DOI: 10.1111/cgf.14421
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

Zhenbang He[1,2] Yunhai Wang[3] † Zhanglin Cheng[1,2] †

[1]Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences
[2]University of Chinese Academy of Sciences
[3]Shandong University

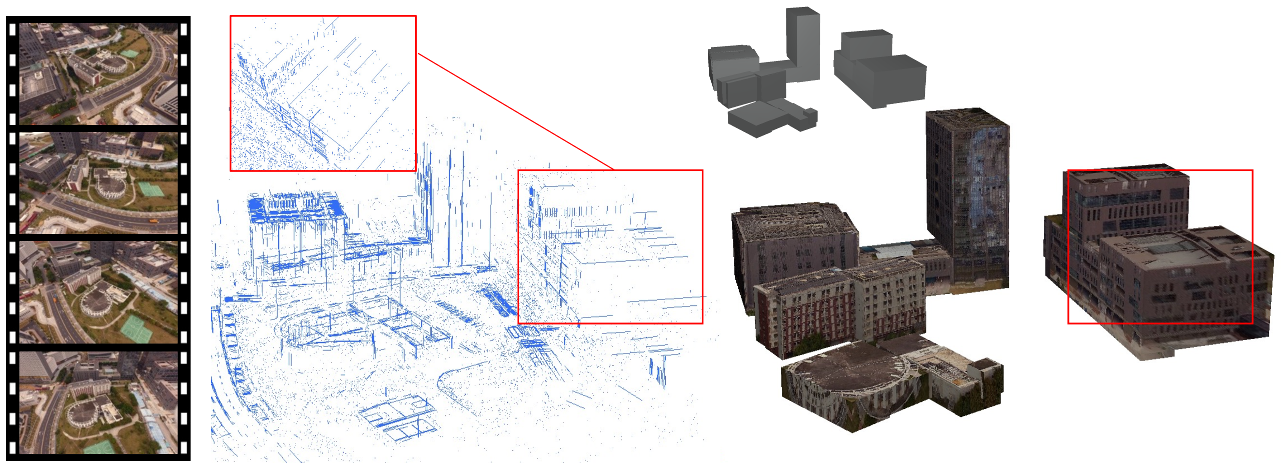

Figure 1: *Reconstruction results on an urban scene. The point cloud and line cloud are first recovered from an image sequence. Then lightweight models are reconstructed by extracting corners from the line cloud and fitting cubes to the point cloud. By fitting cubes from corners, our method can reconstruct buildings even when their back faces are severely lost (see red box). Experiments on various datasets show that our approach is quite competitive in both speed and quality.*

Abstract

The Manhattan-world building is a kind of dominant scene in urban areas. Many existing methods for reconstructing such scenes are either vulnerable to noisy and incomplete data or suffer from high computational complexity. In this paper, we present a novel approach to quickly reconstruct lightweight Manhattan-world urban building models from images. Our key idea is to reconstruct buildings through the salient feature - corners. Given a set of urban building images, Structure-from-Motion and 3D line reconstruction operations are applied first to recover camera poses, sparse point clouds, and line clouds. Then we use orthogonal planes detected from the line cloud to generate corners, which indicate a part of possible buildings. Starting from the corners, we fit cubes to point clouds by optimizing corner parameters and obtain cube representations of corresponding buildings. Finally, a registration step is performed on cube representations to generate more accurate models. Experiment results show that our approach can handle some nasty cases containing noisy and incomplete data, meanwhile, output lightweight polygonal building models with a low time-consuming.

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

1. Introduction

Recently, there has been an emerging interest in urban building modeling due to the increasing demand for 3D urban data in multi-

† Z. Cheng and Y. Wang are joint corresponding authors.
Email:{zb.he, zl.cheng}@siat.ac.cn, cloudseawang@gmail.com

ple application fields, e.g., virtual reality, GIS, BIM, and live map. As a kind of most common architecture, Manhattan-world buildings have been vital modeling objects in the urban building reconstruction. In many applications, the fine details of the building models are not necessary due to the large scale of nowadays cities and the relatively simple geometry of most buildings, especially for Manhattan-world buildings. Lightweight models are more preferred in many applications over highly detailed models containing millions of triangles owing to their superiorities in rendering, data transfer, and storage.

