Fast and Extensible Building Modeling from Airborne LiDAR Data

Qian-Yi Zhou
University of Southern California
qianyizh@usc.edu

Ulrich Neumann
University of Southern California
uneumann@graphics.usc.edu

ABSTRACT

This paper presents an automatic algorithm which reconstructs building models from airborne LiDAR (light detection and ranging) data of urban areas. While our algorithm inherits the typical building reconstruction pipeline, several major distinct features are developed to enhance efficiency and robustness: 1) we design a novel vegetation detection algorithm based on differential geometry properties and unbalanced SVM; 2) after roof patch segmentation, a fast boundary extraction method is introduced to produce topology-correct water tight boundaries; 3) instead of making assumptions on the angles between roof boundary lines, we propose a data-driven algorithm which automatically learns the principal directions of roof boundaries and uses them in footprint production. Furthermore, we show the extendability of our algorithm by supporting non-flat object patterns with the help of only a few user interactions. We demonstrate the efficiency and accuracy of our algorithm by showing experiment results on urban area data of several different data sets.

Categories and Subject Descriptors

I.4.8 [**Image Processing And Computer Vision**]: Scene Analysis—*Range data*

General Terms

Algorithm, Experimentation, Performance

Keywords

LiDAR(light detection and ranging), building modeling, segmentation, building footprints

1. INTRODUCTION

3D building models have long been useful in a variety of applications in geographic information systems, such as urban planning and virtual city tourism. In these applications, building models are preferred to be composed of several *simple* parts and organized in a *meaningful* way.

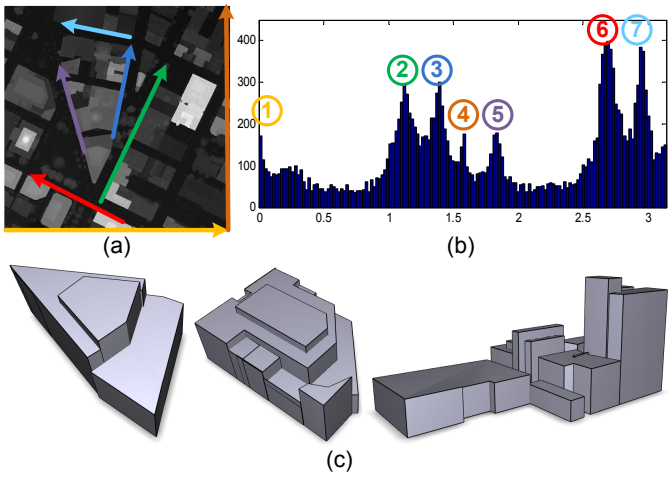

Figure 1: Building modeling of Oakland city: (a) input LiDAR data, point height is illustrated as intensity; (b) a histogram learned from original data showing the seven *principal directions* of this area, which are also illustrated as arrows of the same color in (a); (c) some building modeling results, both orthogonal corners and non-orthogonal corners are reconstructed correctly.

One way to obtain such data is to let skillful workers create building models manually based on building blueprints, aerial images and other data sources. This solution, however, requires a large amount of manual work, thus is both slow and expensive. Airborne LiDAR (Light Detection and Ranging) data, on the other hand, provides a faster and lower cost means to gather first-hand data of selected urban areas. With laser scanners equipped on aeroplanes, a 3D point cloud is collected, which samples the surface of an urban site in an accurate and fast manner. Directly converting LiDAR data into a DSM (Digital Surface Model) provides us another way of obtaining building models. These models, however, consist of millions of triangles and contain undesired noise and vegetation. Therefore they satisfy neither the *simple* requirement nor the *meaningful* requirement.

To bridge this large gap between LiDAR data and 3D building models, much research effort has focused on the building reconstruction problem. There are two major differences between our work and the existing approaches. First, we show that exploiting shared properties of specific types

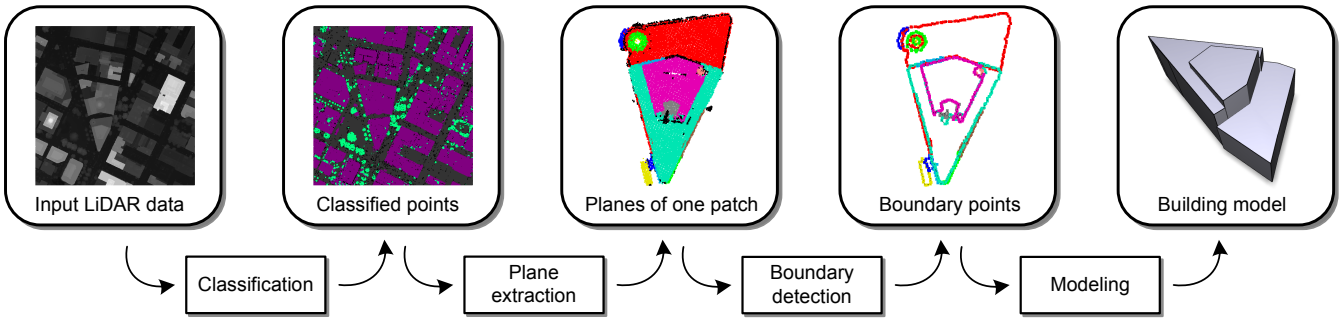

Figure 2: Building modeling pipeline: classification separates vegetation points from roofs and ground; planes are extracted from roof patches and boundaries of each plane are detected; finally, building models are reconstructed from boundary points.

of areas will lead to better reconstruction performance. In this paper we concentrate on the urban areas. Our algorithm can adapt to properties such as vegetation patterns and buildings layouts, therefore yielding better result based on the context knowledge. Second, experiments are done on several data-sets of two cities and one industrial site to show the generality and extendability of our approach.

