Pacific Graphics 2021
E. Eisemann, K. Singh, and F.-L Zhang
(Guest Editors)

Volume 40 (2021), Number 7

Manhattan-world urban building reconstruction by fitting cubes

SUBMISSION 1048

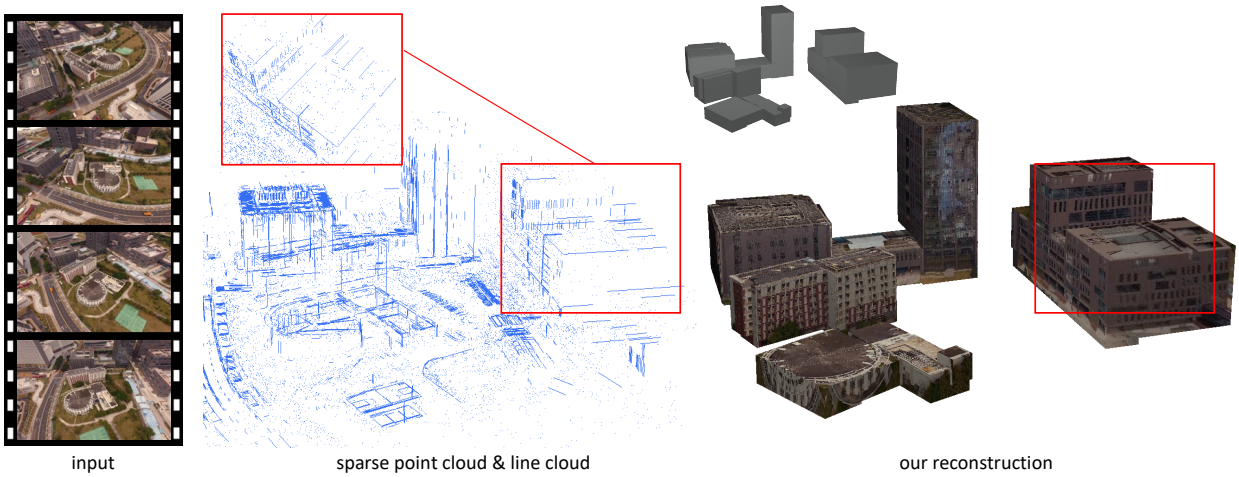

Figure 1: *This figure shows a reconstruction result of our approach on an urban scene. Using an image sequence as input, we firstly generate point cloud and line cloud of Manhattan-world buildings. Then our approach outputs lightweight models by extracting corners from line cloud and fitting cubes from point cloud. By fitting cubes from corners, our method is able to reconstruct buildings even when the back faces of buildings are severely lost (see red box). Experiments on various datasets show that our approach is quite competitive in both speed and quality.*

Abstract

Manhattan-world building is a kind of dominant scene in the urban area. Many existing methods reconstructing such scenes either are vulnerable to noisy and incomplete data or suffer from high computational complexity. In this paper, we present a novel approach to reconstruct lightweight Manhattan-world urban building models from images. Our key idea is to reconstruct buildings from the salient feature - corner. Given a set of urban building images, Structure-from-Motion and 3d line reconstruction operations are applied firstly to obtain camera poses, sparse point cloud, and line cloud. Then we use orthogonal planes detected from the line cloud to generate corners, which indicate a part of possible buildings. Start from corners, we fit cubes with nearby points by optimizing corner parameters and get cuboid represents of corresponding buildings. Finally, a registration step is performed on cuboid represents to generate more accurate models. The experiment result shows that our method can handle some nasty cases which contain noisy and incomplete data, meanwhile, output lightweight polygonal surface building model with a low time-consuming.

CCS Concepts

• ***Computing methodologies*** → *Shape modeling; Reconstruction;*

1. Introduction

Recently there has been an emerging interest in modeling urban buildings due to the increasing need for 3d city data in multiple application fields, e.g., virtual reality, GIS, BIM, and live map. As a kind of most common architecture, Manhattan-world building has been a vital modeling object in the urban buildings reconstruction.

In many applications, the fine details of the building models are not necessary because nowadays cities have a large scale while most buildings have a relatively simple geometry structure, especially for Manhattan-world buildings. Compare to highly detailed models that contain millions of triangles, lightweight models are more preferred in many applications owing to their superiorities in rendering, data transfer, and storage.

Typical photogrammetry reconstruction pipelines for lightweight models rely on Structure-from-Motion(SfM) [SSS06] and Multi-View-Stereo(MVS) [FP09] techniques to obtain the point cloud of urban scenes. Then surface reconstruction [KH13, KBH06, ABCO*03, KSO04, BMR*99] step is applied on the point cloud to get polygonal mesh models. Lastly, mesh simplification [LN21, ZG02, LT98, GH97] algorithm is performed on the complex models to get the final lightweight models. Though SfM is efficient even for large datasets, the MVS step consumes a large amount of time and requires dedicated graphic cards to accelerate. When it comes to large-scale urban datasets, MVS easily takes days even weeks to process. Besides, many expensive computations spent on details are abandoned and wasted on the mesh simplification step. A natural idea is skipping MVS step and reconstructing lightweight models directly from the point cloud generated by SfM step. However, these reconstructed point clouds are often pretty sparse due to the sparsity of feature points extracted from images, and this could be worse when facing man-made environments that contain many textured less surfaces and repetitive structures. Though reconstructing surface from the sparse point cloud directly is not practical, sparse point cloud still indicates the overall shapes and existence of buildings. In our work, we seek for an efficient solution to reconstruct lightweight models by using these information from sparse point clouds.

Recently, 3D line reconstruction techniques provide a new way to recover buildings from images. Sophisticated 3D line reconstructions extract line segments features that are common in the man-made world to generate 3D line models (a.k.a. line cloud) efficiently. The line clouds provide more structural information of buildings (e.g., sharp edges, planes) which are hard to extract from point clouds. But reconstructing polygonal surface models from the line cloud remains an open problem. In our work, we leverage the structural information from line clouds to help reconstruct buildings from point clouds.

Another core difficulty of urban buildings reconstruction is the low quality of obtained data. For example, urban images captured no matter from aerial or land usually contain a large number of occlusions due to the high density of urban buildings. Also, unlike in an indoor environment, laser scanners or cameras can not scan or capture a large-scale city at arbitrary angles to yield high-quality results. All these facts make obtained data usually incomplete, noisy and nonuniform. Reconstruction urban buildings from these low quality data is an ill-posed problem. Therefore recent reconstruction algorithms have been advanced from merely using low-level information (e.g., neighbor points) to combining with high-level information (e.g., semantic segmentation, prior assumption [FCSS09, LWN16], user interaction [NSZ*10], data-driven learning [XZZ*14]). In our work, we reconstruct urban buildings under Manhattan-world assumption and use corners (i.e., the sharp conjunctions that 3 planes of buildings converge) to locate and fit buildings. This is based on an observation that the corners of buildings are salient even when the building data is sparse or partly missing, and this is especially true for Manhattan-world buildings. Besides, we use images to perform an additional registration, thus yields more accurate results.

In this paper, we present a novel approach to reconstruct lightweight Manhattan-world building models from images. In brief, We firstly obtain a point cloud and a line cloud from images, then extract corners using planes detected from the line cloud. Next we fit cubes from corners with nearby points. The buildings are reconstructed as a group of cubes. The intuition of our method is based on a key observation that many Manhattan-world buildings are composed by an assembly of basic primitive shapes especially cube. The main advantage of our method is that it can recover cuboid-shaped buildings efficiently even when parts of their planes are missing. As a kind of robust feature, corners are easy to extract and salient to detect even in the incomplete and noisy data. The position of a potential building can be located quickly by extracting corners. By fitting cubes from corners, cuboid-shaped buildings can be reconstructed even when only there are three planes are presented. Besides, buildings can be encoded as cube represents or exported to shape approximated lightweight models, thus facilitating efficient storage and transmission.

The main technical contributions of this paper are as following:

1. a fast and lightweight framework for automatic reconstruction of Manhattan-world buildings from sparse and incomplete data.
2. a novel method to detect planes and corners of buildings from a line cloud.
3. a numerical optimization formulation for fitting cubes from a sparse point cloud.
4. a novel method to register cuboid represents of urban buildings with corresponding images.

2. Related work

3D Reconstruction from data collected from real world (e.g., scanned point cloud, images, videos) is a challenge problem of long standing, which has received considerable attention over decades. Over the years of development, a large body of techniques were proposed to achieve these tasks. In this section we briefly review some of the related works that focus on 3D reconstruction especially building reconstruction.

