Lightweight 3D Modeling of Urban Buildings From Range Data

Weihong Li
Department of Computer Science
Graduate Center, City University of New York
New York, USA
wli@gc.cuny.edu

George Wolberg
Department of Computer Science
City College of New York
New York, USA
wolberg@cs.ccny.cuny.edu

Siavash Zokai
Brainstorm Technology LLC
New York, USA
zokai@brainstorm.com

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

***Keywords* - 3D Modeling, point cloud, laser scanning, range data, segmentation, Google SketchUp, vectorization**

I. Introduction

The 3D modeling of urban buildings is an area of active research with increasing attention drawn from the computer graphics and computer vision communities. Current state-of-the-art algorithms include procedural modeling, 3D laser scanning, and image-based approaches. In addition, conventional modeling tools are commonly used for this purpose. The most accurate input source for modeling *existing* buildings, though, remains laser range scanners. They provide high geometric detail by collecting range data from hundreds of meters away with an accuracy on the order of a few millimeters. This fidelity is appropriate for construction, architecture, cultural heritage, and forensics applications. Unfortunately, laser range scanning can produce an overwhelming amount of data, which poses great challenges to visualization software that require lightweight 3D models for interactive use. Polygonal data generated from range scans are therefore too dense for use in web-based applications such as Google Earth and Microsoft Virtual Earth. These applications work best with lightweight models consisting of only hundreds of polygons.

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

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

II. Related Work

In an attempt to steer clear of tedious and expensive hand-made models, procedural modeling of buildings in [1] has been proposed. By using an effective description language, buildings and streets of a virtual city can be generated automatically. The strength of this approach is that the description language can generate a huge number of buildings and streets quickly and beautifully. This is particularly useful for gaming and other computer graphics applications. However, since the parameters used to generate the buildings are randomly generated, the city generated with these buildings and streets is a virtual one. This approach is not useful for attempting to model an *existing* building. To do so, one has to manually specify the parameters of the building, which is very cumbersome. Our goal is to automatically infer the contours and parameters of an existing building directly from dense range data.

Reconstruction of 3D models from range data has been addressed in [2] with applications in numerous research areas, including computer-aided design (CAD), computer vision, architectural modeling, and medical image processing. The authors in [3] use a histogram of height data to detect floors and ceilings for creating accurate floor plan models of building interiors. In [4], the authors proposed a 3D building reconstruction from a 2D floorplan image. With the help of a

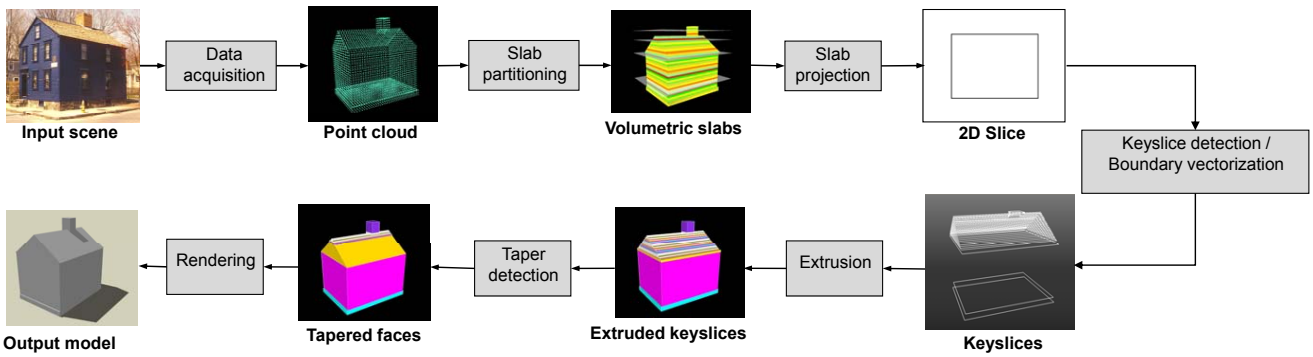


Fig. 1. Overview of the proposed approach.

2D floorplan image, both the interior and exterior of a building can be reconstructed accordingly. A survey on methods for generating 3D building models from architectural floor plans is given in [5]. However, reliance on 2D floor plans makes this approach too limiting for most applications, including our project.

In [6], known manufacturing features were used to infer the 3D structure of mechanical parts. Their method benefits from the domain knowledge that most of the mechanical parts consist of predefined structures, such as holes, bosses, and grooves. Our work is partially motivated by this idea since it also incorporates *a priori* knowledge about the construction of urban buildings for further inference. However, their method is based on predefined simple geometry structures and the assumption that the input 3D data has no holes. This hinders their approach for applications with incomplete data.