1.1 Algorithm overview

In this paper, we present a fully automatic algorithm which creates simple and meaningful building models from urban LiDAR data. Our algorithm requires only the LiDAR data as input and works directly on the original point cloud without rasterization. This provides us with more information especially at the multi-layered area,[1] *e.g.* roof edges and trees. Our output, on the other hand, is a set of watertight models, each of which is composed of a few triangles. The boundaries of neighbor building components are aligned together to reflect the relationship between them.

Most existing works on building reconstruction from LiDAR data follow a three-stage pipeline (Figure 2): first, irrelevant parts such as trees and ground are removed from the LiDAR data; second, roof patches are separated and extracted from the remaining point cloud, and significant features (*e.g.* boundaries) are detected; finally, with these features, each roof patch is fitted to a pre-defined parametric model which is further used in creating polygonal models.

While sharing a similar pipeline, we introduce novel mechanisms to adapt to urban properties and to enhance efficiency. Our contribution mainly includes:

- For classification of vegetation from other urban objects, we introduce a SVM (Support Vector Machine) algorithm which takes several *differential geometry* properties as features, instead of using global-aware features such as height and intensity. Therefore, the learning algorithm only needs to be trained once, and the solution could be reused in other data sets without retraining.

- We propose an efficient and robust boundary extraction algorithm by extending the 3D OcTree contouring algorithm in the field of volumetric geometry processing[17]. Compared with existing boundary extraction algorithm, our method is topology-preserving, faster and much easier to implement.

- For the last stage of the pipeline, we design a data-driven algorithm which learns the principal directions from original data, and aligns roof boundary segments along these directions automatically. As shown in Figure 1, this mechanism enables us to handle arbitrary angles and avoid the pre-assumption or preference for specific angles between neighboring roof boundary segments which is often assumed in previous works (*e.g.* 90°, 45° and 135° in [15]).

- We show our algorithm can be easily extended to handle non-flat object patterns by adding minimal user interaction.

The rest part of this paper is organized as follows: Section 2 summarizes the related literature to this paper; Section 3 presents the algorithm to detect and eliminate vegetation (mainly trees) from original data; Section 4 describes our boundary extraction algorithm; in Section 5, we illustrate the final step of our pipeline - generation of final building models; Section 6 presents an extension to our approach for handling non-flat object patterns, as well as some implementation details; Section 7 gives the experiment results and Section 8 concludes the paper.

2. RELATED WORKS

2.1 Building modeling

Pioneer building modeling algorithms, *e.g.* [2][12][16][11][5][1], convert LiDAR point cloud into a DSM (Digital Surface Model) as their first steps, and then apply image processing algorithms on the depth images to detect building footprints, fit parametric models and reconstruct polygons. Although not explicitly mentioned in all papers, most works follow the pipeline we summarized in Section 1.1. This pipeline is proved to be both useful and efficient in previous research.

Some recent research such as [14] and [15] have introduced the direction of constructing building models from the original LiDAR point cloud. These algorithms also follow the common pipeline, and are similar to our algorithm in structure. Verma *et al.*[14] focus on automatic building topology detection by searching pre-defined roof patterns in the roof-topology graph; while Wang *et al.*[15] concentrate on build-

[1]Note that LiDAR is composed of several pieces of scans from different perspectives, thus may contain multiple layers at the same x-y coordinate.

ing footprint extraction problem; both are different from our approach.

Another related but orthogonal research direction to our approach is to combine LiDAR data with other data sources. For instance, Früh and Zakhor[4] introduce both ground based and aerial LiDAR data. By integrating them, complete building models are constructed to contain not only detailed roofs, but also subtle facades. You *et al.*[16], on the other hand, allow users to make a few clicks and use these inputs as a heuristic guide in model reconstruction. As a subsequent work, Hu *et al.*[6] employ aerial image and ground images to achieve a higher-resolution solution.

2.2 Vegetation detection

As some researchers have pointed out, vegetation (trees) are the most difficult part to be separated from building roofs. Therefore, there are several works addressing this problem such as [13][9][14][15]. In general, these works employ both geometry and reflection properties at each sample point, and then use a classification algorithm to assign each point to corresponding class. The classification algorithm could be a simple thresholding[14], an SVM classifier[13], or an Adaboost classifer[9][15].

2.3 Building footprint detection

Besides tree detection, another well-studied subproblem is building footprint extraction. Haithcoat *et al.*[5] detect building footprints from a depth image, then use an orthogonal simplification algorithm to turn these footprints into rectangle corners. Wang *et al.*[15] raise a bayesian approach in building footprint extraction, which shows a preference to specific corner angles. Alharthy and Bethel[1]'s algorithm is the most similar to ours. They also make the assumption that boundary segments in a local area fall into a couple of dominant directions. However, their algorithm simply detects two orthogonal principal directions, thus limiting their application.