3D point and line reconstruction As a kind of dominated method in 3D reconstruction, typical 3D point reconstruction methods known as Structure-from-Motion (SfM) [SSS06] use distinctive feature points [Low04] extracted from image to estimate camera pose and generate point cloud by solving epipolar geometry. Due to limited number of feature points, generated point clouds are usually sparse. Multi-View Stereo(MVS) [FP09] generate dense point cloud by performing depth map estimation and depth fusion. Current the state of art reconstruction systems [SF16,SZPF16, Wu13, MMPM16] allow non-expert users to generate accurate point cloud from unordered image sequence. These tools have shown impressive results on richly textured surfaces, but they often

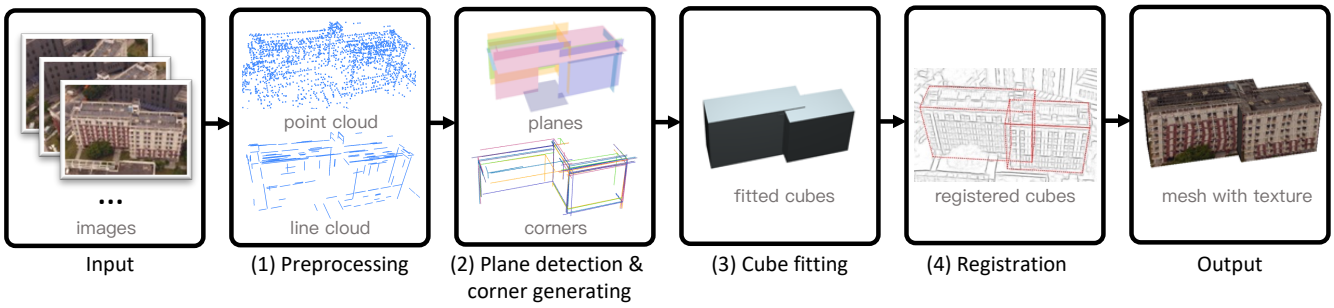

Figure 2: *An overview of our approach for lightweight Manhattan-world reconstruction.*

failed in reconstructing objects that lack texture details e.g., buildings. Another problem is that though SfM step is efficient, running MVS usually takes hours even days on personal computers due to its high computational complexity. Since 3D point reconstruction methods describe shapes using point clouds, a complex scene may easily contain millions of points which makes viewing and processing point cloud also a troublesome task.

Similar to 3D point reconstruction, 3D line reconstruction use line segments to recover 3D lines structures from images. One key problem is matching lines between different images. Some methods achieve it by line feature descriptors [BENV06], pointwise correspondences between lines [BS05] or coplanarity constraint [SMM17]. Some solve it by exploring lines global connectivity [JKTS10] or using weak matching [HMB17], thus bypass finding explicit correspondences. Some methods [SMM17, ZK14, KM14, SKD06, BS05] try to solve camera pose estimation and 3D line reconstruction at same time, others [HMB17, JKTS10] only solve 3D reconstruction based on camera pose that provided by SfM. In contrast to 3D point reconstruction, 3D line reconstruction is especially suitable for urban scene which contains large number of line structures. Though 3D line reconstruction leaves impressive result on man-made scenes, further reconstruction for mesh model is still a problem. In our work, we use line cloud to detect building planes and point cloud to fit cubes.

Surface reconstruction from point and line cloud The state of the art 3D reconstruction methods can generate point cloud and line cloud robustly. However point cloud and line cloud are not suitable for many practical uses thus a surface reconstruction step is needed to generate polygonal mesh. Surface reconstruction from point cloud has been extensively studied. Some methods developed in early stage are mainly based on combinatorial structures, such as Delaunay triangulations [KSO04, BMR*99]. Other methods [KH13, KBH06, ABCO*03] try to reconstruct an approximating implicit surface under surface smoothness assumption. However obtained point clouds are usually incomplete and noisy and aforementioned methods may lead to horrible result. Therefore methods that reconstruct using prior knowledge in a learning [XZZ*14] or data driven way [KMYG12, SFCH12] emerged recently. Besides many methods often yield jagged result on noisy data hence some methods [NW17, BdLGM14] attempt to reconstruct a lightweight model instead. Surprisingly, surface reconstruction from line cloud has little been explored. Sugiura et al [STO15] propose a method to efficiently reconstruct 3D surface as triangular meshes by integrating line cloud with the point clouds. Bay et al [BENV06] use line cloud reconstructed from 2 poorly-textured, uncalibrated images to get planar indoor scene. Recently Langlois et al [LBM19] present a pipeline that reconstruct watertight piecewise-planar model only using line cloud.

Manhattan-world reconstruction The aforementioned general reconstruction techniques do not usually yield desirable results on urban buildings. Hence many efforts have been made to explore dedicated algorithms to reconstruct facades, buildings, and architectures. We refer the reader to the survey by Musialski et al [MWA*13] for an overview of architecture reconstruction and here we only focus on previous works most closely related to ours, i.e., Manhattan-world reconstruction. The Manhattan-world assumption was first proposed by Coughlan and Yuille [CY99] in their work that estimates the viewer orientation from a single image. Matei et al [MSS*08] segment massive aerial LiDAR point cloud under Manhattan-world assumption. Venegas et al [VAB10] reconstruct buildings by extract Manhattan-worlds grammars from aerial images and generating corresponding models. Li et al [LNL16, LWN16] propose fully automatic approaches for reconstructing Manhattan-world buildings from point clouds by partitioning space and selecting boxes that fit point cloud best. The Manhattan-world assumption is also useful in indoor scenes reconstruction. Lee et al [LHK09] propose a framework to recover indoor scene structure from a single image. Furukawa et al [FCSS09] and Ikehata et al [IYF15] reconstruct indoor scenes by fitting planes with MVS point clouds in orthogonal directions. Li et al [LWC*11] and Aron et al [MMBM15] refine primitive extraction results by discovering regularized relations. Recently, many deep-learning-based methods are proposed to reconstruct Manhattan-world structures like wireframes [HWZ*18, ZQZ*19] and cuboids [XRT12]. In our work, we focus on reconstructing urban buildings under Manhattan-world assumption combining the point cloud, line cloud and images.

3. Methodology

The goal of our work is to reconstruct lightweight Manhattan-world urban building models from images. Our method takes an image sequence of urban buildings as input and outputs polygonal surface models. Our method has the following 4 main steps: preprocess-

ing, plane detection and corner generating, cube fitting, and cube registration. See Fig. 2 for an overview.

3.1. Preprocessing

Firstly, we input images into SfM system COLMAP to retrieve camera pose and generate a sparse point cloud $\mathcal{P}_0$. Then camera pose and images are inputted into the Line3D++ system to generate a line cloud $\mathcal{L}_0$.

Line clustering The line cloud generated by Line3D++ is pretty noisy and broken due to the estimation errors and scattered 2D line segments detected by the line segment detector [VGJMR12]. Using raw line cloud $\mathcal{L}_0$ to detect planes may lead to the generation of many virtual planes and repetitive detections of the same plane with a displacement, thus introduces large errors and unnecessary computation. We perform a line clustering before plane detection to get less noisy and more consistent line cloud $\mathcal{L}$.

Sample points Since the point cloud is relatively sparse, we sample points from clustered line cloud $\mathcal{L}$ to generate more points and merge them with point cloud $\mathcal{P}_0$. We use notation $\mathcal{P}$ to denote this new point cloud. One good property of Line3D++ is that it shares the same coordinate with COLMAP, so no additional registration step is required when merging point cloud.

3.2. Plane detection and corner generating

In this section, we detect planes from line cloud $\mathcal{L}$ and generate corners from detected planes. One might consider obtaining a corner by detecting 2 or 3 mutually perpendicular lines of which endpoints are close. However, the absence of any line that forms a corner will lead to the corner detection failure. So we use a somewhat more robust method to generate corners – generate corners by planes intersection. Compared to a line, a plane is less likely to be utterly missing since a plane is supported by multiple lines. Besides, some cuboid-shaped buildings may not have a salient orthogonal corner, e.g., round corner. In this case, generating corners from planes still works.