Multimodal data fusion is another approach for large-scale urban environment modeling. In [7], both air and ground data are fused, including laser scans, camera images, and aerial images. The LIDAR scans are used to create the models and the camera images are used for texture mapping. Citing the cumbersome and expensive use of laser scanners, the researchers in [8] propose an approach that relies solely on passive sensors (cameras) mounted on a moving vehicle. Dense 3D point cloud measurements are derived using their multiview stereo module based on multiple plane sweeping directions. In an attempt to compress the voluminous data produced in the method of [8], Xiao et al. [9] introduced an alternate approach for modeling facades along a street using prior knowledge about the buildings. They achieve geometry compression and deliver a clean approximation of the facades by applying a combination of plane fitting and window detection. Their method, however, relies on limited assumptions about the planarity of the buildings. The method introduced in this paper, however, places no such limitations. We can handle facades of any shape that exploit extrusion and tapering operations. Toshev *et al.* proposed a grammar based method for detecting and parsing buildings from unorganized street-level point clouds [10] . Despite its efficiency in modeling, the results could not generate enough level of details for the buildings.

In [11], the authors divide the point cloud into slices from which circles can be fitted to extract pillars of buildings. Although we also use slices of point cloud data, our work generalizes to arbitrary profiles and multiple sweeping directions. In related work, [12] uses 2D slices of point cloud data to segment simple primitives such as lines and arcs. Clustering these 2D primitives from slice to slice is used to fit planes and cylinders to the 3D data. This limits the work to recovering only simple planar and cylindrical fragments of the complete model. The remaining segments are left as point cloud data.

III. Overview

We propose an efficient way to reconstruct 3D models from range data by partitioning the data into thin cross-sectional volumetric slabs. For each slab, all range data in that slab is projected onto a 2D cross-sectional contour slice. Producing this array of slices permits us to avoid costly computation directly on 3D data. A similarity measure is used to cluster the sliced images together into *keyslices*. This term is analogous to the use of "keyframes" in computer animation, which denote important snapshots in the animation sequence from which intermediate results can be derived. In essence, each keyframe is a slice in the spatiotemporal volume of an animation. Similarly, each keyslice is a 2D image which contains a *transitional* cross-section of the building, encapsulating major contours in the facade. The model is then generated by applying basic extrusion and tapering operations from one keyslice to the next. This produces a lightweight representation consisting of only a few hundred polygons.

Fig. 1 depicts the basic concept of our algorithm. We begin with the acquisition of a dense 3D point cloud C of a building. C is then partitioned into a nonoverlapping set of volumetric slabs. Each slab S is associated with one projection plane P, sitting at the base of S. The purpose of partitioning C is to establish a set of cross-sections, or contour slices. By examining the changes among these slices, we can identify the prominent slices, or *keyslices*, as well as the necessary

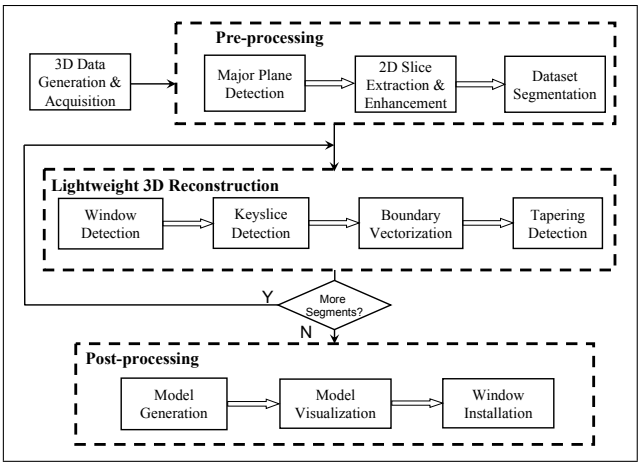


Fig. 2. The flow diagram of the system.

extrusion and tapering operations that must apply to them to generate the model. By casting this 3D modeling task into a series of 2D operations, we reduce the dimension of the problem to achieve a significant savings in computational complexity.

The modular flow diagram for our system is shown in Fig. 2. The whole system consists of three stages of computation. In the first pre-processing stage, 2D slices are extracted from the heavy 3D range data and are enhanced by noise removal and the filling of missing data. The segmentation module is then carried out to divide the complicated 3D dataset into simpler segments. The second stage is iteratively applied to each segment, including window detection, keyslice detection, boundary vectorization, and tapering detection. The final stage reconstructs each segment and assembles them into a whole model.

IV. Preprocessing the Range Data

The input to our system is range data assembled as a 3D point cloud. We have registered the voluminous 3D data acquired from multiple scans of buildings using the algorithm in [13]. That same algorithm is also responsible for extracting the major axes of the building in order to align it to the axes of the world coordinate system. This is necessary to properly infer the keyslices. Fig. 3(b) displays a properly aligned, *registered* 3D point cloud.

Fig. 3. (a) Input scene. (b) 3D point cloud of scene assembled by registering 14 scans, each having one million points.