3. VEGETATION DETECTION

As we have discussed in section 1.1, vegetation, mainly in the form of trees in urban area, is an irrelevant part in our task of building reconstruction and should therefore be eliminated in the first step. Assume that $\mathbf{p} \in P$ is a sample point in the original LiDAR point cloud, $N_p = \{\mathbf{q}|\mathbf{q} \in P, \|\mathbf{p}-\mathbf{q}\| < \delta\}$ 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, we compute five features based on the local differential geometry properties:

1. **Regularity:** we first measure if the point distribution around a point is regular by calculating

$$\mathcal{F}_1 = \|\mathbf{p} - \overline{\mathbf{p}}\|. \tag{1}$$

Intuitively, sample distribution around a roof point is more likely to be regular, thus the distance between original point and centroid point should be smaller than that of vegetation points.

2. **Horizontality:** LiDAR captures roof from a top view. Therefore, the normal at a roof point is more likely to be vertical. In this case, we compute:

$$\mathcal{F}_2 = 1 - |\mathbf{n}_p \cdot \mathbf{e}_z|, \tag{2}$$

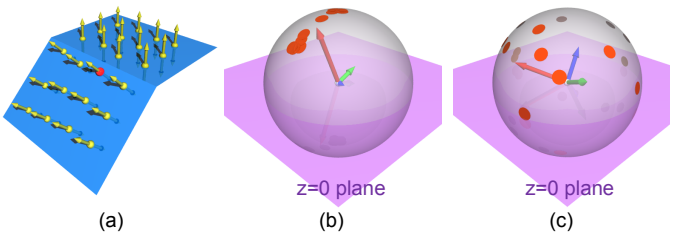

Figure 3: Illustration of normal variation measurements: (a) normals of points around a roof ridge; (b) normals of points from (a), distributed in a Gauss sphere. Red/green/blue arrows point to the eigenvectors of $\mathbf{C}_p^n$ with length equal to corresponding eigenvalues; (c) normals of points in the neighborhood of a tree point. Both λ_1^n and λ_2^n are large due to the irregularity of normals.

where $\mathbf{e}_z = (0,0,1)$ is the vertical direction, and $\mathbf{n}_p$ is obtained through covariance analysis [10].[2] In particular, we solve the eigenvector problem for the covariance matrix

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

The three eigenvalues are sorted in ascending order, *i.e.* $\lambda_0 \leq \lambda_1 \leq \lambda_2$; and the eigenvector corresponding to the smallest eigenvalue, *i.e.* $\mathbf{v}_0$, is the approximated normal at point $\mathbf{p}$.

3. **Flatness:** with covariance analysis results, the flatness at this point, also known as *surface variation*[10], could be estimated as

$$\mathcal{F}_3 = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2}. \tag{3}$$

Similar to the former 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, we could further apply another covariance analysis on these normals, but within a different neighborhood $N_p^n = \{\mathbf{q}|\mathbf{q} \in P, \|\mathbf{p}-\mathbf{q}\| < \eta\}$. We solve the eigenvector problem for a normal covariance matrix

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

and get corresponding eigenvalues $\lambda_0^n \leq \lambda_1^n \leq \lambda_2^n$. As Pauly [10] has pointed out, λ_1^n measures the maximum variation of these normals on the Gauss sphere, hence could be regarded as a kind of *normal variation*. Therefore we define

$$\mathcal{F}_4 = \lambda_1^n. \tag{4}$$

A roof point prefers a smaller normal variation than a tree point.
Moreover, we observe that λ_0^n also reflects some kind of regularity in normal distribution. An example is shown

[2]Note that the covariance analysis needs enough sample points in the neighborhood to make the matrix nonsingular. Hence, if the number of points in N_p is less than some specified number (experimentally, we require at least 10 points), we make the reasonable assumption that this point belongs to the *noise* category.

in figure 3. In this case, normals of points around a roof ridge (which is a common pattern in buildings) form two clusters on the Gauss sphere. Hence λ_1^n (illustrated as the length of the green vector in Figure 3(b)) is large while λ_0^n (length of the blue vector in figure 3(b)) is very small. In contrast, normal distribution around a tree point is fairly irregular and exhibits large λ_0^n and λ_1^n (shown in figure 3(c)). Therefore, our last feature is defined as

$$\mathcal{F}_5 = \lambda_0^n. \quad (5)$$

To this end, we use a linear discriminative function to classify trees from ground and roof area, denoted as:

$$\mathcal{K} = \omega_0 + \omega_1\mathcal{F}_1 + \omega_2\mathcal{F}_2 + \omega_3\mathcal{F}_3 + \omega_4\mathcal{F}_4 + \omega_5\mathcal{F}_5, \quad (6)$$

where $\omega_{0,1,2,3,4,5}$ are undetermined parameters which are then learned using an unbalanced soft margin SVM algorithm from a small urban area with manual labeling. We use a third-party library *SVM Light*[8] as our implementation. Once these parameters are determined, the classification algorithm simply computes $\mathcal{K}$ for each point and determine its category with $sgn(\mathcal{K})$.

To further improve the classification results, we introduce a voting algorithm as a post-processing step. The idea is based on the fact that points of same category usually occur together in space. Hence, for each point, we let all points in its neighborhood N_p vote for the category, and point $\mathbf{p}$ belongs to tree category only if the percentage of tree votes is greater than a certain value ω, which is also learned from the labeled data set.

Note that all of the features are designed based on a local geometry analysis, thus we avoid absolute variants such as height and intensity to enhance the generalization ability of the learned function. In our experiments, all the parameters are learned from a $100m \times 100m$ labeled area. They show great adaptability on our testing data sets.