Plane detection Methods that extract planes from a point cloud usually rely on the estimated point normals [SWK07]. However, the point cloud $\mathcal{P}$ is too sparse and nonuniform to perform a reliable normal estimation, which often leads to poor detection results. Therefore we detect planes using the line cloud $\mathcal{L}$ instead, since line segments provide more structural information of urban scenes and facilitate robust plane detections. We adopt the idea from Langlois et al [LBM19] and Bay et al [BENV06] that one line can support two planes at most. Lines that support one single plane are textural lines, which are usually located within a plane. Lines that support two planes are structural lines, which are at the intersection edge of two planes. Unlike RANSAC method [LBM19] that extracts planes by picking lines randomly, we only extract planes that are supported by mutually parallel or perpendicular lines. This is derived from an important observation that most lines that form Manhattan-world planes are regularized. This constraint may be strict but very efficient and practical in terms of urban scenes. We use notations $\Lambda(P)$ denotes the line set supporting plane P and $\Pi(l)$ denotes the plane set supported by line l. Fig. 3 illustrates the main steps of plane detection. Initially, detected plane set $\mathcal{S}$ and supported plane set $\Pi(l)$ for each $l \in \mathcal{L}$ are set to $\emptyset$. For each l_i in $\mathcal{L}$, we iterate each line l_j that satisfy $|\Pi(l_j)| < 2$ and is coplanar with l_i. If l_j is roughly parallel or perpendicular to l_i and the plane l_j already supports is not parallel to the plane formed by l_i and l_j, then l_i and l_j form a seed plane P_{l_i,l_j} and we add it to seed plane set $\Omega(l_i)$. Next, for each seed plane, we calculate 2 main directions, i.e., $d_1(P_{l_i,l_j}) = normalize(\overrightarrow{l_i})$ and $d_2(P_{l_i,l_j}) = normalize(\overrightarrow{l_i} \times (\overrightarrow{l_i} \times \overrightarrow{l_j}))$ and the confidence $p(P_{l_i,l_j}) = min((\overrightarrow{l} \cdot \overrightarrow{d})/(proj(\overrightarrow{d}))), l \in \{\overrightarrow{l_i}, \overrightarrow{l_j}\}$, where $\overrightarrow{d}$ is the main direction roughly parallel with $\overrightarrow{l}$ and $proj(\overrightarrow{d})$ denotes the projection range of l_i and l_j on the direction $\overrightarrow{d}$. The seed plane with highest confidence in $\Omega(l_i)$ is accepted as a candidate plane. Once a candidate plane is selected, say P_{l_i,l_k}, we then enlarge this candidate plane by adding coplanar lines l that $|\Pi(l)| < 2$ and are parallel or perpendicular with 2 main directions.

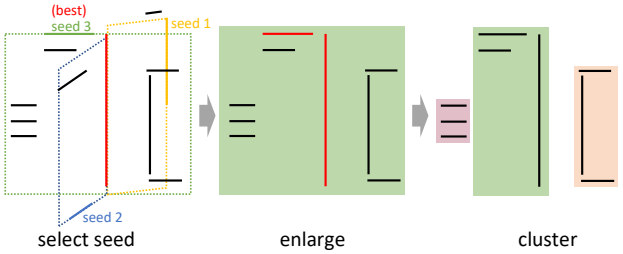

Figure 3: *The main steps of plane detection. Firstly best seed plane is selected from candidate seed planes, then seed plane is enlarged by adding coplanar lines, lastly a clustering operation is performed to break the plane into several sub planes.*

Lines in the enlarged candidate plane may be pretty scattered. Then a DBSCAN-like line clustering is applied in which point distance is replaced by minimal distance of 2 lines, to break P_{l_i,l_k} into several compact sub-planes. We define the score of a plane P as $score(P) = \sum_{l \in \Lambda(P)} |l|$ and remove the planes that have a score below the threshold T_p. In our implementation T_p is set to $0.05 \cdot avg(score(P))$. Finally the remaining sub-planes are added to plane set $\mathcal{S}$ and update $\Pi(l)$ for lines l that support these sub-planes. We continue add plane for each l_i until $\Pi(l_i) = 2$ or there is no candidate plane available.

Lastly, we set the boundary for each plane by calculating the minimal and maximal projections of all lines supporting the plane in 2 main directions. Hence each plane gets a rectangular boundary. Moreover, we enlarge the boundary of each plane a little bit by adding a margin around it. The results of each step of plane detection are shown in Fig. 4.

Corner generating Once planes are obtained from the line cloud $\mathcal{L}$, we calculate intersection points of all three mutually orthogonal planes. These intersection points are the potential corner origins. Only intersection points within the boundaries of all three planes are counted as valid points since the plane's boundary represents the physical range of the corresponding plane in the real world,

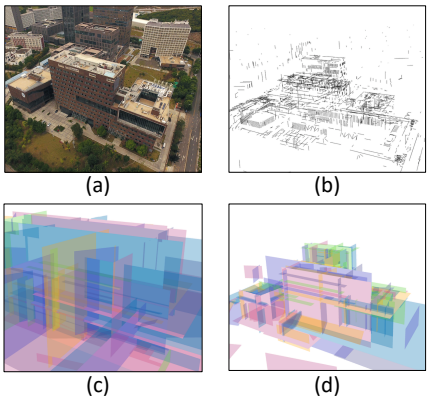

Figure 4: *The result of plane detection. (a) is the scene image and (b) is the line cloud. (c) and (d) are the planes detected without and with plane clustering.*

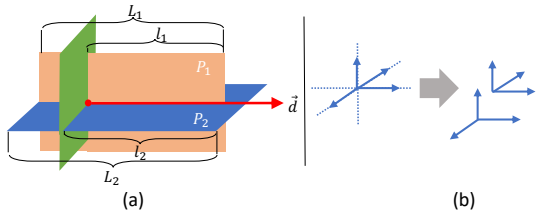

Figure 5: *Generating corners from an intersection point. We first determine valid directions (a), then pick all combinations of valid directions to form corners (b).*

and intersection points outside any boundary tend to be nonexistent. Three orthogonal planes intersect and generate three intersection lines. Start from intersection point and move along intersection lines we can get six extending directions. Then we determine whether these directions are legal. If a direction is legal, this means we can explore the rest parts of a cube along with it. As depicted in Fig. 5(a), to exam whether a direction $\overrightarrow{d}$ is legal, let P_1 and P_2 be the two neighbor planes of $\overrightarrow{d}$, we calculate the confidence of this direction $\overrightarrow{d}$ as $p(\overrightarrow{d}) = \sum_{i\in\{1,2\}} min(max(\frac{l_i}{L_i}, \frac{l_i}{\lambda}), 1)$, where l_i is the distance from the origin to the P_i's boundary along direction $\overrightarrow{d}$, L_i is the width of the P_i's boundary along direction $\overrightarrow{d}$, and λ is a good enough length that we are likely to accept $\overrightarrow{d}$ immediately when $l_i > \lambda$. A direction with a confidence larger than a threshold $T_c = 0.8$ is accepted as a valid axis direction and assigned with an initial axis length $0.5 \cdot (l_1 + l_2)$. As shown in Fig 5(b), all possible combinations of three mutually orthogonal directions are picked from valid axis directions to generate corners along with origin, and these corners are appended to corner set $\mathcal{C}$, where each $c_i \in \mathcal{C}$ consists of one origin position o_i, three axis directions $\overrightarrow{d_i^1}, \overrightarrow{d_i^2}, \overrightarrow{d_i^3}$ and their corresponding initial axis lengths l_i^1, l_i^2, l_i^3. Fig. 6 shows the result of corner generating.

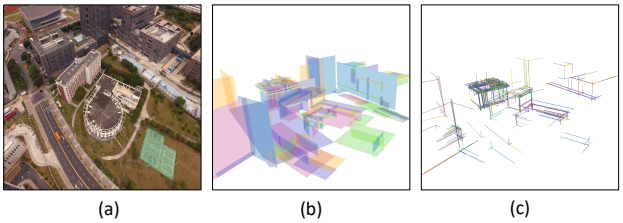

Figure 6: *The result of corner generating. (a) is the scene image, (b) is the planes detected from the line cloud and (c) is the generated corners by plane intersecting.*

3.3. Cube fitting

In this part, we fit cubes with point cloud to get cube represents of buildings. The cube represent is a cube shape parameterized by a fixed corner. The inputs of this step are point cloud $\mathcal{P}$ and corners $\mathcal{C}$ generated in previous step. The output is cube represents of buildings $\mathcal{R}$, where each $r_i \in \mathcal{R}$ contains one origin position o^i and three axis directions $\overrightarrow{d_i^1}, \overrightarrow{d_i^2}, \overrightarrow{d_i^3}$ and their corresponding fixed axis lengths l_i^1, l_i^2, l_i^3.