In addition to real data, we also generated some synthetic datasets for experiments. These synthetic datasets were sampled from 3D building models containing 3D faces and their normals, which were downloaded from Google 3D warehouse. The first two rows of Fig. 9 show two such models and their 3D point clouds in (a) and (b), respectively.

A. Major Plane Detection

The input to our system consists of unorganized 3D point clouds. We solve for the major planes, whose normals are the sweeping directions from which to extract 2D slices. These slices are the starting point for segmentation and window detection. We used moving least squares (MLS) for deriving a smooth plane from a set of neighboring data points in space for normal computation. After the normal is computed for each 3D point, the Hough transform was used to identify the major plane normals based on voting [14]. These normals will determine the orientation of the cross-sections that sweep through the PCD.

B. Extraction of 2D Slices

We consider the PCD as a large array of 3D points to be sliced into equispaced parallel volumetric slabs. All 3D points within each slab are projected onto a projection plane, or slice, at the base of the slab. Fig. 4(a) shows the 3D point cloud in Fig. 3(b) partitioned into 50 slabs. The projected 3D points in each slab form cross-sectional contour slices. Fig. 5 depicts four such slices, associated with the four displayed projection planes of Fig. 4(a).

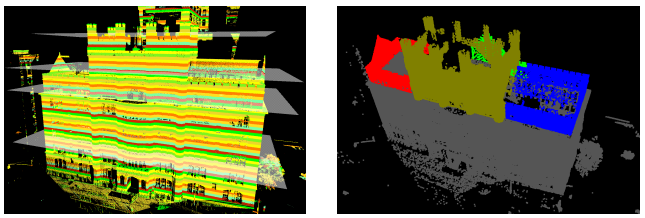


Fig. 4. (a) The 3D point cloud of Fig. 3(b) partitioned into uniform volumetric slabs. The 3D points in each slab are projected onto a projection plane to form cross-sectional slices. Four such planes are shown; (b) Segmentation result of Fig. 3(b).

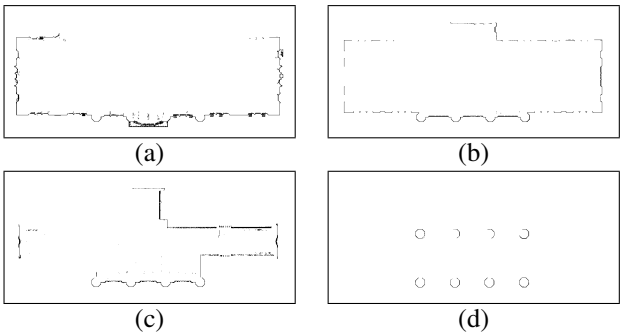


Fig. 5. The set of slices corresponding to the four projection planes in Fig. 4(a).

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

$$[\, x^{2D},\ y^{2D}\,]^T = \omega \cdot [\, x_i^{3D} - X_{MIN},\ z_i^{3D} - Z_{MIN}\,]^T \quad (1)$$

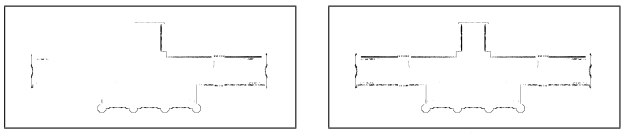
Note that $\omega = W/(X_{MAX} - X_{MIN})$, and that the $[X_{MIN}, X_{MAX}]$ and $[Z_{MIN}, Z_{MAX}]$ pairs define the 3D bounding box, which can be obtained through user input and can be used to clip away noise data. Fig. 5(a)-(d) show some examples of the 2D slices, where noise and incomplete data are observed. We repeatedly sweep through the volume to extract parallel volumetric slabs along the directions of the major plane normals computed in Sec. IV-A. Fig. 8, for example, depicts slices extracted from the side view.

C. Filling Missing Data

Extracted slices often have missing data due to occlusion or other visibility issues. Fortunately, most urban buildings have symmetry that we can exploit to fill these gaps. Symmetry computation on 3D data is expensive [15], so we conduct this computation on the 2D image slices. Since the 3D data has been already rectified during the registration process and projected onto 2D slices [13], symmetry computation now only needs 2D translation. Let $P(x, y)$ be a point on the original image I and $P'(x', y')$ be the reflected point of P with respect to a symmetry line L. The symmetry computation equation for L is as follows:

$$L = \arg\min_{x,y} \sum d_{x,y}(P', I) \quad (2)$$

where $d_{x,y}(P', I)$ is the distance between the self-reflected point P' and its nearest data point in image I. The reflected point P' of the original point P is computed with respect to a line along either the $x-$ or $y-$ axis. Therefore, the symmetry line L is obtained as the line with minimum summation error over the reflected data points. Fig. 6(a) and Fig. 6(b) depict the original input with missing data, and the output after gap filling using symmetry computation, respectively.



