorithm is applied based on certain distance criteria. While large building patches are adopted as rooftops, small patches are considered as outliers and removed henceforth.

Figure 8.4 demonstrates the entire process of point cloud classification. Input points are first discretized into layer fragments. For illustration purpose, Figure 8.4(b) shows only the lowest layer fragment in each cell. They faithfully capture the skyward surfaces of 2.5D structures. By solving an energy minimization problem, building and ground layer fragments are detected and shown in Figure 8.4(c). Because of the smooth energy term, eaves are segmented as part of the building roof, and small clusters of tree layer fragments with low heights are correctly detected, as illustrated in the closeups respectively. Finally, the classification result is applied to the point cloud and a region growing algorithm successfully groups roof patches from building points, *i.e.*, Figure 8.4(d).

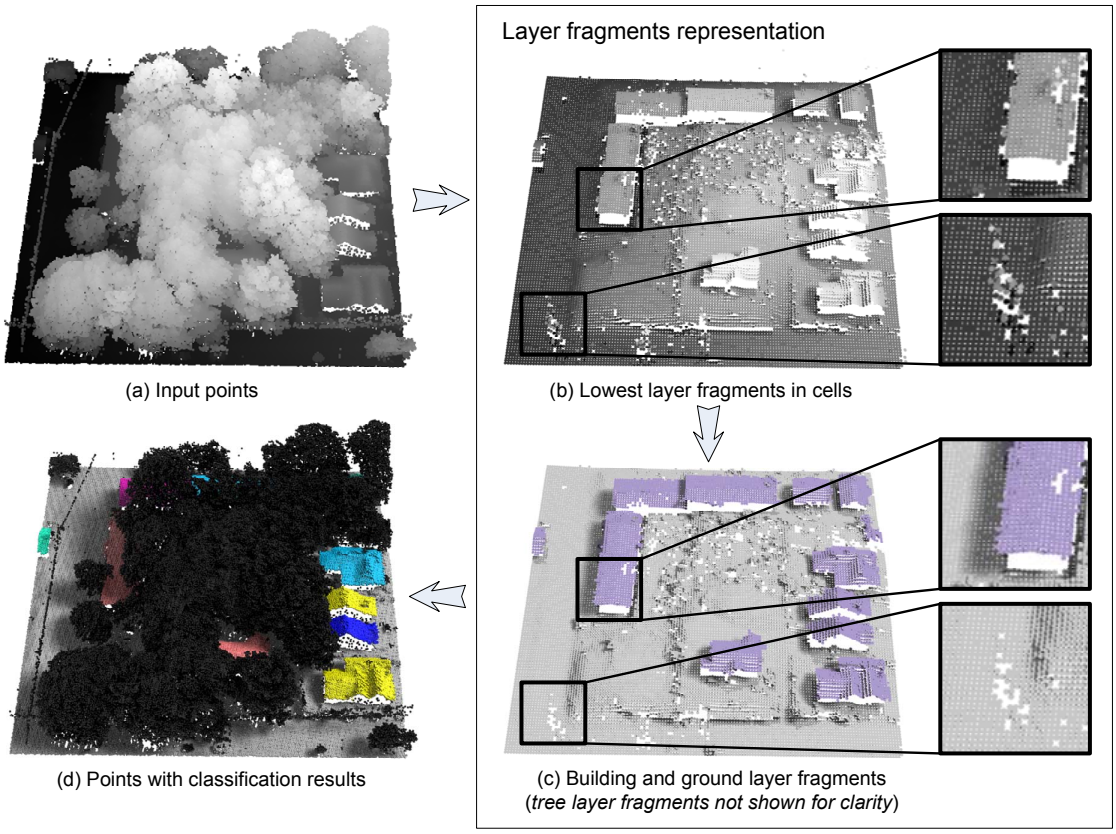


Figure 8.4: A demonstration of the classification algorithm: (b) the lowest layer fragments faithfully capture the skyward surfaces of 2.5D structures; (c) building and ground layer fragments are rendered in purple and grey respectively; (d) trees and outliers are in black while building roof patches are rendered in bright colors.

8.3 Modeling of Urban Elements

Based on the successful classification of input points, different modeling approaches are introduced for trees, buildings, and ground respectively.