Since corners are weak evidence of cubes' presence, we fit cubes from the point cloud $\mathcal{P}$ to search for more supports. By changing the parameter of a corner, i.e., lengths of axis and poses of origin and axis directions, cubes in different sizes and positions can be generated. Please note that we also change the origin and axis directions of a corner because these parameters obtained from the previous step may not be accurate due to errors. The goal of the fitting process is to find the optimal parameters of a corner so that generated cube fits nearby point cloud best. We formulate the fitting process as a non-linear least square optimization problem, i.e., given a corner $c_i \in \mathcal{C}$, finding the optimal parameters $(o_i, \overrightarrow{d_i^1}, \overrightarrow{d_i^2}, \overrightarrow{d_i^3}, l_i^1, l_i^2, l_i^3)$ that minimizes the following objective function

$$E(c_i) = E_{cover}(c_i) + E_{regul}(c_i) + E_{const}(c_i) \tag{1}$$

which contains three terms: a coverage term $E_{cover}(c_i)$ that encourages a cube to fit nearby point cloud as much as possible, a regularization term $E_{regul}(c_i)$ that penalizes the over-extension of axes and over-shift of origin and axis directions, a constraint term $E_{const}(c_i)$ that forces three axis directions of a corner to keep mutually perpendicular and maintain cuboid shape.

Coverage term This term rewards points that roughly lie on the surfaces of the generated cube and penalizes points that are located inside the cube, thus encourages axes to prolong to find more points on the cube and fits the entire building from a corner. The smaller this term is, the better the cube fits point cloud. The coverage term is defined as

$$E_{cover}(c_i) = \phi^2 \left(\sum_{p\in\mathcal{P}} f_l(d(p, c_i)) \right), \tag{2}$$

where $d(p, c_i)$ is the nearest distance from a point p to the surfaces of the cube determinated by c_i. $d(p, c_i)$ is negative when p is

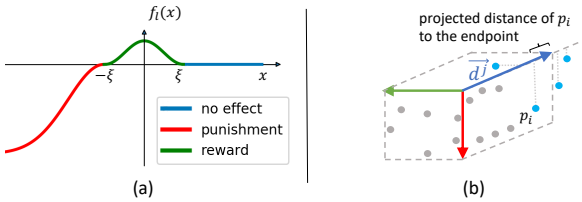

Figure 7: *(a) The overall shape of loss function $f_l(x)$. Points that roughly lie on the cube surfaces get a reward (green line). Points inside the cube get a punishment (red line). Others make no contributions (blue line). (b) An example of the calculation of $\pi(\overrightarrow{d^j},N)$ in regularization term where $N=4$, $\pi(\overrightarrow{d^j},N)$ is obtained by calculating the average of N closest projected distance on axis $\overrightarrow{d^j}$, the blue points are the N points used to calculate projected distance.*

inside the cube and positive when p is outside. The loss function $f_l(x)$ is defined as

$$f_l(x)=\begin{cases}0 & ,x>\xi\\ 0.5\cdot(cos(\frac{\pi\cdot x}{\xi})+1) & ,-\xi\le x\le\xi\\ \eta\cdot(\exp(-\frac{(x+\xi)^2}{2\sigma^2})-1) & ,x<-\xi\end{cases}, \tag{3}$$

where ξ is the maximal distance that a point to the plane it belongs to, η and σ control the punishment. The overall shape of $f_l(x)$ is depicted in Fig. 7(a).

$\phi(x)$ is a mapping function that maps $[-\infty,+\infty]$ to $[0,+\infty]$ which is defined as

$$\phi(x)=e^{-p\cdot x}, \tag{4}$$

where p controls convergence speed and is set to 0.4.

Regularization term This term prevents axes of corner from excessive prolonging and limits the shift of origin and axis directions from initial poses. To calculate this term, we first project all the points that have positive reward on each axis $\overrightarrow{d^j}$ and select N of them with the shortest projected distances to the endpoint of the axis to calculate the average projected distance, which is denote as $\pi(\overrightarrow{d^j},N)$. Fig. 7(b) depicts a case where $N=4$. The regularization term is defined as

$$E_{regul}(c_i)=\sum_{j=1}^{3} {f_r}^2(\pi(\overrightarrow{d_i^j},N))+{f_d}^2(c_i,c_{i0}), \tag{5}$$

where $f_r(x)$ is a loss function that penalizes excessive axis prolonging and is defined as

$$f_r(x)=\gamma\cdot(e^{q\cdot x}-1), \tag{6}$$

where γ denotes the regularization term punishment and q is set to 0.2 by default. $f_d(c_i,c_{i0})$ describes the difference between current parameter c_i and initial parameter c_{i0} and is defined as

$$f_d(c_i,c_{i0})=|o_i-o_{i0}|+\sum_{j=1}^{3}|\overrightarrow{d_i^j}-\overrightarrow{d_{i0}^j}|. \tag{7}$$

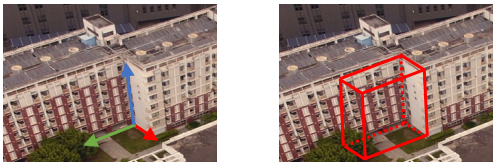

Figure 8: *An example of a virtual cube generated in the cube fitting step. (a) is the corner and (b) is the fitted cube that doesn't exist in real world.*

Constraint term This term maintains the constraint that three axis directions are approximately mutually perpendicular and is simply defined as

$$E_{const}(c_i)=k\cdot((\overrightarrow{d_i^1}\cdot\overrightarrow{d_i^2})^2+(\overrightarrow{d_i^1}\cdot\overrightarrow{d_i^3})^2+(\overrightarrow{d_i^2}\cdot\overrightarrow{d_i^3})^2), \tag{8}$$

where k is a factor that balances the influence of this term and is set to 0.02 in our implementation.

Virtual cube removal In aforementioned steps we don't apply any visibility examination, some corners don't belong to any actual building can be generated and their corresponding fitted cubes are *virtual*, as illustrated in Fig. 8. In this step these virtual cubes are removed. For each surface s_i^j of corresponding cube of each $r_i\in\mathcal{R}$, we count the numbers of points that roughly lie on s_i^j, denoted as $N_p(s_i^j)$, and the number of cameras that are able to see s_i^j,denoted as $N_c(s_i^j)$. Then surfaces are sorted in descending twice according to $N_p(s_i^j)$ and $N_c(s_i^j)$ respectively while the ranks are recorded as $R_p(s_i^j)$ and $R_c(s_i^j)$. Lastly we calculate a rank difference sum $D(r_i)=\sum_{j=1}^{6}|R_p(s_i^j)-R_c(s_i^j)|$ for each $r_i\in\mathcal{R}$. r_i that holds condition $D(r_i)>N_t$ ($N_t=8$ by default) is classified as a virtual cube and removed from $\mathcal{R}$. The intuition of this step is that if a cube is real then the surface seen by more cameras tends to have more points.

3.4. Cube registration

Due to the data missing and outliers in the point cloud, not all cubes are fitted in the proper size, i.e., the axis lengths are not converged to the proper length. In contrast, the estimation of a corner's origin and axis directions is accurate in most cases. In this section, we apply a registration step to ensure all cubes are in the correct sizes. Since the point cloud is already incomplete, we seek more hints from images. For each cube, an image that sees it most is picked to perform registration.

Guiding vector map and step map To properly guide cubes to desirable shapes, we introduce the guiding vector map and the step map to assist cubes in aligning their 2D projections with corresponding visible edges on the images. The guiding vector map indicates the moving direction of each pixel on the image, along which a salient edge can be found. To generate the guiding vector map of an image, we use structured forests [DZ13] to obtain an edge map where each pixel p_i is assigned with a response value $s_i\in[0,1]$, indicating the probability of being an edge pixel. For

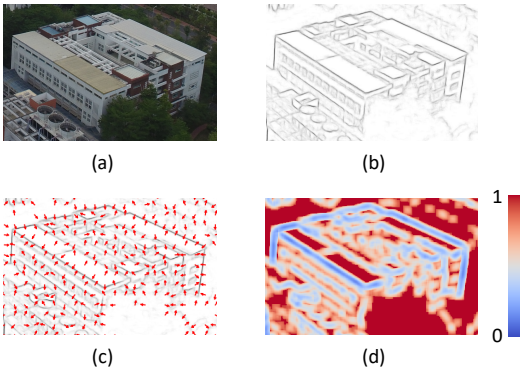

Figure 9: *The registration step. (a) the original image. (b) the detected edge. (c) the calculated guiding vector map, note that the arrows represent the guiding vectors of pixels in the arrow tail. (d) the calculated step map, where red denotes larger steps and blue denotes smaller steps.*

each pixel p_i we calculate its guiding vector as

$$g_i = (\sum_{p_j \in N(p_i,r)} \frac{w_{ji} \cdot s_j \cdot \overrightarrow{p_j p_i}}{|\overrightarrow{p_j p_i}|}) / (\sum_{p_j \in N(p_i,r)} w_{ji}), \tag{9}$$

where $w_{ji} = \exp(-|\overrightarrow{p_j p_i}|)$ and $N(p_i,r)$ denotes edge pixels inside the circle with a center p_i and radius r. The step map indicates the moving step that each pixel should advance to approach a salient edge. Closer the pixel to the edge, smaller the step is. This property helps pixel to approach a salient edge gradually rather than oscillating around it. The moving step of each pixel p_i is calculated as

$$h_i = \left(1 - (\sum_{p_j \in N(p_i,r)} w_{ji} \cdot s_j) / (\sum_{p_j \in N(p_i,r)} w_{ji})\right)^c. \tag{10}$$

Here constant parameter c is set to $c = 4$ by default.