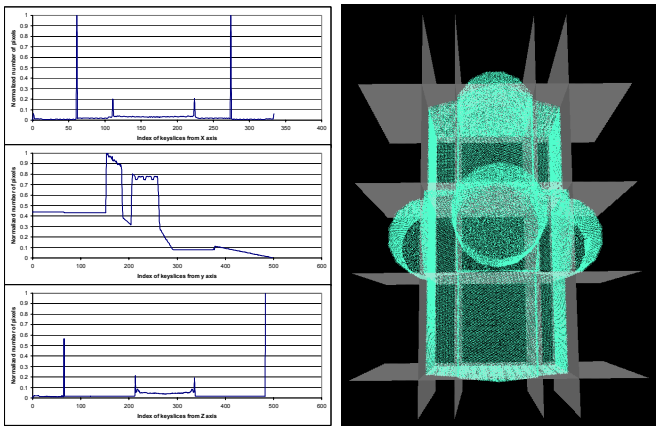
(a) (b)

Fig. 6. Symmetry-based gap filling. (a) Original 2D slice image and (b) output image after gap filling.

D. Dataset Segmentation

Modeling the PCD of a building as a whole structure simultaneously is complicated due to the natural complexity of buildings. To simplify this problem, we utilize the divide and conquer strategy to segment the whole PCD into simpler parts. Each of these parts can be easily represented by extrusion/taper operations. 3D PCD segmentation is generally performed using region based methods, although their computational cost may be high. We propose an efficient segmentation approach based on the observation that different parts of a building are usually separated by walls, ledges, and other architectural elements. These *"separators"* provide segmentation clues.

To detect these separators, we examine the data point distribution of 2D slices extracted from all major planes, as depicted in Fig. 7. We identify the separators as the indices of the slices that coincide with the local maxima and inflection points in the data point distribution. Once we obtain the separators, we can segment the original dataset based on the intersections of these separator planes. As Fig. 4(b) shows, there are a total of five regions that are identified for the PCD shown in Fig. 3(b), where each segment is labeled with a different color.



(a) (b)

Fig. 7. The distribution of data points in cross-sectional slices along X, Y, and Z axes for the model shown in the second row of Fig. 9(b). The distribution is defined as the sum of pixels of each cross-sectional slice. (b) Top view of the point cloud with cutting planes inserted at the extremas of the point distributions shown in (a).

V. Lightweight 3D Reconstruction

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

A. Window Detection

Windows and doors are important features for buildings to be modeled. Moreover, accurate computation of the extrusion structures depends on this information. Without knowing the locations of the windows, extra keyslices may be computed hence leading to excessive extrusion operations on windows. Our window detection algorithm is based on the work presented in [16]. Following this, we can generate mask images based on the boundaries of the detected windows to discard the 3D points in the window regions for keyslice computation.

During the window and door detection step, we retain the boundary, position, and depth of the windows. After the facade is extruded from one keyslice to the next (see Sec. V-B), we project the boundary of the window/door onto the extruded facade. This projected boundary is pushed into the facade in the opposite direction of the face normal up to the recovered depth of the window. This action significantly reduces the number of polygons from the model.

B. Keyslice Detection

The 2D image slices of an extruded region are similar to each other. Thus, to detect the keyslices that delimit extruded regions one only needs to compute the similarity between adjacent slices. We adopted a light-weighted global and efficient key image detection approach based on distance function similar to Hausdorff distance as the similarity measure. Let $P_r(x_r, y_r)$ and $P_i(x_i, y_i)$ be a data point in a reference image and a new observed image I, respectively. The distance function of image I to reference image I_r is defined as:

$$d_H(I, I_r) = \sum_{i=0}^{N} d_{min}(P_i, I_r) \tag{3}$$

where $d_{min}(P_i, I_r)$ is the minimum distance from P_i in image I to the reference image I_r. Alternatively, we can also define the distance, $d_H(I_r, I)$, from I_r to a new observed image I, using Eq. (3). These two distances are usually not equal to each other. As a rule of thumb, one can choose $d_{HD} = \text{MAX}\{d_H(I, I_r), d_H(I_r, I)\}$ as the distance. To compute the keyslices, a threshold τ_d is used for the distance d_{HD}. If $d_{HD} < \tau_d$, the two images I and I_r are considered similar to each other. Otherwise, a keyslice image is found and I_r is updated with I, the new keyslice image.

The accuracy of the keyslices detected by using the distance function is closely tied to threshold τ_d. Small τ_d leads to more accurate models and will require more time and space to compute and store the result. When the threshold τ_d is relatively large, potential keyslices which contain salient structure may be missed. Therefore, there is a trade-off between model accuracy and time-space efficiency. To address this problem, the curvature information is computed as a complementary criteria for keyslice detection.