Typical photogrammetry-based 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 a surface reconstruction [KH13, KBH06, ABCO*03, KSO04, BMR*99] step is applied to the point cloud to obtain a polygonal mesh model. Lastly, mesh simplification [LN21, ZG02, LT98, GH97] algorithm is performed on the complex model to get the final lightweight model. 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, the MVS step easily takes days even weeks. Besides, many expensive computations spent on details are discarded and wasted on the mesh simplification step. A natural idea is skipping the MVS step and reconstructing lightweight models directly from the point cloud generated by the SfM step. However, these reconstructed point clouds tend to be pretty sparse owing to the sparsity of feature points extracted from images, which could be worse when facing man-made environments that contain many less textured surfaces and repetitive structures. Although it is impractical to reconstruct surfaces directly from the sparse point cloud, sparse point clouds still reveal the overall shape and existence of buildings. In our work, we seek an efficient solution to reconstruct lightweight models from sparse point clouds by leveraging this information.

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

Another core difficulty of urban building reconstruction is the poor quality of obtained data. For example, urban images captured no matter from the air or land usually contain large amounts 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 large-scale cities at arbitrary angles. All these facts make the obtained data usually incomplete, noisy, and nonuniform. Reconstructing 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 assumptions [FCSS09, LWN16], user interactions [NSZ*10], data-driven learning [XZZ*14], etc.). In our work, we reconstruct urban buildings under the Manhattan-world assumption and use corners (i.e., the sharp conjunction where 3 planes converge) to locate and fit buildings. This is based on an observation that the corners of buildings are salient even when the data is sparse or partially missing, which is especially true for Manhattan-world buildings. Besides, we use images to perform an additional registration, thus generating more accurate results.

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

In summary, the main contributions of this work are as following:

- a framework for the fast and lightweight reconstruction of Manhattan-world buildings from sparse and incomplete data.
- a novel method for detecting planes and corners of buildings from a line cloud.
- a numerical optimization formulation for fitting cubes from a sparse point cloud.
- a novel method for registering cube models of urban buildings with the corresponding images.

2. Related work

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 extracted from images to estimate camera pose and generate point clouds by solving epipolar geometry. Due to the limited number of feature points, generated point clouds are usually sparse. Multi-View Stereo(MVS) [FP09] can generate dense point cloud by performing depth map estimation and depth fusion. Current the state of art reconstruction systems [SF16, SZPF16] allow non-expert users to generate accurate point clouds from an unordered image sequence. These tools have shown impressive results on richly textured surfaces, but they often 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 mil-

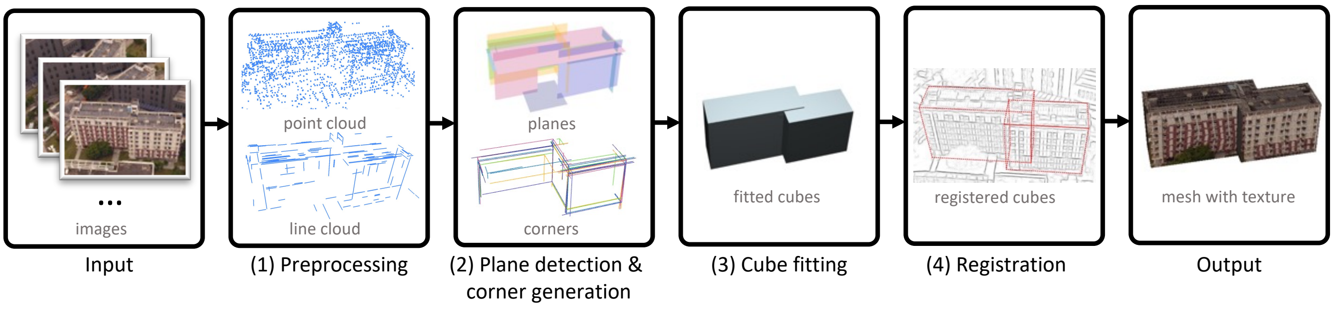

Figure 2: *An overview of our approach for lightweight Manhattan-world urban building reconstruction.*

lions of points, which makes viewing and processing dense point clouds troublesome.

Similar to 3D point reconstruction, 3D line reconstruction uses line segments to recover 3D line structures from images. One key issue is matching lines between different images. Some methods achieve it through line feature descriptors [BENV06], point-wise correspondences between lines [BS05] or coplanarity constraint [SMM17]. Some methods solve it by exploring the lines' global connectivity [JKTS10], or using weak matching [HMB17], thus bypassing the search for explicit correspondences. Some methods [SMM17, ZK14, KM14, SKD06, BS05] try to solve camera pose estimation and 3D line reconstruction simultaneously, others [HMB17, JKTS10] only solve 3D line reconstruction based on camera poses that provided by SfM. In contrast to 3D point reconstruction, 3D line reconstruction is particularly suitable for urban scenes that contain a large number of line structures. Though 3D line reconstruction leaves impressive results on man-made scenes, further reconstruction for mesh models is still a problem. In our work, we use line clouds to detect building planes and fit cubes to point clouds.