The only parameter we need to set for each data set is δ and η. Empirically, we set δ to the value which makes the average point number in N_p between $13 \sim 15$, and η to two times the value of δ.

Our classification algorithm achieves an accuracy above 95% on all testing set. The feature values and final classification results are shown in Figure 4. Note that our algorithm could precisely classify trees and non-trees even at areas where points of both categories have similar height and are connected together.

4. ROOF BOUNDARY GENERATION

4.1 Planar roof patch extraction

Once trees are eliminated from the LiDAR data, we need to identify ground points and separate different roof patches. This is achieved using a distance-based region growing algorithm. Starting from a random seed point $\mathbf{p}$, the algorithm searches its neighborhood N_p, and assigns all untouched points with its distance to $\mathbf{p}$ less than a certain parameter α.[3] By iteratively running this algorithm, the LiDAR point cloud is divided into different patches. We accept the largest patch as ground. To further improve the results, we introduce a post-processing algorithm which merges low-height

[3]We choose $\alpha = 1m$ in our experiments.

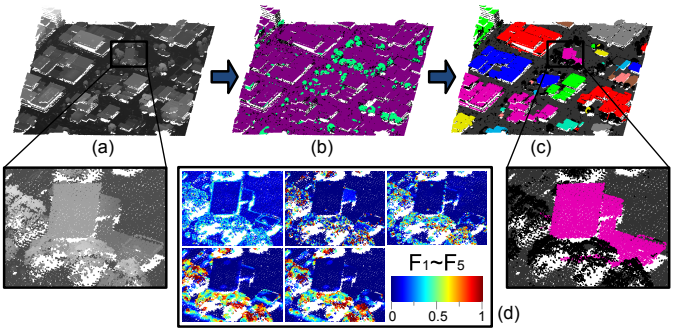

Figure 4: Tree detection and roof patch extraction of a piece of Oakland city: (a) original LiDAR data, color intensity represents the height at each point; (b) green points are detected as tree; (c) different roof patches are labeled with different colors; (d) five features at each point within a highlight area.

large patches into grounds. This is reasonable as in some area ground might be divided into several pieces due to the cutoffs on roads caused by occlusion. Finally, we combine roof patches with neighboring x and y coordinates into one roof for certain building. Figure 4(c) shows such a result.

Note that our method for this mainly focuses on planar roof patterns. Hence, we need to detect planar shapes from roof patches. We use a clustering algorithm based on the similarity between normals of neighboring points. In particular, the algorithm is similar to the region growing algorithm presented in the previous paragraph: it also searches the neighborhood N_p of point $\mathbf{p}$, but the classification is based on normal difference, *i.e.* whether $1 - |\mathbf{n}_p \cdot \mathbf{n}_q| < \beta$.[4]

4.2 Boundary production

The next step in our algorithm is to mark boundary points on each plane roof patch, which are later used in footprint generation and polygon construction. Some previous works, *e.g.* [15], find boundary points by measuring certain characteristics of roof points. These kinds of methods, although efficient, cannot guarantee a watertight manifold boundary, hence it limits the subsequent processing. Other works use 2D delaunay triangulation to find polygonal boundaries, *e.g.* Verma *et al.* [14] introduce a plane ball-pivoting algorithm to triangulate roof points and detect boundaries. These algorithms are able to generate perfect boundaries, however, the efficiency is sacrificed. In addition, a robust implementation for such algorithm is hard to achieve.

We propose an algorithm which combines the benefits of these two kinds of algorithm. Our method is inspired by the contouring algorithm in [17].

Initially, for each plane roof patch, all of the roof points are projected to the same plane. Then a uniform grid is applied onto this plane and all the grid cells containing roof points are marked. These grid cells are defined as *object cells*, which are illustrated as grey cells in Figure 5.

Note that each connected component of these object cells is surrounded by a watertight polygonal boundary composed of grid lines and corners, shown as the red grid lines and corners in the figure. Therefore, we construct a closed boundary for the LiDAR points as follows:

- For each boundary grid line l, which separates an ob-

[4]Any β between $1-\cos(5^\circ)$ and $1-\cos(10^\circ)$ is fine in our experiments.

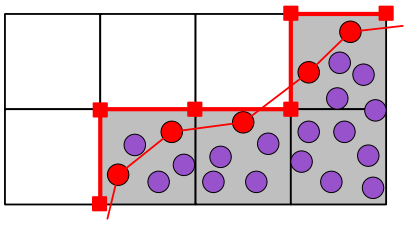

Figure 5: An illustration of the boundary extraction algorithm. Circles represent the LiDAR points projected on the plane. The red circles and red edges connecting them form up the boundary.

ject cell c_{in} and a background cell c_{out}, we take the nearest LiDAR point to l in c_{in} as a boundary point $\mathbf{p}(l)$, shown as red circles in Figure 5.[5]

- For each boundary grid corner c, which connects two boundary grid lines l_1 and l_2, we add a boundary edge $(\mathbf{p}(l_1), \mathbf{p}(l_2))$, shown as red thin edges in Figure 5.

In this way, we construct a watertight manifold boundary for every component of object cells. Our method is easy to implement, at the same time guarantees the correctness of topology. A detailed correctness proof could refer to the 2D version of proofs in [17]. The completeness of geometry, on the other hand, cannot be guaranteed by our method. For example, in Figure 5, there is one point outside the extracted boundary polygon. However, we find this not a common case and our boundary is a good approximation for later processing.