This idea is based on the observation that the keyslices are generally located at large curvature changes along 2D slices extracted in the orthogonal direction (e.g., side view), as shown in Fig. 8. Therefore, instead of computing the difference between two images directly, we compute the curvature of orthogonal 2D slices, map the positions of curvature extrema back to cross-sections in the original set of volumetric slabs, and mark these cross-sections as keyslices.

To compute the curvature, we first apply the slice extraction algorithm described in Sec. IV-B to obtain a series of 2D cross-sectional images in the orthogonal direction. We then apply the ball-pivoting algorithm described in Sec. V-C to vectorize the boundary for each sliced image. We locate those curvatures that appear in most of the sliced images as the places where

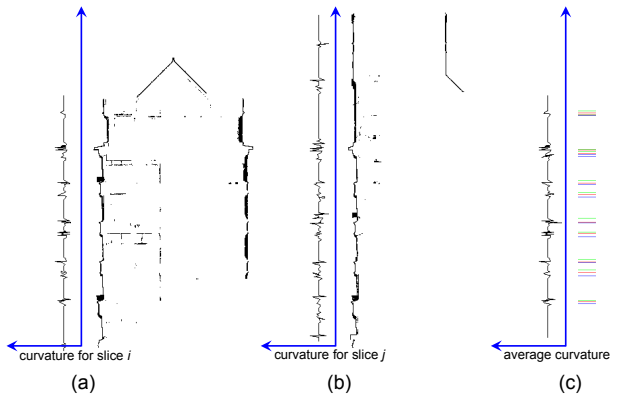


Fig. 8. Curvature-based key slice detection. (a,b) Two 2D sliced images from the orthogonal direction (side view). A plot of the curvature of each slice is displayed alongside. (c) A plot of the average curvatures detected over all of the sliced images along the orthogonal direction. Thresholding the curvature yields the location of keyslices, displayed alongside the plot. Red lines along the average curvature plot indicate local maxima. Blue and green lines, respectively, indicate zero-crossings in the average curvature. This delineates the keyslice positions.

keyslices are found, as shown in Fig. 8(c). The combination of similarity measure and curvature inference ensures that the salient structures of a building will be preserved.

C. Boundary Vectorization

After the keyslices are detected, K keyslices will be identified from a total of A image slices. Depending on the threshold τ_d, K is usually about one to two orders of magnitude smaller than A, e.g., K/A is 0.06 when τ_d = 4.0 for the example in Fig. 3(b). To generate the 3D model, these keyslice images need to be vectorized to represent the contours of the building facade. The Douglas-Peucker algorithm attempts to connect all of the existing points to form a polygon [17]. Although the implementation of this approach is very efficient with the improvement described in [18], this method cannot handle the case where spurious interior points are present, which contributes to outlier data. To tackle this issue, we adapted the ball-pivoting algorithm (BPA) [19] from its original use on 3D PCD to use on 2D keyslice images where it produces vectorized boundaries. The key parameter for the BPA algorithm to work successfully is to find the right size of the ball for pivoting. We implemented a coarse-to-fine adaptive BPA algorithm to solve this problem.

D. Extrusion and Taper Detection

After the keyslices are detected and vectorized, the contours of the set of K keyslices are used to represent the building based on the extrusion operation. That is, the space between each adjacent pair of keyslices is filled by extruding one keyslice to the next. By modeling a building using extrusion operations on the keyslices, we significantly reduce the number of polygons for urban buildings.

In addition to the extrusion operation, we can further improve the model and reduce the model size based on the

observation that part of the keyslice images may belong to the same tapering structure. The difficulty in inferring tapering structures is tied to the complexity of a building structure itself. Fortunately, the dataset segmentation module introduced in Sec. IV-D has segmented the complicated structures into simpler parts. Although the majority of building tapering structures are *linear tapering*, such as *tapering to point* (TTP), a cone shape geometry, and *tapering to line* (TTL), a wedge shape, it could also be a complicated *non-linear tapering*, such as a dome shape. Furthermore, a structure may look like a tapering structure, but it is actually not a real one. For example, a series of small extruded structures form a tapering-like shape.

We introduced a two-step workflow to accomplish the above goal. The first step is to locate the potential tapering keyslices and infer the structure by making an assumption that the underlying tapering structure is either a TTP or a TTL. A verification step is conducted to check the correctness of the inferred shape by measuring the error between the model and the corresponding 3D PCD. If the error is small, the inferred shape is confirmed and the underlying keyslices are not rendered. Otherwise, we can choose to model this special structure by fitting a triangular mesh to the underlying 3D point cloud to produce a polygonal model. This algorithm cannot model a sphere or a dome because such structure cannot be linearly interpolated as a taper operation.