8.3.1 Tree Modeling

Modern tree modeling approaches adopt a general tree structure composed of skeletal branches and leaves attached to them. Tree reconstruction usually begins with a

branch generation algorithm followed by a leaf modeling approach. However, unlike ground-based laser scans and imagery, aerial LiDAR date captures very few samples on branches, making branch generation a difficult task. Therefore, I choose to directly model tree leaves by fitting surface shapes around tree points having sufficient neighbors.

In particular, for each tree point p with sufficient neighbors, Principal Component Analysis is applied to its neighboring point set $N(p)$ to fit an ellipsoid. Eigenvectors $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2$ and eigenvalues $\lambda_0, \lambda_1, \lambda_2$ of the covariance matrix represent the axes directions and lengths of the ellipsoid respectively. The inscribed octahedron of the ellipsoid is then adopted to represent the local leaf shape around p. Specifically, an octahedron is created with six vertices located at $\{\mathbf{v}_p \pm s\lambda_0\mathbf{v}_0, \mathbf{v}_p \pm s\lambda_1\mathbf{v}_1, \mathbf{v}_p \pm s\lambda_2\mathbf{v}_2\}$, where $\mathbf{v}_p$ is the location of p and s is a user-given size parameter.

A uniform sampling over the tree point set P_{tree} can be applied to further reduce the scale of the reconstructed models.

8.3.2 Building Modeling

2.5D dual contouring detailed in Chapter 5 is adopted to create building models from rooftop patches through three steps: (1) sampling 2.5D Hermite data over a uniform 2D grid, (2) estimating a hyper-point in each grid cell, and (3) generating polygons.

The only challenge in applying 2.5D dual contouring to residential area data lies in rooftop holes caused by occlusion. To solve this problem, a hole-filling step is added right after 2.5D Hermite data is sampled from input points. In particular, the hole-filling step scans the entire 2D grid to detect rooftop holes, and solves a Laplace's equation $\nabla^2 z = 0$ to fill these holes, where z represents the heights of *surface Hermite samples*

at grid corners[2]. Existing surface Hermite samples serve as the boundary condition of the Laplace's equation.

8.3.3 Ground Modeling

Ground models can be easily created by rasterizing ground points into a DSM (digital surface model). Holes are filled via linear interpolation.

8.4 Experimental Results

The residential urban modeling system is tested on various data sets. For each data set, the following parameter configuration is adopted with respect to the data resolution. The neighborhood radius r is set to $\frac{3}{\sqrt{d}}$ given d as the point density in m^{-2}, to ensure sufficient samples in a point's neighborhood. Octahedron size s is chosen by the user in the interval $[\frac{1}{r}, \frac{3}{r}]$. Energy function parameters, *i.e.*, $\{\alpha, \beta, \gamma, \lambda\}$ are set to $\{1.0, 2.0, 0.5, 4.0\}$ empirically. This parameter configuration works well for all the data sets that have been tested.

Figure 8.6 shows the urban reconstruction results for a 520m-by-460m residential area in the city of Atlanta. The input contains 5.5M aerial LiDAR points with 22.9/m^2 resolution. The modeling system reconstructs 56K triangles for building models, and 53K octahedrons as tree leaves, in less than two minutes on a consumer-level laptop. As illustrated in the closeups of Figure 8.6, the classification algorithm successfully classifies points into trees, ground, and individual building patches (second column). A hybrid urban model is generated by combining 2.5D polygonal building models and

[2]Surface Hermite data is sampled per grid corner, by intersecting a vertical line and the rooftop surface. See Section 5.3.1 for details.