Correspondence search We call edge e a *free edge* of axis direction $\overrightarrow{d}$ when e is on the cube plane that is perpendicular to $\overrightarrow{d}$, that is to say, the position of e is affected by the axis length on axis direction $\overrightarrow{d}$.

For a given axis direction $\overrightarrow{d_i}$, we use notation $\mathcal{E}_i$ to denote all free edges of d_i that are visible on I. For each $e \in \mathcal{E}_i$, we sample points along e uniformly to obtain projected point set $\mathcal{V} = \{v_1, ..., v_n\}$. Then we find corresponding points on the edge map of I for all points in $\mathcal{V}$. The correspondence search is performed in an iterative way. Start from a point $v_i \in \mathcal{V}$, we evaluate the moving step of v_i as $step(v_i) = \Delta \cdot h_i$, if $step(v_i) < \Delta_0$ holds we stop moving and set current point as the stopping point. Here Δ and Δ_0 are set according to image size, and we set $\Delta = 6$ and $\Delta_0 = 2$ by default. Otherwise, we move to next point $v_i + g_i \cdot step(v_i)$ and repeat until a stopping point is found. Once reached a stopping point, we search the corresponding point p_j of v_i that makes score

$$S(v_i, p_j) = \frac{{s_j}^a \cdot \exp(-\frac{(1-|\overrightarrow{t_i} \cdot \overrightarrow{t_j}|)^2}{2\sigma^2}) \cdot \exp(-\frac{(1-l_i/l_j)^2}{2\sigma^2})}{|\overrightarrow{v_i p_j}|^b} \tag{11}$$

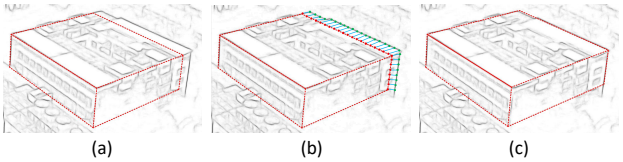

Figure 10: *Searching corresponding points for free edges. (a) is a cube that not converged to proper size due to data missing. (b) is the searched corresponding on free edges. (c) is the newly re-estimated cube size.*

reach maximum. Here, t_i is the edge orientation at v_i and is calculated as $t_i = normalize(\overrightarrow{v_i v_{i-1}})$, t_j is the edge orientation at p_j and is obtained in edge detection step using structured forests [DZ13], l_i denotes the length of $\overrightarrow{v_i v_{i-1}}$, l_j denotes the length of $\overrightarrow{p_i p_{i-1}}$. The parameters are set to $a = 0.7$, $b = 0.5$ and $\sigma = 0.3$. The matching score term is inspired by the work of Huang et al [HXM*18] and take distance continuity, orientation consistency, edge saliency into consideration for desirable correspondence search. We use notation $m(v_i)$ to denote the matched corresponding point p_j.

Estimating new length After searching corresponding points for all sampled points of e on an image I, we calculate the axis length offset $\delta(e, I, \overrightarrow{d})$ which indicates how much the axis length should adjust to make e align with matched edge on I. The offset $\delta(e, I, \overrightarrow{d})$ is calculated as

$$\delta(e, I, \overrightarrow{d}) = \frac{\sum_{v_i \in \mathcal{V}} (\overrightarrow{v_i m(v_i)} \cdot \overrightarrow{d'})}{|\mathcal{V}|} \cdot \frac{l}{l'}, \tag{12}$$

where $\overrightarrow{d'}$ is the normalized projected direction of $\overrightarrow{d}$ on image I, l is the axis length of $\overrightarrow{d}$ and l' is the projected length of l on image I. We calculate offset for all visible free edges on all selected images and then calculate the final axis offset $\bar{\delta}(\overrightarrow{d_k})$ as

$$\bar{\delta}(\overrightarrow{d_k}) = \frac{\sum_{e \in \mathcal{E}_k} \delta(e, I, \overrightarrow{d_k})}{|\mathcal{E}_k|}. \tag{13}$$

The new length of the axis on direction d_k is estimated as $l_k = l_k + \bar{\delta}(\overrightarrow{d_k})$. We repeat correspondence search and new length estimating until the axis length on d_k converges.

We apply the steps mentioned above for all axes of all cube represents, thus ensure all cubes are in the correct sizes. Lastly, we convert all cube represents to polygonal surface models.

4. Experiments and discussion

Dataset We tested our method in both real and synthetic datasets. The real datasets are captured by drone. The synthetic datasets are obtained by rendering polygonal models in different views using photorealistic renderers. These datasets contain various scenes from single buildings to large-scale urban scenes that contain multiple buildings. All the buildings in the datasets are in the different styles and shapes.

Implementation detail We implemented our approach using C++. In the cube fitting step(Sect 3.3), we solve the optimization

problem using Google Ceres solver [AMO10]. Besides, we only evaluate points around the cube center in a certain radius and use kd-tree to accelerate points inquiry in consideration of algorithm efficiency. After fitting cubes for all corners, only results with a score below a threshold are considered valid ones and are appended to set $\mathcal{R}$. Cubes that overlap significantly with other ones are removed to avoid redundancy. After polygonal models are generated, we assign textures for each triangle from the input images to yield better visual effects.

Reconstruction results Our approach is designed to reconstruction Manhattan-world buildings. We test our approach on the aforementioned datasets. The reconstruction results are shown in Fig. 11. Datasets (a)-(f) are captured from real world and datasets (g)-(h) are captured from synthetic scenes. In Tab. 1 we list some statistics on the tested datasets. Due to the complexity of real world scene, the reconstructed point clouds are relatively sparse and both line clouds and point clouds contain significant amounts of outlier. Our method still roughly reconstructed these buildings and yielded good approximations by leveraging the structural information from line cloud and the range information from point cloud. In Fig. 11(f), we reconstructed an office building that consists of multiple cubes. Though the small cube shapes of the building are barely recognizable in the sparse point cloud, our approach still recovered all the cube details thanks to structural information from the line cloud. In Fig. 12, we show the details of our reconstructed results. In the nasty cases shown in the top view Fig. 12 a(1), building highlighted in the red box, whose back planes are entirely missing, still got reconstructed. In the dataset *IT lab*, though the building (Fig. 12 a(2)) is not a cuboid shape building in the strict sense, our approach still reconstructed it using an assembly of several cubes. In the dataset metropolis, while cuboid shape buildings can be reconstructed easily (Fig. 12 a(3), our approach failed in handling non-Manhattan-world buildings in Fig. 12 a(4), where 2 cylinder shape buildings (indicated with red arrows) are completely missing in our result. In Tab. 1, we show the statistics of our reconstruction results. Our reconstruction results have only a few face numbers while preserving the shape features of buildings, which is very suitable for large-scale city modeling.

Robustness of cube fitting The critical step of our approach is cube fitting, and the quality of this step largely dominates the result of reconstruction. We commit several experiments on how effective and robust our cube fitting step is. As shown in Fig. 11, most cubes are fitted to the correct size in most cases. For a given corner, our objective function encourage the corner to explore the rest part and prevent over extending at the same time. Recall that in the corner generating step (Sect 3.2) each axis direction is assigned with an initial length according to the distance from the corner origin to the neighbor plane boundary. Since the plane's boundary indicates the actual size of the real plane in some ways, this initial value usually sits around the optimal point, which makes the optimization algorithm easily converge at the desired position, not local optima. Solvers like Ceres normally use gradient-based methods to solve optimization problems, a corner's optimal parameter can be solved efficiently within few iterations under this circumstance. Though our method do not require the point cloud to be dense, the points shall depict the overall shape of the building's planes. When the plane is severely missing, the fitting step usually fails to converge at desirable size.

Effect of registration In most cases, our registration step is able to adjust cubes into proper size. Since our registration algorithm is running in a greedy strategy, cubes might be aligned to close but incorrect edges. Fig. 13 shows a failure case on this occasion. Another factor impact registration results is the radius used to compute guiding vector and step. Larger radius leads to large searching range and more robust corresponding searching, meanwhile, the computation time also increases with the radius.