Another advantage of our algorithm is that it could support morphological operations in an easy manner. After marking *object* and *background* cells, one could treat this grid as a monochrome image and apply any morphological operation onto it. Figure 6 illustrates an example from part of an industrial site. Directly extracting boundaries from the object patches produces numerous artifacts, shown in (b). Hence, we introduce an *opening* morphological operation to remove the unimportant information as in (c).

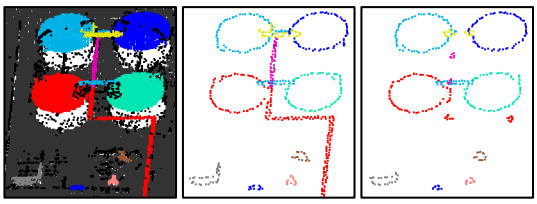

Figure 6: Boundary extraction for part of an industrial site: (b) without any morphological operation; (c) with an *opening* morphological operation, most artifacts are removed.

Only one parameter is involved in our boundary extraction algorithm, which is the unit length of the uniform grid. This parameter gives us the control of the connectivity of the patch, similar to the ball radius parameter in ball-pivoting algorithm used by [14]. Based on our experience, we find that a unit length equal to δ performs well on most of our data sets.

[5]To guarantee the correctness of topology, if a LiDAR point is assigned to different boundary grid lines, we duplicate it and assign different copies of this point to different lines. These copies might be merged together in later processing if the correctness of topology is not violated.

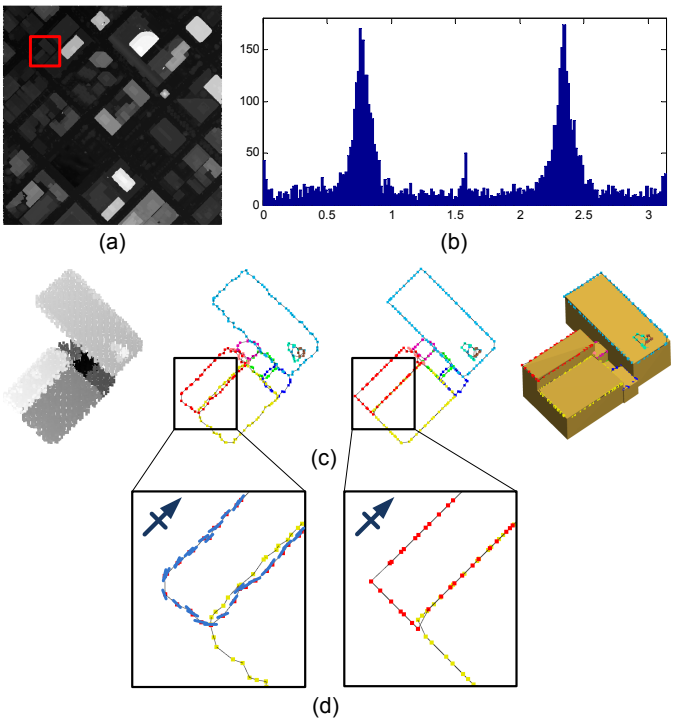

Figure 7: An illustration for the snapping algorithm: (a) LiDAR input of a square piece of Denver city; (b) histogram of tangent directions, an interesting observation is that 0° and 90° are also counted as principal directions due to the cut area boundary; (c) from left to right, initial point cloud, extracted boundary loops, snapped boundaries, and constructed polygonal model; (d) closeups of boundaries before and after snapping.

5. BUILDING MODELING

In this section, we present a data driven algorithm which generates simple and correct polygonal mesh models from the extracted roof patch boundaries. Our main idea is based on the following observations. First, most boundary line segments in a local area fall into a couple of directions, known as *principal directions*. Second, if two roof planes are neighbors when projected to the $x-y$ plane, they are very likely to share a same boundary line segment or be connected by a vertical wall. In the latter case, the two line segments connected by the wall are also overlapping on the $x-y$ plane. Based on these two observations, we design a 3-step algorithm as follows.

5.1 Find principal directions

For each boundary point, we first estimate the tangent direction across this point, by applying a 2D covariance analysis on seven nearby boundary points.[6] The estimated tan-

[6]Since we have a closed manifold boundary, we hereby find the nearby boundary points by traveling on the boundary through two opposite directions respectively.

gent directions of a building example is shown as the blue lines across red boundary points in Figure 7(d).

Although in general these tangent directions should follow the principal directions, yet due to the noise and insufficient sampling, boundaries often exhibit zigzags in detail and hence the tangents do not follow the principal directions very well in micro scope. As in Figure 7(d), blue tangent lines do not follow the principal directions (illustrated as the dark blue cross arrow) precisely.

To overcome the difficulty, we introduce statistical analysis. In particular, we record the tangent directions for all the boundary points in a local area, say a $1km \times 1km$ area, and build a histogram of direction angles. In this histogram, each peak represents a principal direction, to which numerous boundary points shows preference. Note that a histogram may have several peaks. Hence, we use an iterative algorithm to find all the peaks, as follows:

1. Find the highest peak in the histogram; take it as a principal direction, denoted as d_{peak}.
2. Remove this peak from the histogram. Note that a peak is similar to a Gauss distribution, hence we remove a whole section $[d_{peak} - \phi_0, d_{peak} + \phi_1]$ from the histogram, where ϕ_0 and ϕ_1 are found by following the falling trend along x axis until reaching a local minimum.
3. Repeat step 1 and step 2, until the highest peak does not contain enough samples.