VI. Experimental Results

To generate the final 3D model, the control points of the 2D contours can be transformed back into 3D world coordinate system using the reverse matrix T in equation Eq. (1). For each segment, we first exam whether it can be modeled by simple keyslices along any major plane. If so, the push-pull operation is applied to the keyslice contours to generate the extruded model. If a tapering structure is detected, we construct the faces based on the control points of the 2D base geometry polygon and their corresponding converged points to create the tapering model.

Fig. 9 shows the experimental results for both synthetic (the first two rows) and real datasets (the last three rows). Each row shows a reconstruction of a building. The snapshot of the original model or the image of the real building is shown in (a), followed by the snapshot of 3D PCD in (b). Columns (c)-(e) depict the segmentation result, the vectorized keyslices, and the snapshot of the reconstructed model, respectively.

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

$$E = \frac{1}{|X|} \sum_{x \in X} d(x, M) \tag{4}$$

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

Table I lists the relationship among the τ_d, errors, number of faces, and model size for the input data in Fig. 3(b). The unit for τ_d is in pixels and the unit for error is in meters. The size of the original point cloud for the 3D building is more than 700 MB. From the table, we can see that even for the most accurate model, the size is dramatically reduced compared with the original 3D PCD. This is a desirable property for web-based applications. These models were generated on a laptop PC with an Intel Core 2 T7200 CPU at 2.0 GHz with 2.0 GB RAM. Future work includes the optimization of the BPA vectorization module since it consumes approximately 70% of the computation time.

τ_d(pixel)	Error (m)	# of faces	Size (KB)	time (s)
64	.658 ± .158	1471	15	1977
32	.294 ± .103	3284	32	2353
16	.141 ± .058	8574	86	3008
8	.131 ± .074	13955	137	3696
4	.094 ± .068	27214	261	5391
2	.088 ± .036	31331	335	7586
1	.083 ± .041	32187	337	10927

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

A. Model Comparison

Although models generated by 3D BPA are of high resolution, they usually require excessive storage capacity. The model in Fig. 10, for example, needs almost 400 MB of storage, which prevents this solution from being applied to web-based applications. One way to improve matters is to apply some approximation/decimation technique to reduce the space required by these models.

The holes in the 3D BPA model in Fig. 10 are present in the original dataset. They are due to the fact that the laser never reflected back to the scanner after penetrating the glass windows. The 3D BPA method is deficient in filling these holes. We counter this problem by first applying a symmetry-based hole filling algorithm on the 2D slices to create enhanced slices that are processed by an adaptive 2D BPA method to fill gaps. Finally, an extrusion operation is applied to create a watertight 3D model.

Among all mesh reduction techniques, *qslim* is one of the most sophisticated and efficient algorithms [20]. We carried out a comparison between models generated by our proposed method and those approximated by *qslim*. The comparisons were conducted on models sharing the same number of faces. It is worth noting that *qslim* ran out of memory on the 3D model data generated by BPA in Fig. 10. In order to reduce the size of the model for *qslim* to work, we had to either downsample the 3D model generated by BPA or split it into sub-models which can be handled by *qslim*.

Fig. 11(a) and Fig. 11(c) respectively depict the models generated by *qslim* and our proposed method with approximately 2,000 faces each. Higher resolution models with roughly 32,000 faces each are shown in Fig. 11(b) and Fig. 11(d). Notice that the models approximated by *qslim* are inferior