Performance We tested our approach on a personal laptop with a 2.7GHz dual-core CPU and 8GB memory. Performing a reconstruction usually takes few minutes, and Tab. 1 illustrates the computation times on various data. The most time-consuming part of our approach is cube fitting (Sect 3.3) which mostly takes about 70% of total running time. The fitting time is linearly related to the number of corners. The number of corners increases significantly with the size and number of planes. Larger plane boundary and plane clustering radius often lead to more corners, thus more cubes can be fitted i.e., better reconstruction result. However, in our test most fitted cubes are removed in the virtual cube removal step and many computations are wasted. Smaller plane boundary and clustering radius lead to fewer corners and some small cubes might be missed while shorter running time is got in return. So it requires user to adjust parameters to balance between quality and speed according to dataset. As for the registration step, most time is spent on the edge detection and this time increases significantly along with the image size. The corresponding search and estimating new length are relatively fast since the cubes tend to converge to proper size within 3-5 iterations. The aforementioned time does not take preprocessing time into account, which usually takes few minutes to a half-hour depending on the number of images.

Comparison with other methods Fig. 14 shows the comparison results with 5 mainstream methods on several datasets. We first follow the typical photogrammetry pipeline using SfM technique to obtain camera pose and sparse point cloud. Then run MVS to get dense point cloud. Screened Poisson reconstruction [KH13] is applied to reconstruct surface from both sparse (Fig. 14(a)) and dense point cloud (Fig. 14(b)). A mesh simplification using quadric metric [Hop99] (Fig. 14(c)) is performed on the model reconstructed from dense point cloud to reduce face number to the 0.01 times of original number. The sparse point cloud is inputted to other two methods polyfit [NW17] (Fig. 14(d)) and Li et al [LWN16] (Fig. 14(e)). Due to lack of data, the Screened Poisson reconstruction on sparse point cloud is pretty bumpy and lacking details. Though Screened Poisson reconstruction on dense point cloud recovered significant details, the models contain millions of triangles and occupy large storage. While the simplified model contains fewer triangles, some features of building structure are severely lost. Other lightweight reconstruction methods polyfit [NW17] (Fig. 14(d)) and Li et al [LWN16] (Fig. 14(e)) that heavily rely on plane detection results, failed to reconstruct faithful models due to data missing and unreliable plane detection from point cloud. Compare to methods merely using point cloud, our point-line-combined method is more robust in plane detection and can capture more building features. Besides, thanks to fitting cubes

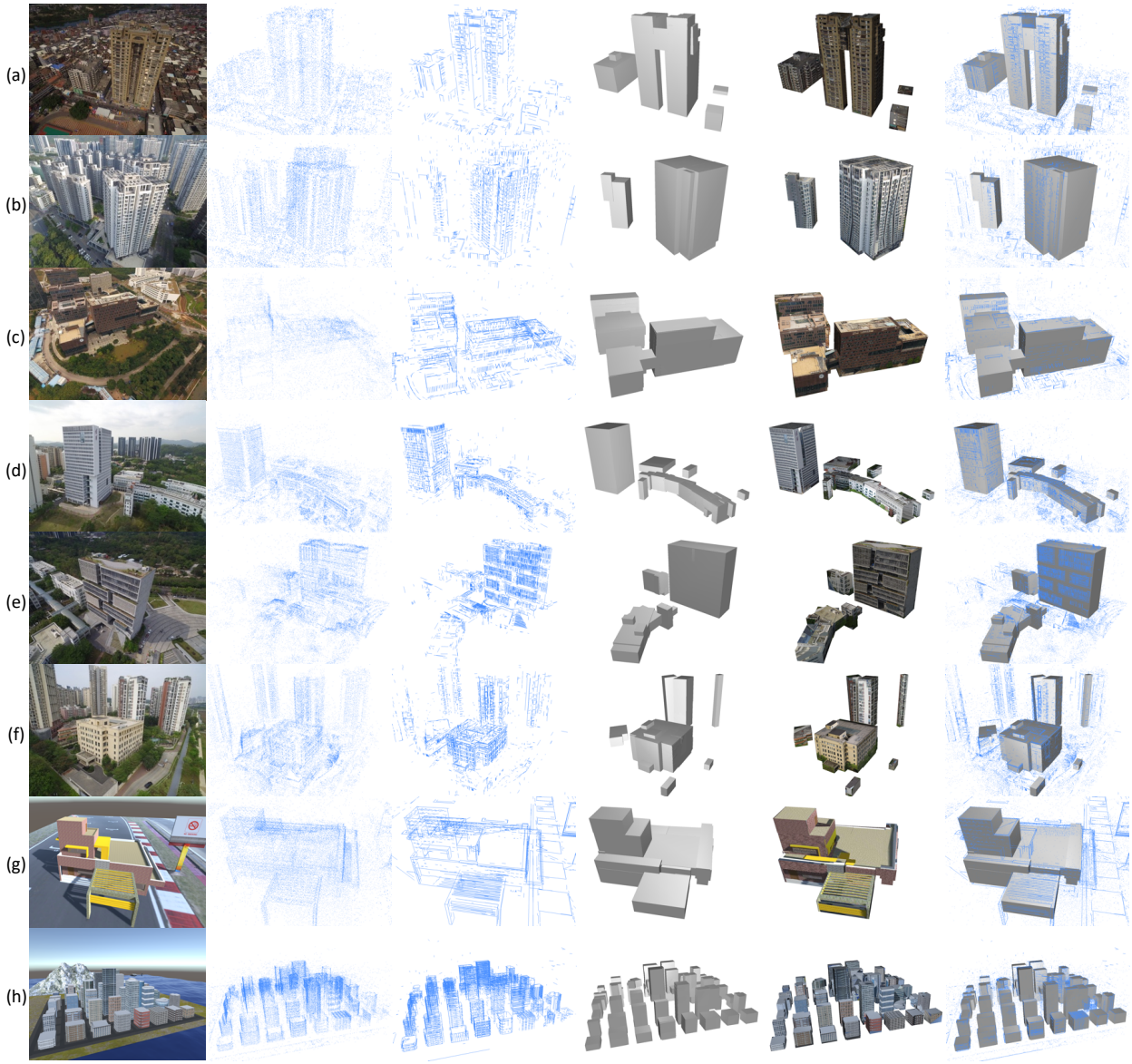

Figure 11: *Reconstruction results on various datasets. From left to right: reference image, point cloud, line cloud, fitted cubes, fitted cubes with textures, fitted cubes overlaid with point cloud.*

from corners, our approach can reconstruct Manhattan-world buildings even when only few planes are available, as our result shown in Fig. 14(f) on case *ocean lab*. Besides, our method also shows the competitive results in face number and running time, which has significant advantages on large-scale city reconstruction application. The statistic on face number and running time about the results in Fig. 14 are available in Tab. 2.

Limitation The result of our approach may heavily rely on the outputs of COLMAP and Line3D++ systems. Thus the quality of point cloud and line cloud is a crucial bottleneck of our approach. Another limitation is that our method assumes that buildings are cuboid-shaped and do not work for buildings with complex structures (e.g., cylinder buildings in Fig. 12 a(4), wireframe structures), limiting our approach's applicability.

5. Conclusion and future work

We introduced a novel approach addressing the challenge task that reconstructing Manhattan-world buildings from low-quality data. Our approach firstly extracts planes from the line cloud and generates corners by intersecting planes, and we proposed a novel optimization formulation to fit cube represents of buildings starting from corners. Lastly, more accurate lightweight polygonal surface models are obtained through the registration step. Experiments on both real and synthetic datasets show that our approach is robust