Figure 7(b) illustrates the direction histogram of a Denver city piece shown in Figure 7(a). The four principal directions detected are $0^\circ, 43^\circ, 90^\circ, 133^\circ$ respectively, which could be separated into two orthogonal pairs. This phenomenon fits into the observation made by the previous works that orthogonal corners are a common pattern in building footprints. In addition, we notice the 0° and 90° directions contain less samples than the other two. In fact, this LiDAR point cloud piece is a square cut from the original data set, hence these two directions represent the directions at the area boundary.

5.2 Snapping

5.2.1 Snap to principal directions

Once the principal direction set $\mathcal{D}$ is determined, we divide the boundaries into line segments and align as many segments to the principal directions as possible. This process is performed for each boundary loop $\mathcal{B}$ through a greedy algorithm as follows:

For each boundary point $\mathbf{b}_i \in \mathcal{B}$, we test for every principal direction $\mathbf{d}_k \in \mathcal{D}$, and count the number of continuous boundary points of $\mathbf{b}_i$ which agree with line $l : (\mathbf{p} - \mathbf{b}_i) \times \mathbf{d}_k = 0$, denoted as $Count(\mathbf{b}_i, \mathbf{d}_k)$. The "agree" criteria is defined based on the distance from point $\mathbf{p}$ to line l, *i.e.*

$$distance(\mathbf{p}, l) = \frac{|(\mathbf{p} - \mathbf{b}_i) \times \mathbf{d}_k|}{|\mathbf{d}_k|} < \varepsilon. \quad (7)$$

We now pick the $(\mathbf{b}_i^{max}, \mathbf{d}_k^{max})$ pair which maximizes $Count(\mathbf{b}_i, \mathbf{d}_k)$, take line $l^{max} : (\mathbf{p} - \mathbf{b}_i^{max}) \times \mathbf{d}_k^{max} = 0$ as a boundary segment, and project the continuous boundary points which agree with l^{max} onto l^{max}.

We ignore all the projected points, recalculate $Count(\mathbf{b}_i, \mathbf{d}_k)$, and find the next $(\mathbf{b}_i^{max}, \mathbf{d}_k^{max})$ pair. The whole process is done iteratively, until the maximal number of agreement falls under a certain value (*e.g.* 10 points).

5.2.2 Neighbor segments snapping

The former snapping algorithm provides us line segments aligning to the principal directions. However, as we have pointed out, we could further improve the results by aligning neighbor line segments of different plane patches onto the same line. This is achieved through a neighbor segments snapping algorithm, as follows:

- For two neighbor boundary segments on the *same* plane patch, if they point toward the same direction and the distance between them is under a certain value; then we snap these two segments onto the same line. This operation solves the case in which a line segment is broken into two neighbor segments.
- For two boundary segments from *different* plane patch on same building roof, if they point toward the opposite direction (*i.e.* are of the opposite orientation), and the distance between them is under a certain value; then we snap these two segments onto the same line in $x - y$ plane, while keep the z coordinates for each segment. This operation eliminates the gaps between neighbor plane patches due to the noise and insufficient sampling.

Figure 7(d) shows a case of snapping. The boundary points shown in the left figure are rather irregular, while in the right figure, points have been snapped to the correct line segments. Note that a red segment and a yellow segment are snapped together in the right figure. This exposes as a vertical wall in the final reconstructed polygonal mesh.

5.3 Generate polygonal mesh

Finally, we finish the whole pipeline by constructing a polygonal mesh from the extracted boundary line segments. Typically, we construct a polygon for each boundary loops on the plane patches, and further connect them to the ground by adding vertical walls. Note that the boundaries we have on each plane patch are already composed of topology-correct loops. Thus, for each loop, we generate a polygon based on the line segments. Consider a pair of neighbor segments of a same boundary loop, if their end points are close enough, we create a new footprint as the intersection point of these two segments. Otherwise, we find the best approximation line to the interior boundary points and thus link all the segments into a complete polygon.

Figure 7(c) shows a whole pipeline to create a building model. Starting from a building roof patch, plane patches are detected and boundaries are extracted. Boundary points are then snapped and form up the line segments. Finally, a polygonal model is constructed from these segments, which is shown in the last figure. The final result, produced from 9,778 roof points, only contains 72 triangles, yet representing this building in a precise manner.

6. POST PROCESSING AND EXTENSIONS

6.1 Ground modeling

To construct the complete city model, we still need a ground model which could be combined with the building

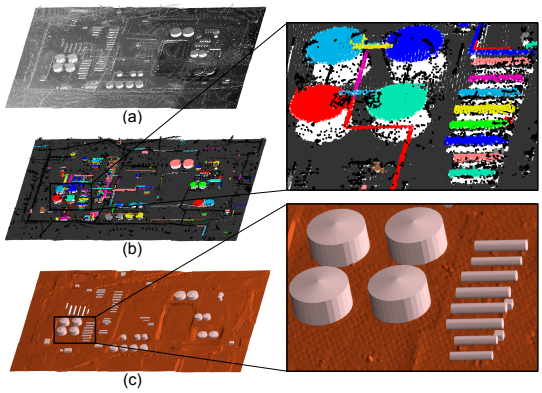

Figure 8: An example of an industrial site: (a) input LiDAR data; (b) detected object patches, different patches are rendered with different colors, ground and noise are dark-grey and black respectively; (c) the reconstructed models, different types of objects are generated in different ways.

models. We use the terrain geometry modeling algorithm presented in [14]. Briefly speaking, a uniform grid is aligned onto the ground points of our urban patch. If a grid cell contains some ground points, its ground height is computed as the average height of them. We then solve a Laplace's equation $\nabla^2 z = 0$ taking the heights in all empty grid cells as unknowns. In particular, for each empty cell at (i, j), we have,

$$4z_{i,j} = z_{i-1,j} + z_{i+1,j} + z_{i,j-1} + z_{i,j+1}. \quad (8)$$

With this equation array, all the heights could be calculated and a DSM is generated from them as the ground mesh.