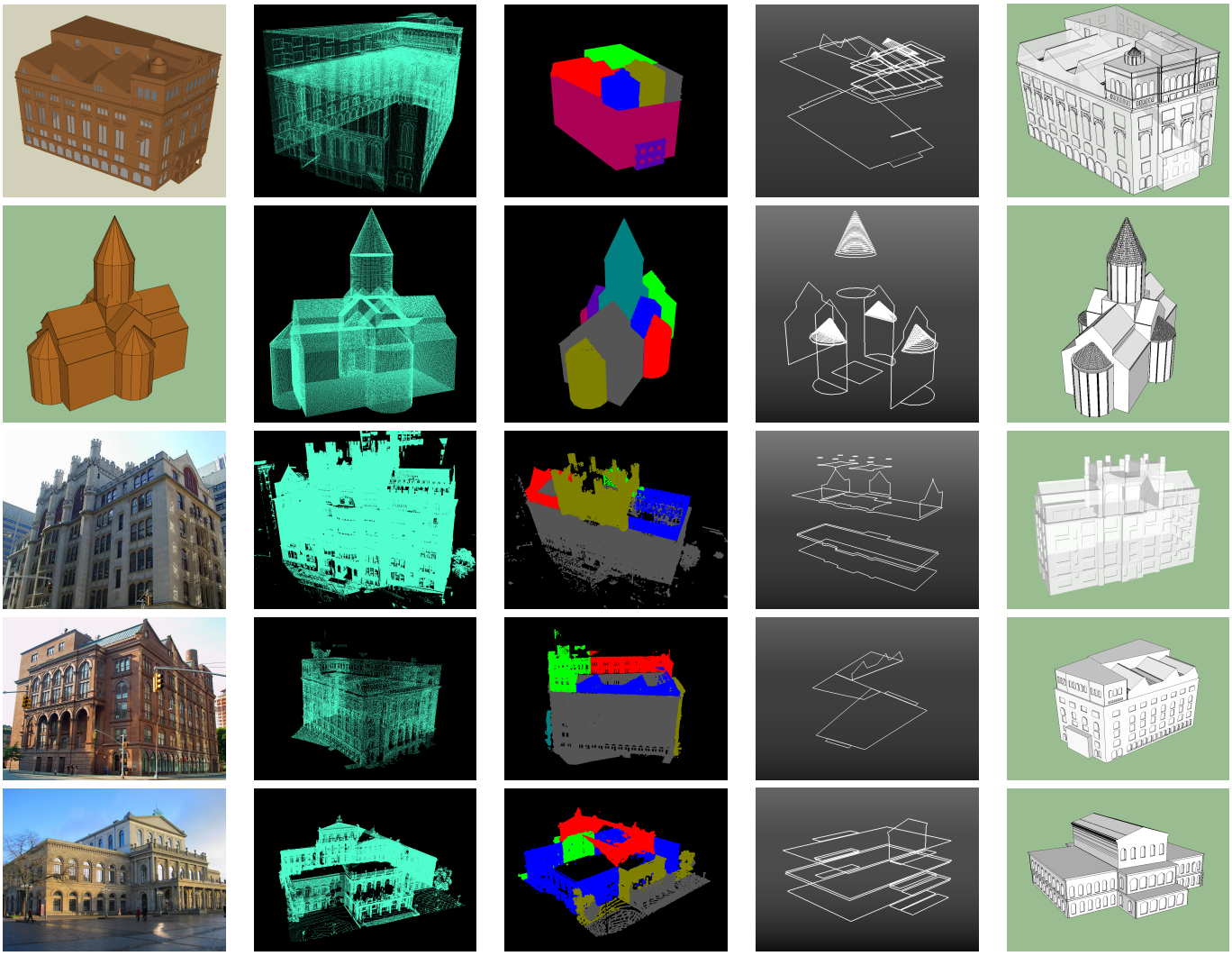


Fig. 9. Experimental results. (a) original model / picture, (b) 3D point cloud of (a), (c) segmentation, (d) keyslices, and (e) reconstructed model with windows. The data of the Opernhaus Hannover in the bottom row is provided courtesy of the Institute of Cartography and Geoinformatics, University of Hannover.

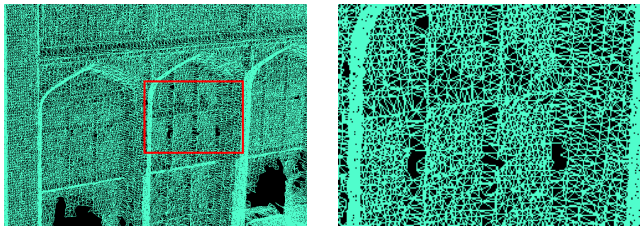


Fig. 10. Dense triangulated BPA mesh cropped from Fig. 3(b).

since they do not preserve the sharpness of the original model and are replete with holes. Our symmetry detector and extrusion operation guarantees no holes.

There have been recent attempts at preserving the sharp features of buildings during the decimation process [21]. Nevertheless, the output model remains a triangulated mesh. The benefit of our approach is that we represent the model with a set of polygons and an associated grammar of extrusion/taper operations. This furnishes a powerful mechanism by which to infer a procedural model through 3D point clouds.

VII. Conclusion

This paper has presented an efficient algorithm for lightweight 3D modeling of urban buildings from range data. Our work is based on the observation that buildings can be viewed as the combination of two basic operations: extrusion and taper. The range data is partitioned into volumetric slabs whose 3D points are projected onto a series of uniform cross-sectional planes. The points in those planes are vectorized using an adaptive BPA algorithm to form a set of polygonal contour slices. Prominent keyslices are extracted from this set. Applying extrusion to these keyslices forms lightweight 3D models. We achieve further geometry compression by detecting a series of slices that coincide with a linear tapering operation.