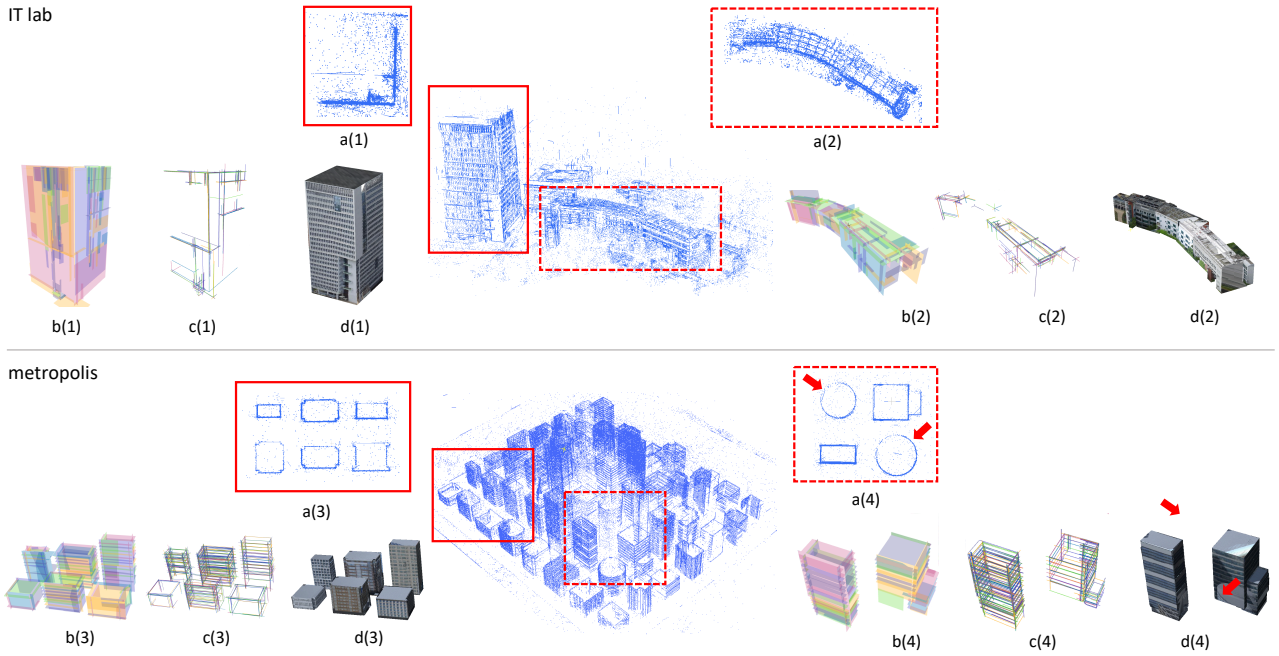

Figure 12: *Reconstruction details on datasets IT lab and metropolis. The image in the middle of each line is the sparse point cloud and line cloud. a(1-4) are the top view of the target buildings in the red boxes, b(1-4) are detected planes, c(1-4) are generated corners and d(1-4) are reconstructed models.*

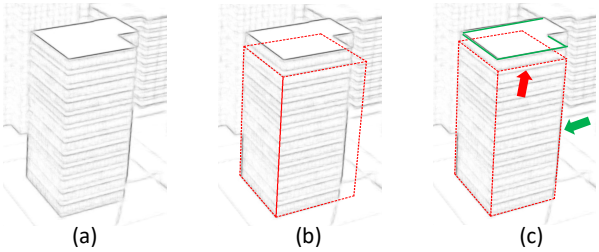

Figure 13: *A bad case of registration. (a) is the building edge map. (b) is the cube before registration. (c) is the registered cube, where green arrow indicates an edge properly aligned while red arrow indicates the edges that failed to align with desirable target (green lines) and got stuck at local optima.*

and efficient in reconstructing cuboid-shaped buildings from the incomplete and sparse point cloud. Our approach breaks the limitation of the previous approaches that they are vulnerable to plane missing since our approach fits a cube from a corner without the prerequisite that all planes of a cube shall be present.

As future work, we would like to explore the possibility of fitting more primitives (e.g., cylinder and sphere) and reconstruct buildings in arbitrary shapes from a sparse point cloud, a line cloud, and images. Another possible extension is to use more semantic information obtained from images to improve reconstruction results.

Table 1: *Statistics on the datasets in Fig. 11*

Dataset	#img	#line	#point	#face	time
(a) apartment1	24	1314	19385	192	16s
(b) apartment2	27	2274	37952	156	11s
(c) institute	33	3375	28246	768	2m12
(d) IT lab	51	4024	57750	924	53s
(e) ocean lab	59	4554	43137	612	1m42s
(f) office	59	4751	46835	2280	3m32s
(g) restaurant	66	3032	49719	384	1m3s
(h) metropolis	88	11606	79070	7920	5m12s

References

[ABCO*03] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Computing and rendering point set surfaces. *IEEE Transactions on visualization and computer graphics 9*, 1 (2003), 3–15.

[AMO10] AGARWAL S., MIERLE K., OTHERS: Ceres solver. http://ceres-solver.org, 2010.

[BdLGM14] BOULCH A., DE LA GORCE M., MARLET R.: Piecewise-planar 3d reconstruction with edge and corner regularization. In *Computer Graphics Forum* (2014), vol. 33, Wiley Online Library, pp. 55–64.

[BENV06] BAY H., ESS A., NEUBECK A., VAN GOOL L.: 3d from line segments in two poorly-textured, uncalibrated images. In *Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06)* (2006), pp. 496–503.

[BMR*99] BERNARDINI F., MITTLEMAN J., RUSHMEIER H., SILVA C., TAUBIN G.: The ball-pivoting algorithm for surface reconstruction.

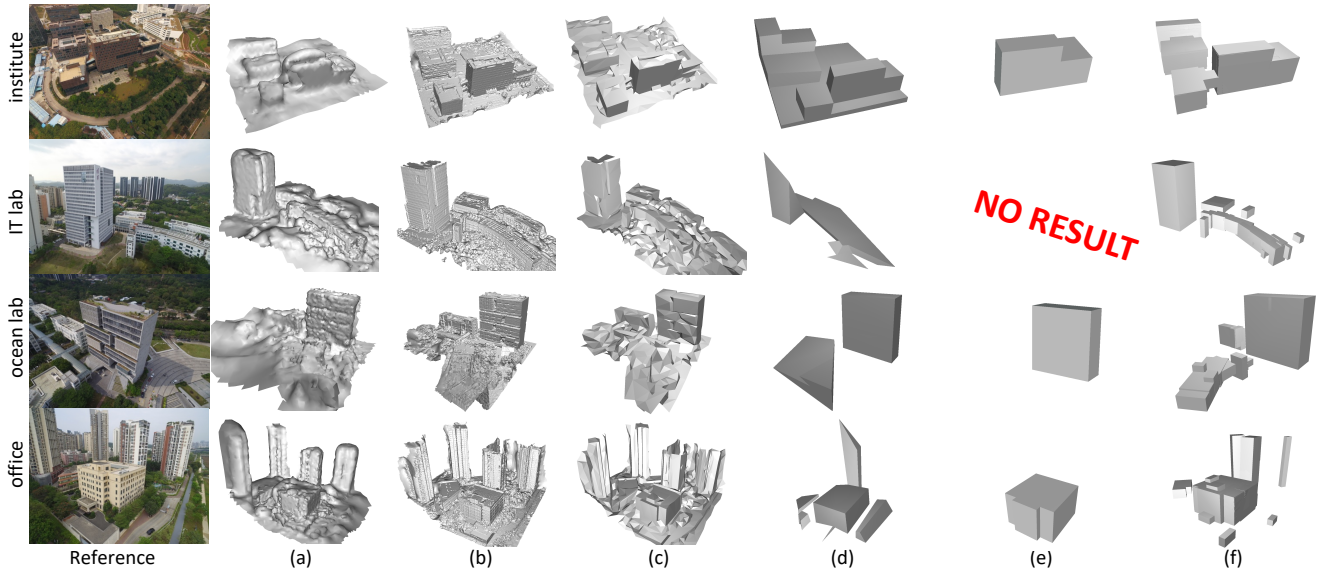

Figure 14: *Comparison with five methods on several datasets. (a) sparse point cloud + Screened Poisson [KH13]. (b) MVS + Screened Poisson [KH13] then (c) simplified using quadric metric [Hop99]. (d) polyfit [NW17]. (e) Li et al [LWN16]. (f) our results.*

Table 2: *Face number and full running time of methods shown in Fig. 14. (a) sparse point cloud + Screened Poisson [KH13]. (b) MVS + Screened Poisson [KH13] then (c) simplified using quadric metric [Hop99]. (d) polyfit [NW17]. (e) Li et al [LWN16]. (f) ours.*

method	institute		IT lab		ocean lab		office	
	#face	time	#face	time	#face	time	#face	time
(a)	9K	3.5min	20K	5.4min	22K	6.6min	22K	9min
(b)	1.8M	150min	0.9M	96min	1.3M	155min	2.8M	181min
(c)	18K	151min	9K	96min	13K	156min	28K	182min
(d)	290	4min	192	5.5min	96	6.8min	152	9.2min
(e)	32	3.3min	-	-	12	6.2min	28	8.2min
(f)	768	5.5min	924	6.9min	612	9min	2280	13.5min

IEEE transactions on visualization and computer graphics 5, 4 (1999), 349–359.

[BS05] BARTOLI A., STURM P.: Structure-from-motion using lines: Representation, triangulation, and bundle adjustment. *Computer Vision and Image Understanding 100*, 3 (2005), 416–441.