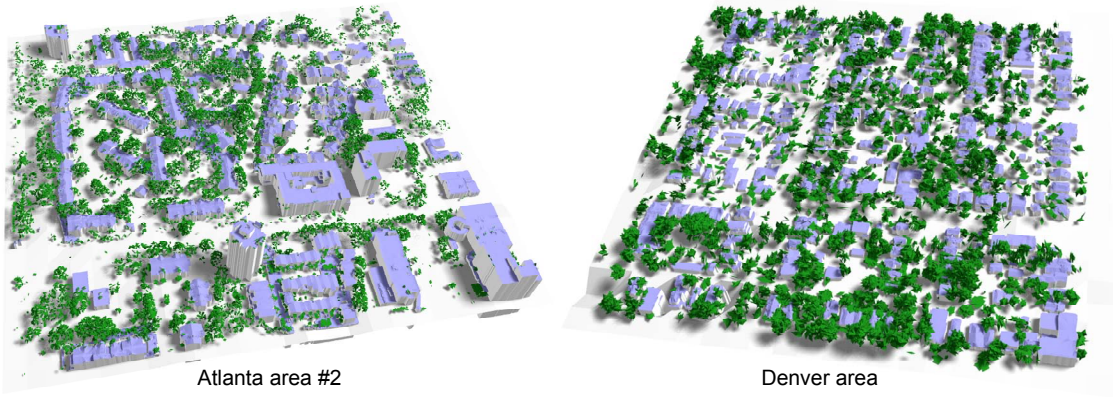


Figure 8.5: The residential urban modeling system is tested against multiple data sets from different sources. The system robustly reconstructs urban reality despite the variation in data resolution, building patterns, and tree types.

leaf-based tree models (third column). Aerial imagery is given in the last column as a reference.

The residential urban modeling system is then tested on another two data sets, with data resolution ranging from 6/m^2 to 19/m^2. Visually appealing urban models are reconstructed respectively, despite the variation in point density, building model patterns, and tree types, as shown in Figure 8.5. To quantitatively evaluate the modeling results, the false positives (unexpected results) and false negatives (missing results) of buildings are counted by comparing modeling results with aerial imagery as a trusted external judgement. In all three experiments, no false negative is found, *i.e.*, all building structures are successfully detected and reconstructed by the modeling system. In addition, false

Data set	Point #	Resolution	Building #	Building Tri. #	Building error rate	Octahedron #
Atlanta #1	5.5M	22.9/m^2	418	55,568	1.1%	52,924
Atlanta #2	4.0M	18.8/m^2	323	61,492	0.7%	29,151
Denver	1.0M	6.3/m^2	290	42,942	0.6%	17,054

Table 8.1: Statistics of the experiments on three different data sets

Data set	Classification	Normal estimation	Building modeling	Tree modeling	Total time
Atlanta #1	9s	45s	54s	<1s	109s
Atlanta #2	6s	29s	32s	<1s	68s
Denver	3s	8s	11s	<1s	23s

Table 8.2: Execution time of each step in the modeling system

positives exist in the form of small building-like trees and incorrectly classified ground components. The error rate is then calculated as the ratio of the number of triangles in false positives to the total number of triangles in all building models. Table 8.1 contains statistics of the three experiments, in which error rates are generally low. Table 8.2 shows the computation time, measured on a laptop with Intel i-7 CPU 1.60GHz and 6GB memory.