6.2 Non-flat objects support

One limitation of the automatic pipeline presented in former sections is that it only supports flat roofs. However, in some cases, there are some non-flat objects in our data set. For example, figure 8(a) shows an industrial site in which the most interesting objects are the oil tins and tanks represented in cylinder and cone shapes. These shapes increase the complexity of the object reconstruction problem. However, fortunately, we find that these non-flat shapes usually fall into only a few specific patterns. In this case, once the pattern type is known, we could apply typical pattern recognition algorithms such as RANSAC[3].

Therefore, we provide a user input mechanism to help with this task. The initial LiDAR data is first segmented into different patches and the noise and ground points are removed from the data set. Now the object patches are shown to the user as illustrated in Figure 8(b). The user could move his mouse onto the object patch, click to select this patch, and press a key to specify its pattern type, *e.g.* cone, cylinder, or plane-roof object. Since we could extract boundary loops for each object, as presented in the previous sections, now we perform a RANSAC algorithm[3] based on the specified pattern type input by the user. Finally, we construct the complete model as shown in Figure 8(c).

7. EXPERIMENTS

We have tested our algorithm on several types of LiDAR data taken from 3 data sets - Denvor city, Oakland city, and an industrial site. The resolution of our data sets varies from 6 samples/sq.m. to 17 samples/sq.m. Our experiments show that the time cost of our approach is proportional to the number of points. We test all the data sets on a consumer-level laptop (Intel Core2 1.83GHz CPU, 2G memory) with an external hard disk, and find the ratio be around 8 minutes/million points.

Figure 9 shows the reconstruction results of a part of Denver city. Our building models are composed of simple and clean triangular meshes, and thus are ideal to some applications such as digital city visualization. In addition, the basic structures of these building models are aligned tightly together and form buildings of good shape. As shown in closeups of Figure 9(b), these building models fit our initial LiDAR point cloud in an excellent manner.

To further demonstrate the ability of supporting multiple principal directions, we test our program on part of Oakland city, as shown in Figure 1. Seven principal directions are automatically detected from the original data sets, illustrated as the arrows in Figure 1(a). Since our program 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 1(c).

As we have pointed out in Section 6.2, there are cases where our automatic flat roof detection algorithm cannot work well. Hence, we propose a semi-automatic algorithm which benefits from a few user inputs. Figure 8 demonstrates an example of an industrial site. In this example, we define three object patterns, namely, a standing cylinder with a cone on top of it, representing a tin; a lying cylinder representing a tank; and plane shaped roofs. Using our algorithm, objects of all three patterns could be detected and reconstructed, as shown in closeup of figure 8(c).

Finally, we would like to point out one limitation of our algorithm. In this paper, we use airborne LiDAR data as the whole input. However, this kind of data contains depth information almost only from the top view, thus occlusion becomes a serious problem in case when the "vertical wall assumption" does not hold true. For example, in our industrial site data set, we lack point samples under tanks and pipes, hence we experience difficulties in reconstructing such objects. Another observation is that, when we try to apply our algorithm to the residential areas of some cities, trees become the majority objects and house roofs are usually partial occluded by the trees. In such cases, it is much more difficult for our algorithm to extract the precise boundaries for these roofs.

8. CONCLUSION

This paper presents an automatic building modeling algorithm for airborne LiDAR data. The roof and ground points are first separated from tree points by applying an SVM method based on local geometry property analysis. Once the plane roof patches are generated through clusters of roof points, a novel boundary detection algorithm is applied to extract a watertight manifold boundary for each planar patch. These boundaries are later analyzed to find principal directions of a local area, which are used as the guidance of a snapping algorithm. After all these steps, most roof points are snapped onto certain boundary line segments, and polygonal building models are produced from them.