[CY99] COUGHLAN J. M., YUILLE A. L.: Manhattan world: Compass direction from a single image by bayesian inference. In *Proceedings of the seventh IEEE international conference on computer vision* (1999), vol. 2, IEEE, pp. 941–947.

[DZ13] DOLLAR P., ZITNICK C. L.: Structured forests for fast edge detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (December 2013).

[FCSS09] FURUKAWA Y., CURLESS B., SEITZ S. M., SZELISKI R.: Manhattan-world stereo. In *2009 IEEE Conference on Computer Vision and Pattern Recognition* (2009), pp. 1422–1429.

[FP09] FURUKAWA Y., PONCE J.: Accurate, dense, and robust multiview stereopsis. *IEEE transactions on pattern analysis and machine intelligence 32*, 8 (2009), 1362–1376.

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (1997), pp. 209–216.

[HMB17] HOFER M., MAURER M., BISCHOF H.: Efficient 3d scene abstraction using line segments. *Computer Vision and Image Understanding* (2017), 167–178.

[Hop99] HOPPE H.: New quadric metric for simplifying meshes with appearance attributes. In *Proceedings Visualization '99 (Cat. No.99CB37067)* (1999), pp. 59–510.

[HWZ*18] HUANG K., WANG Y., ZHOU Z., DING T., GAO S., MA Y.: Learning to parse wireframes in images of man-made environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 626–635.

[HXM*18] HUANG H., XIE K., MA L., LISCHINSKI D., GONG M., TONG X., COHEN-OR D.: Appearance modeling via proxy-to-image alignment. *ACM Transactions on Graphics (TOG) 37*, 1 (2018), 1–15.

[IYF15] IKEHATA S., YANG H., FURUKAWA Y.: Structured indoor modeling. In *Proceedings of the IEEE International Conference on Computer Vision* (2015), pp. 1323–1331.

[JKTS10] JAIN A., KURZ C., THORMÄHLEN T., SEIDEL H.-P.: Exploiting global connectivity constraints for reconstruction of 3d line segments from images. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2010), IEEE, pp. 1586–1593.

[KBH06] KAZHDAN M., BOLITHO M., HOPPE H.: Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing* (2006), vol. 7.

[KH13] KAZHDAN M., HOPPE H.: Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG) 32*, 3 (2013), 1–13.

[KM14] KIM C., MANDUCHI R.: Planar structures from line correspondences in a manhattan world. In *Asian Conference on Computer Vision* (2014), Springer, pp. 509–524.

[KMYG12] KIM Y. M., MITRA N. J., YAN D.-M., GUIBAS L.: Acquiring 3d indoor environments with variability and repetition. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 1–11.

[KSO04] KOLLURI R., SHEWCHUK J. R., O'BRIEN J. F.: Spectral surface reconstruction from noisy point clouds. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (2004), pp. 11–21.

[LBM19] LANGLOIS P., BOULCH A., MARLET R.: Surface reconstruction from 3d line segments. In *2019 International Conference on 3D Vision (3DV)* (Sep. 2019), pp. 553–563.

[LHK09] LEE D. C., HEBERT M., KANADE T.: Geometric reasoning for single image structure recovery. In *2009 IEEE Conference on Computer Vision and Pattern Recognition* (2009), IEEE, pp. 2136–2143.

[LN21] LI M., NAN L.: Feature-preserving 3d mesh simplification for urban buildings. *ISPRS Journal of Photogrammetry and Remote Sensing 173* (2021), 135–150.

[LNL16] LI M., NAN L., LIU S.: Fitting boxes to manhattan scenes using linear integer programming. *International journal of digital earth 9*, 8 (2016), 806–817.

[Low04] LOWE D. G.: Distinctive image features from scale-invariant keypoints. *International journal of computer vision 60*, 2 (2004), 91–110.

[LT98] LINDSTROM P., TURK G.: Fast and memory efficient polygonal simplification. In *Proceedings Visualization'98 (Cat. No. 98CB36276)* (1998), IEEE, pp. 279–286.

[LWC*11] LI Y., WU X., CHRYSATHOU Y., SHARF A., COHEN-OR D., MITRA N. J.: Globfit: Consistently fitting primitives by discovering global relations. In *ACM SIGGRAPH 2011 papers*. 2011, pp. 1–12.

[LWN16] LI M., WONKA P., NAN L.: Manhattan-world urban reconstruction from point clouds. In *European Conference on Computer Vision* (2016), Springer, pp. 54–69.

[MMBM15] MONSZPART A., MELLADO N., BROSTOW G. J., MITRA N. J.: Rapter: rebuilding man-made scenes with regular arrangements of planes. *ACM Trans. Graph. 34*, 4 (2015), 103–1.

[MMPM16] MOULON P., MONASSE P., PERROT R., MARLET R.: Openmvg: Open multiple view geometry. In *International Workshop on Reproducible Research in Pattern Recognition* (2016), Springer, pp. 60–74.

[MSS*08] MATEI B. C., SAWHNEY H. S., SAMARASEKERA S., KIM J., KUMAR R.: Building segmentation for densely built urban regions using aerial lidar data. In *2008 IEEE Conference on Computer Vision and Pattern Recognition* (2008), IEEE, pp. 1–8.

[MWA*13] MUSIALSKI P., WONKA P., ALIAGA D. G., WIMMER M., VAN GOOL L., PURGATHOFER W.: A survey of urban reconstruction. In *Computer graphics forum* (2013), vol. 32, Wiley Online Library, pp. 146–177.

[NSZ*10] NAN L., SHARF A., ZHANG H., COHEN-OR D., CHEN B.: Smartboxes for interactive urban reconstruction. In *ACM SIGGRAPH 2010 papers*. 2010, pp. 1–10.

[NW17] NAN L., WONKA P.: Polyfit: Polygonal surface reconstruction from point clouds. In *Proceedings of the IEEE International Conference on Computer Vision* (2017), pp. 2353–2361.

[SF16] SCHÖNBERGER J. L., FRAHM J.-M.: Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).

[SFCH12] SHEN C.-H., FU H., CHEN K., HU S.-M.: Structure recovery by part assembly. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 1–11.

[SKD06] SCHINDLER G., KRISHNAMURTHY P., DELLAERT F.: Line-based structure from motion for urban environments. In *Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06)* (2006), pp. 846–853.

[SMM17] SALAÜN Y., MARLET R., MONASSE P.: Line-based robust sfm with little image overlap. In *2017 International Conference on 3D Vision (3DV)* (2017), IEEE, pp. 195–204.

[SSS06] SNAVELY N., SEITZ S. M., SZELISKI R.: Photo tourism: exploring photo collections in 3d. In *ACM siggraph 2006 papers*. 2006, pp. 835–846.

[STO15] SUGIURA T., TORII A., OKUTOMI M.: 3d surface reconstruction from point-and-line cloud. In *2015 International Conference on 3D Vision* (2015), IEEE, pp. 264–272.

[SWK07] SCHNABEL R., WAHL R., KLEIN R.: Efficient ransac for point-cloud shape detection. In *Computer graphics forum* (2007), vol. 26, Wiley Online Library, pp. 214–226.

[SZPF16] SCHÖNBERGER J. L., ZHENG E., POLLEFEYS M., FRAHM J.-M.: Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)* (2016).

[VAB10] VANEGAS C. A., ALIAGA D. G., BENES B.: Building reconstruction using manhattan-world grammars. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2010), IEEE, pp. 358–365.

[VGJMR12] VON GIOI R. G., JAKUBOWICZ J., MOREL J.-M., RANDALL G.: Lsd: a line segment detector. *Image Processing On Line* (2012), 35–55.

[Wu13] WU C.: Towards linear-time incremental structure from motion. In *2013 International Conference on 3D Vision-3DV 2013* (2013), IEEE, pp. 127–134.

[XRT12] XIAO J., RUSSELL B., TORRALBA A.: Localizing 3d cuboids in single-view images.

[XZZ*14] XIONG S., ZHANG J., ZHENG J., CAI J., LIU L.: Robust surface reconstruction via dictionary learning. *ACM Transactions on Graphics (TOG) 33*, 6 (2014), 1–12.

[ZG02] ZELINKA S., GARLAND M.: Permission grids: Practical, error-bounded simplification. *ACM Transactions on Graphics (TOG) 21*, 2 (2002), 207–229.

[ZK14] ZHANG L., KOCH R.: Structure and motion from line correspondences: Representation, projection, initialization and sparse bundle adjustment. *Journal of Visual Communication and Image Representation 25*, 5 (2014), 904–915.

[ZQZ*19] ZHOU Y., QI H., ZHAI Y., SUN Q., CHEN Z., WEI L.-Y., MA Y.: Learning to reconstruct 3d manhattan wireframes from a single image. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2019), pp. 7698–7707.