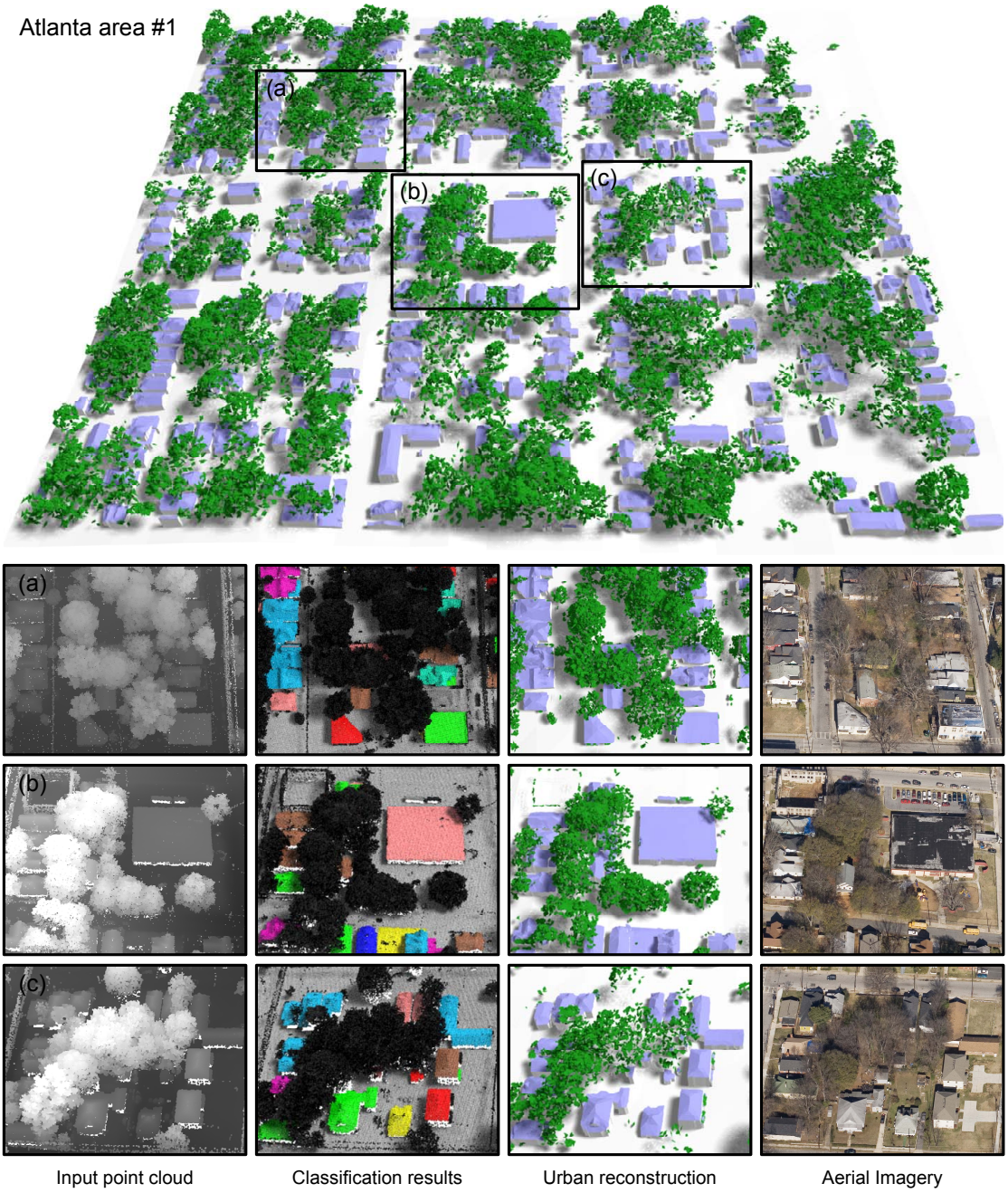


Figure 8.6: Urban models reconstructed from 5.5M aerial LiDAR points for a residential area in the city of Atlanta

Chapter 9

Conclusion and Future Work

9.1 Conclusion

This research studies the complicated problem of reconstructing 3D urban models from aerial LiDAR point clouds. An automatic urban modeling system is proposed in Chapter 3, which divides this problem into four sub-problems, namely, classification, segmentation, building modeling, and terrain modeling. Automatic algorithms are given to solve these problems individually. Two extensions based on prior knowledge are introduced to enhance the system, including a principal direction snapping mechanism and a non-planar shape modeling module.

In order to give the urban modeling system the ability to seamlessly handle huge data sets with billions of LiDAR points, Chapter 4 proposes a streaming framework which utilizes an out-of-core processing architecture. By decomposing each pipeline module into streaming operators and streaming states, the general urban modeling system is adapted into the streaming framework. Experiments are done on a few large-scale data sets, which no previous approach is able to process with such a small amount of resource.

From Chapter 5 to Chapter 8, I study the 2.5D nature of building structures. Theories and algorithms are proposed based on the 2.5D characteristic of building models.

In Chapter 5, a 2.5D geometry representation is given for polygonal building models. A robust data-driven method is proposed to automatically create building models from aerial LiDAR point clouds. The reconstruction results are 2.5D models composed of

complex building roofs connected by vertical walls. By extending dual contouring into a 2.5D method, the 2.5D dual contouring algorithm optimizes the surface geometry and the boundaries of roof layers simultaneously. Sharp features are detected and faithfully preserved during triangulation.

Chapter 6 defines 2.5D building topology as a combination of topological features and the associations between them. Convenient tools are given to change model geometry without modifying the topology. In addition, 2.5D dual contouring is extended with topology control strategies, to achieve a more flexible adaptive structure for simplification. The outputs have the same representability as models created by 2.5D dual contouring, but contain fewer vertices and triangles.