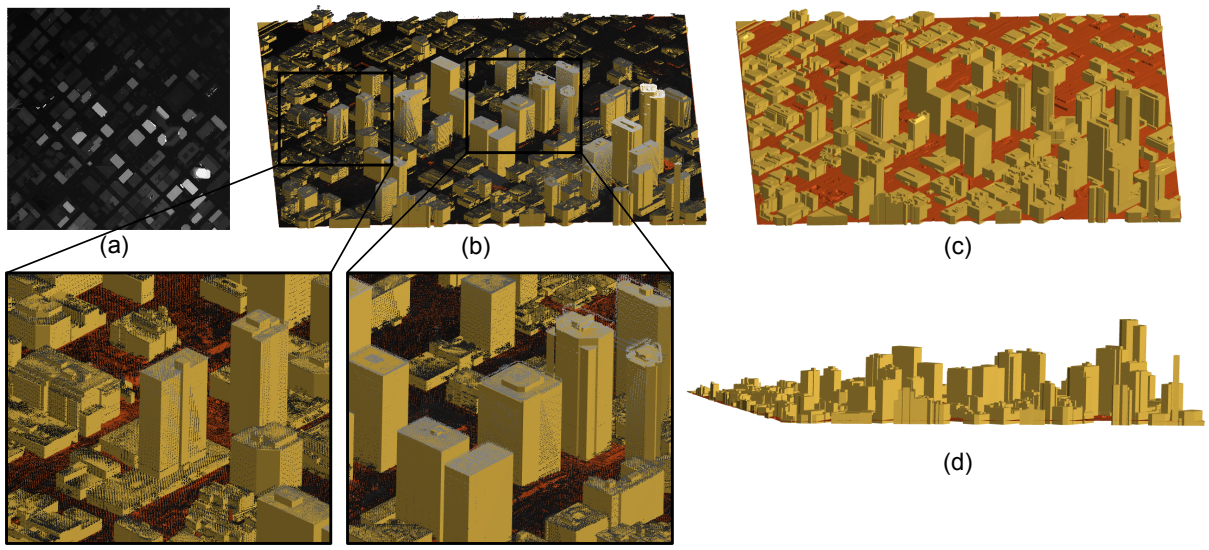

Figure 9: Building modeling from a $1km \times 1km$ piece of Denver city: (a) input LiDAR data; (b) reconstructed building models overlaying with initial point cloud; (c)(d) modeling result viewed from different perspectives.

Possible future work lies in the following directions. First, since we do not have ground-truth building models to our data set, it is a difficult task for us to analyze our results quantitatively. However, we notice the lack of quantitative error analysis is a common problem in this area. Hence, it may be necessary to investigate approaches for measuring the error of reconstructing results without ground-truth.

Second, since LiDAR data is normally a huge data set containing several millions of points, it is a common solution to divide the initial data set into several different pieces, each of which could be loaded into memory and processed at a time. Our implementation also follows this solution, however, we find that additional processing is needed at the area boundaries. On the other hand, we notice that our algorithm runs locally, *i.e.* any operation in our algorithm is a local operation which only involves points in a small neighborhood. Therefore it is suitable to be made into an out-of-core implementation, inspired from the streaming techniques[7]. In addition, in our current implementation, we assume that the principal directions remain the same in the same local patch - this assumption may not hold for a large urban area. Hence, it would be an interesting topic to study a global principal direction model for the whole urban area.

9. ACKNOWLEDGEMENT

We would like to thank the anonymous reviewers for their valuable comments. We gratefully acknowledge the sources of our data sets: Airborne 1 Corp. for Oakland, Sanborn Corp. for Denver, and Chevron Corp. for the industrial site. We thank Suya You and Yuan Li for helpful discussion. This work is partially supported by a Provost's Fellowship from USC.

10. REFERENCES

[1] A. Alharthy and J. Bethel. Heuristic filtering and 3d feature extraction from lidar data. In *ISPRS Commission III, Symposium 2002*, pages 29–35, 2002.

[2] A. Elaksher and J. Bethel. Reconstructing 3d buildings from lidar data. In *ISPRS Commission III, Symposium 2002*, pages 102–107, 2002.

[3] 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*, 24(6):381–95, 1981.

[4] C. Früh and A. Zakhor. Constructing 3d city models by merging aerial and ground views. *IEEE Computer Graphics and Applications*, 23(6):52–61, 2003.

[5] 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*, pages 74–78, 2001.

[6] J. Hu, S. You, and U. Neumann. Integrating lidar, aerial image and ground images for complete urban building modeling. In *3DPVT'06*, pages 184–191, 2006.

[7] M. Isenburg, Y. Liu, J. Shewchuk, J. Snoeyink, and T. Thirion. Generating raster dem from mass points via tin streaming. In *Proceedings*, Geographic Information Science, pages 186–98, 2006.

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

[9] S. K. Lodha, D. M. Fitzpatrick, and D. P. Helmbold. Aerial lidar data classification using adaboost. In *3DIM'07*, pages 413–20, 2007.

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

[11] G. Priestnall, J. Jaafar, and A. Duncan. Extracting urban features from lidar digital surface models. *Computers, Environment and Urban Systems*, 24(2):65–78, 2000.

[12] F. Rottensteiner. Automatic generation of high-quality building models from lidar data. *IEEE Computer Graphics and Applications*, 23(6):42–50, 2003.

[13] J. Secord and A. Zakhor. Tree detection in urban regions using aerial lidar and image data. *IEEE Geoscience and Remote Sensing Letters*, 4(2):196–200, 2007.

[14] V. Verma, R. Kumar, and S. Hsu. 3d building detection and modeling from aerial lidar data. In *CVPR 2006*, volume 2, pages 2213–2220, 2006.

[15] O. Wang, S. K. Lodha, and D. P. Helmbold. A bayesian approach to building footprint extraction from aerial lidar data. In *3DPVT'06*, pages 192–199, 2006.

[16] S. You, J. Hu, U. Neumann, and P. Fox. Urban site modeling from lidar. In *Proceedings, Part III*, volume 3 of *ICCSA 2003*, pages 579–88, 2003.

[17] Q.-Y. Zhou, T. Ju, and S.-M. Hu. Topology repair of solid models using skeletons. *IEEE Trans. Vis. Comput. Graph.*, 13(4):675–85, 2007.