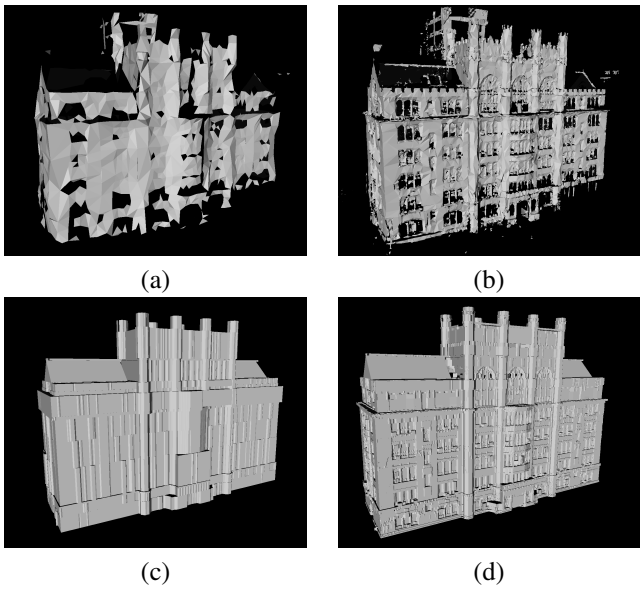


Fig. 11. Models generated by *qslim* with (a) 2,000 and (b) 32,000 faces and by our approach with (c) 2,000 and (d) 32,000 faces.

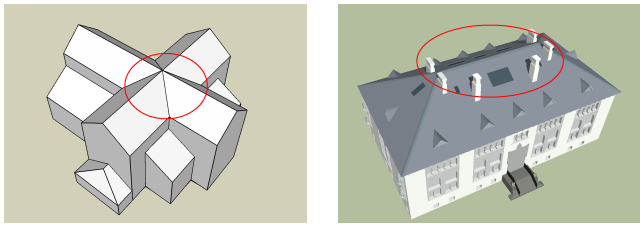


Fig. 12. Examples of failed cases: (a) intersection of two extruded structures. (b) intersection of a tapered structure with an extruded structure.

One limitation of the proposed method is that it could not handle the intersection of two structures from different directions. For example, Fig. 12(a) shows a case where two extruded structures intersect and Fig. 12(b) shows one with intersection of a tapered structure and an extruded structure. Therefore, one future direction of our work is to expand our system to handle them. Additional future work is to investigate the modeling of the "follow-me" geometry structure. This is a more complicated geometry structure featured in Google SketchUp that exists when the model can be reconstructed by moving a cross-sectional unit along a curve trajectory. Finally, we will optimize the performance of the BPA vectorization module, which consumes the bulk of the computation time.

REFERENCES

[1] P. Mueller, P. Wonka, S. Haegler, A. Ulmer, and L. V. Gool, "Procedural modeling of buildings," *ACM Transactions on Graphics (TOG)*, pp. 614–623, 2006.

[2] R. B. Fisher, "Applying knowledge to reverse engineering problems," *Computer-Aided Design*, vol. 36, pp. 501–510, 2004.

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

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

[5] X. Yin, P. Wonka, and A. Razdan, "Generating 3D building models from architectural drawings: A survey," *IEEE Computer Graphics and Applications*, pp. 20–30, 2009.

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

[7] A. Zakhor and C. Frueh, "Automatic 3D modeling of cities with multimodal air and ground sensors," *Multimodal Surveillance: Sensors, Algorithms, and Systems, Artech House, Boston*, vol. Chapter 15, 2007.

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

[9] J. Xiao, T. Fang, P. Tan, P. Zhao, E. Ofek, and L. Quan, "Image-based facade modeling," *SIGGRAPH Asia*, 2008.

[10] A. Toshev, P. Mordohai, and B. Taskar, "Detecting and parsing architecture at city scale from range data," *IEEE Computer Vision and Pattern Recognition*, pp. 398–405, 2010.

[11] D. Luo and Y. Wang, "Rapid extracting pillars by slicing point clouds," *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XXXVII, Part B3b, 2008.

[12] P. R. Gonzalvez, D. G. Aguilera, and J. G. Lahoz, "From point cloud to surface: Modeling structures in laser scanner point clouds," *ISPRS Workshop on Laser Scanning*, pp. 338–344, 2007.

[13] I. Stamos, L. Liu, C. Chen, G. Wolberg, G. Yu, and S. Zokai, "Integrating automated range registration with multiview geometry for the photorealistic modeling of large-scale scenes," *International Journal of Computer Vision*, vol. 78, no. 2-3, pp. 237–260, 2008.

[14] G. Vosselman, B. Gorte, G. Sithole, and T. R. Levin, "Recognising structure in laser scanner point clouds," *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, pp. 33–38, 2004.

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

[16] S. Pu and G. Vosselman, "Extracting windows from terrestrial laser scanning," *ISPRS Workshop on Laser Scanning*, pp. 320–325, 2007.

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

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

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

[20] M. Garland and P. Heckbert, "Surface simplification using quadric error metrics," *ACM SIGGRAPH*, 1997.

[21] S. Möser, R. Wahl, and R. Klein, "Out-of-core topologically constrained simplification for city modeling from digital surface models," *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XXXVIII-5/W1, Feb. 2009.