Chapter 7, defines global regularities in 2.5D building models to characterize the intrinsic structure of building models. A primitive-based algorithm is proposed to discover and enforce global regularities through a series of alignment steps. This building modeling method automatically integrates global regularities and local geometry, and thus creates 2.5D building models with high quality in terms of both geometry and visual judgement.

Finally in Chapter 8, the urban modeling problem is extended from downtown areas to residential urban areas with rich vegetation. I observe the key difference between buildings and trees in terms of the 2.5D characteristic: while buildings are composed of opaque skyward rooftops and vertical walls, trees allow point samples underneath the crown. This feature enables a powerful classification algorithm based on an energy minimization scheme. By combing classification, building modeling and tree modeling together, the residential urban modeling system automatically reconstructs a hybrid model composed of buildings and trees from the aerial LiDAR scan. The experiments demonstrate the effectiveness and efficiency of this system.

9.2 Future Work

Possible future work lies within three directions.

Modeling from dense LiDAR data

With the fast advance of acquisition techniques, extremely dense LiDAR point clouds become available. *E.g.*, a scan for the city of Vancouver in the year 2011 has approximately 70 samples/m^2 resolution, comparing with $6 \sim 17$ samples/m^2 resolution in the experiments shown in this thesis. These extremely dense data sets bring two new challenges. First, besides typical urban features (buildings, trees, and ground), small urban features such as vehicles, poles, and signal lights are also accessible in the dense LiDAR data sets. It is a desire to reconstruct these features towards a better urban reality. Second, detailed geometry of building facades are partially captured by the dense LiDAR scans. Building modeling method can benefit from this additional information.

Building modeling with both rooftops and facades

This thesis adopts the 2.5D characteristic of building structures, and describes the building facades as plain walls orthogonal to the x-y plane. However, in some application, building models are required to have both rooftops and detailed facade structures. While this thesis focuses on the building roof reconstruction, many research efforts have attempted to model building facades from imagery (*e.g.*, [42]) or ground based LiDAR (*e.g.* [43, 67]). Future research may seek a conjunction between roof modeling and facade modeling. The hybrid approach should connect these two types of building structures while maintaining the consistency at roof boundaries. In addition, the 2.5D nature of building models may serve as a supporting structure for facade modeling approaches. The global regularities in 2.5D building models can be further extended to handle the facade structures.

Integrating aerial LiDAR with other data sources

Aerial LiDAR data has great advantages in the urban reconstruction problem, as demonstrated in this thesis. However, its limitation is also noticed: as point samples are collected from nadir perspective, aerial LiDAR data hardly captures surfaces beneath opaque objects (*e.g.*, bridges, highways, and vehicles). Future research may alleviate this problem by introducing other data sources to the urban modeling system, including aerial imagery, ground based LiDAR, and panorama images (street view).

Bibliography

[1] A. Alharthy and J. Bethel. Heuristic filtering and 3d feature extraction from lidar data. In *ISPRS Commission III, Symposium*, 2002.

[2] B. E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. *Computational Learing Theory*, pages 144–152, 1992.

[3] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE PAMI*, 2001.

[4] M. Carlberg, P. Gao, G. Chen, and A. Zakhor. Classifying urban landscape in aerial lidar using 3d shape analysis. *IEEE ICIP*, 2009.

[5] G. Chen and A. Zakhor. 2d tree detection in large urban landscapes using aerial lidar data. *IEEE ICIP*, 2009.

[6] D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. In *ACM SIGGRAPH*, 2004.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, second edition, 2001.

[8] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 1995.

[9] J.-F. Côté, J.-L. Widlowski, R. A. Fournier, and M. M. Verstraete. The structural and radiative consistency of three-dimensional tree reconstructions from terrestrial lidar. *Remote Sensing of Environment*, 2009.

[10] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *ACM SIGGRAPH*, 1996.

[11] A. Elaksher and J. Bethel. Reconstructing 3d buildings from lidar data. In *ISPRS Commission III, Symposium*, 2002.

[12] B. S. Everitt, S. Landau, and M. Leese. *Cluster Analysis*. Wiley, 4th edition, 2009.

[13] M. Fiocco, G. Boström, J. G. M. Gonçalves, and V. Sequeira. Multisensor fusion for volumetric reconstruction of large outdoor areas. *3DIM*, 2005.

[14] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 1981.

[15] R. A. Fisher. The statistical utilization of multiple measurements. *Annals of Eugenics*, pages 376–386, 1938.

[16] R. B. Fisher. Applying knowledge to reverse engineering problems. *CAD*, 2004.

[17] D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, 1999.

[18] C. Früh, S. Jain, and A. Zakhor. Data processing algorithms for generating textured 3d building facade meshes from laser scans and camera images. *IJCV*, 2005.

[19] R. Gal, O. Sorkine, N. J. Mitra, and D. Cohen-Or. iwires: An analyze-and-edit approach to shape manipulation. In *ACM SIGGRAPH*, 2009.

[20] M. Garland. Quadric-based polygonal surface simplification. *PhD thesis, Computer Science Department, Carnegie Mellon University, CMU-CS-99-105*, 1999.

[21] Google. Google 3d warehouse. *http://sketchup.google.com/3dwarehouse/*.

[22] T. L. Haithcoat, W. Song, and J. D. Hipple. Building footprint extraction and 3-d reconstruction from lidar data. In *IEEE/ISPRS Joint Workshop on Remote Sensing and Data Fusion over Urban Areas*, 2001.

[23] A. Hatcher. *Algebraic Topology*. Cambridge University Press, first edition, 2001.

[24] J. Hu, S. You, and U. Neumann. Integrating lidar, aerial image and ground images for complete urban building modeling. In *3DPVT*, 2006.

[25] M. Isenburg and S. Gumhold. Out-of-core compression for gigantic polygon meshes. In *ACM SIGGRAPH*, 2003.

[26] M. Isenburg and P. Lindstrom. Streaming meshes. In *IEEE Visualization*, 2005.

[27] M. Isenburg, Y. Liu, J. Shewchuk, and J. Snoeyink. Streaming computation of delaunay triangulations. In *ACM SIGGRAPH*, 2006.

[28] M. Isenburg, Y. Liu, J. Shewchuk, J. Snoeyink, and T. Thirion. Generating raster dem from mass points via tin streaming. In *Proceedings of Geographic Information Science*, 2006.

[29] T. Joachims. Svm light. *http://svmlight.joachims.org/*, 2004.

[30] I. T. Jolliffe. *Principal Component Analysis*. Springer, second edition, 2003.

[31] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring on hermite data. In *ACM SIGGRAPH*, 2002.

[32] Y. Kurozumi and W. Davis. Polygonal approximation by the minimax method. In *Computer Graphics and Image Processing*, 1982.

[33] F. Lafarge, X. Descombes, J. Zerubia, and M. Pierrot-Deseilligny. Building reconstruction from a single dem. In *CVPR*, 2008.

[34] F. Lafarge and C. Mallet. Building large urban environments from unstructured point data. In *ICCV*, 2011.

[35] J.-F. Lalonde, N. Vandapel, and M. Hebert. Data structure for efficient dynamic processing in 3-d. *IJRR*, 2007.

[36] Y. Li, X. Wu, Y. Chrysanthou, A. Sharf, D. Cohen-Or, and N. J. Mitra. Globfit: Consistently fitting primitives by discovering global relations. In *ACM SIGGRAPH*, 2011.

[37] P. Lindstrom. Out-of-core simplification of large polygonal models. In *ACM SIGGRAPH*, 2000.

[38] Y. Livny, S. Pirk, Z. Cheng, F. Yan, O. Deussen, D. Cohen-Or, and B. Chen. Texture-lobes for tree modelling. In *ACM SIGGRAPH*, 2011.

[39] S. K. Lodha, D. M. Fitzpatrick, and D. P. Helmbold. Aerial lidar data classification using adaboost. In *3DIM*, 2007.

[40] W. Lorensen and H. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM SIGGRAPH*, 1987.

[41] B. Matei, H. Sawhney, S. Samarasekera, J. Kim, and R. Kumar. Building segmentation for densely built urban regions using aerial lidar data. In *CVPR*, 2008.

[42] P. Müller, G. Zeng, P. Wonka, and L. Van Gool. Image-based procedural modeling of facades. In *ACM SIGGRAPH*, 2007.

[43] L. Nan, A. Sharf, HaoZhang, D. Cohen-Or, and B. Chen. Smartboxes for interactive urban reconstruction. In *ACM SIGGRAPH*, 2010.

[44] B. Neubert, T. Franken, and O. Deussen. Approximate image-based tree-modeling using particle flows. In *ACM SIGGRAPH*, 2007.

[45] M. Nielsen, O. Nilsson, A. Soderstrom, and K. Museth. Out-of-core and compressed level set methods. In *ACM SIGGRAPH*, 2007.

[46] R. Pajarola. Stream-processing points. In *IEEE Visualization*, 2005.

[47] M. Pauly. Point primitives for interactive modeling and processing of 3d geometry. *PhD thesis, ETH Zurich*, 2003.

[48] C. Poullis and S. You. Automatic reconstruction of cities from remote sensor data. In *CVPR*, 2009.

[49] G. Priestnall, J. Jaafar, and A. Duncan. Extracting urban features from lidar digital surface models. *Computers, Environment and Urban Systems*, 2000.

[50] F. Rottensteiner. Automatic generation of high-quality building models from lidar data. *IEEE Computer Graphics and Applications*, 2003.

[51] S. Schaefer, T. Ju, and J. Warren. Manifold dual contouring. *IEEE Trans. Vis. Comput. Graph.*, 2007.

[52] J. Secord and A. Zakhor. Tree detection in urban regions using aerial lidar and image data. *IEEE Geoscience and Remote Sensing Letters*, 2007.

[53] X. Shi, H. Bao, and K. Zhou. Out-of-core multigrid solver for streaming meshes. In *ACM SIGGRAPH Asia*, 2009.

[54] P. Soille. *Morphological Image Analysis: Principles and Applications*. Springer-Verlag Telos, 2002.

[55] P. Tan, T. Fang, J. Xiao, P. Zhao, and L. Quan. Single image tree modeling. In *ACM SIGGRAPH Asia*, 2008.

[56] P. Tan, G. Zeng, J. Wang, S. B. Kang, and L. Quan. Image-based tree modeling. In *ACM SIGGRAPH*, 2007.

[57] S. Thrun and B. Wegbreit. Shape from symmetry. In *ICCV*, 2005.

[58] A. Toshev, P. Mordohai, and B. Taskar. Detecting and parsing architecture at city scale from range data. In *CVPR*, 2010.

[59] G. Varadhan, S. Krishnan, Y. Kim, and D. Manocha. Feature-sensitive subdivision and iso-surface reconstruction. In *IEEE Visualization*, 2003.

[60] V. Verma, R. Kumar, and S. Hsu. 3d building detection and modeling from aerial lidar data. In *CVPR*, 2006.

[61] H. Vo, S. Callahan, P. Lindstrom, V. Pascucci, and C. Silva. Streaming simplification of tetrahedral meshes. *IEEE Trans. Vis. Comput. Graph.*, 2007.

[62] O. Wang, S. K. Lodha, and D. P. Helmbold. A bayesian approach to building footprint extraction from aerial lidar data. In *3DPVT*, 2006.

[63] H. Xu, N. Gossett, and B. Chen. Knowledge and heuristic-based modeling of laser-scanned trees. *ACM Trans. Graph.*, 2007.

[64] S. You, J. Hu, U. Neumann, and P. Fox. Urban site modeling from lidar. In *Proceedings, Part III*, ICCSA, 2003.

[65] L. Zebedin, J. Bauer, K. Karner, and H. Bischof. Fusion of feature- and area-based information for urban buildings modeling from aerial imagery. In *ECCV*, 2008.

[66] N. Zhang, W. Hong, and A. Kaufman. Dual contouring with topology preserving simplification using enhanced cell representation. In *IEEE Visualization*, 2004.

[67] Q. Zheng, A. Sharf, G. Wan, Y. Li, N. J. Mitra, B. Chen, and D. Cohen-Or. Non-local scan consolidation for 3d urban scene. In *ACM SIGGRAPH*, 2010.

[68] Q.-Y. Zhou, T. Ju, and S.-M. Hu. Topology repair of solid models using skeletons. *IEEE Trans. Vis. Comput. Graph.*, 2007.