Surface reconstruction from point and line cloud. The state-of-the-art 3D reconstruction methods can generate point clouds and line clouds robustly. However, point clouds and line clouds are not suitable for many applications. Thus a surface reconstruction step is needed to generate polygonal meshes. Surface reconstruction from point clouds has been extensively studied. Some of the early methods are mainly based on combinatorial structures, such as Delaunay triangulations [KSO04, BMR*99]. Other methods [KH13, KBH06, ABCO*03] try to reconstruct an approximate implicit surface under surface smoothness assumption. However obtained point clouds are usually incomplete and noisy, and the aforementioned methods may lead to terrible results. Therefore, methods that use prior knowledge to perform reconstruction in a learning [XZZ*14] or data-driven way [KMYG12, SFCH12] have recently emerged. Many methods often produce jagged results on noisy data, hence some methods align surfaces with lines detected from images [HFB16], or planes detected from point clouds [HOP*17] and line clouds [HMFB18] to yield more visually appealing results. Some methods [FL20, ZSGH18, NW17, BdLGM14] attempt to reconstruct a lightweight model ignoring the details instead. Surprisingly, surface reconstruction from line cloud has rarely been explored. Sugiura et al [STO15] proposed a method to efficiently reconstruct 3D surfaces as triangular meshes by integrating line clouds with the point clouds. Bay et al [BENV06] used line clouds reconstructed from 2 poorly-textured, uncalibrated images to reconstruct planar indoor scenes. Recently Langlois et al [LBM19] present a pipeline that uses only line clouds to reconstruct watertight piecewise-planar models.

Manhattan-world reconstruction. The aforementioned general reconstruction techniques usually do not yield desirable results on urban buildings. Therefore, 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 urban 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] segmented massive aerial LiDAR point clouds under the Manhattan-world assumption. Venegas et al [VAB10] reconstructed buildings by extracting Manhattan-world grammars from aerial images and generating corresponding models. Li et al [LNL16, LWN16] proposed a fully automatic approach for reconstructing Manhattan-world buildings from point clouds by partitioning the space and selecting boxes that fit point clouds best. The Manhattan-world assumption is also useful in indoor scenes reconstruction. Lee et al [LHK09] proposed a framework to recover indoor scene structure from a single image. Furukawa et al [FCSS09] and Ikehata et al [IYF15] reconstructed indoor scenes by fitting planes with MVS point clouds in orthogonal directions. Li et al [LWC*11] and Aron et al [MMBM15] refined primitive extraction results by discovering regularized relations. Recently, many deep-learning-based methods have been proposed to reconstruct Manhattan-world structures like wireframes [HWZ*18, ZQZ*19]. In our work, we focus on reconstructing urban buildings under the Manhattan-world assumption by 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 framework takes an image sequence of urban buildings as input and outputs polygonal surface models. Figure 2 shows an overview of our method, which has 4 main steps: preprocessing, plane detection and corner generation, cube fitting, and cube registration.

3.1. Preprocessing

Firstly, we input images into the SfM system COLMAP [SF16, SZPF16] to recover camera poses and generate a sparse point cloud $\mathcal{P}_0$. Then camera poses and images are inputted into the Line3D++ [HMB17] system to generate a line cloud $\mathcal{L}_0$.

Line clustering. The line cloud generated by Line3D++ is pretty noisy and broken. Using the raw line cloud $\mathcal{L}_0$ to detect planes may lead to the generation of many virtual planes and duplicate detections of the same plane with a tiny displacement, which introduces large errors and unnecessary computation. We perform a line clustering before the plane detection to get a less noisy and more consistent line cloud $\mathcal{L}$.

Sample points. Since the point cloud $\mathcal{P}_0$ is relatively sparse, we sample points from the 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 clouds.

3.2. Plane detection and corner generation

In this section, we detect planes from line cloud $\mathcal{L}$ and generate corners from detected planes. An intuitive idea is to obtain a corner by detecting 2 or 3 mutually perpendicular lines of which endpoints are close. However, the absence of any line that constitutes a corner will lead to corner detection failure. So we use a more robust method to generate corners – generating corners by planes intersecting. Compared to a line, a plane is less likely to be utterly missing since a plane is supported by multiple lines. In addition, 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 for extracting planes from point clouds usually rely on the estimated point normals [SWK07]. However, the point cloud $\mathcal{P}$ is usually too sparse and nonuniform to perform a reliable normal estimation, which often leads to poor plane detection results. Therefore we detect planes using the line cloud $\mathcal{L}$ 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 a line can support two planes at most. Each line is tagged with a state: free line or textural line or structural line. Lines that don't support any plane are free lines. Lines that support only one 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 the RANSAC method [LBM19] that extracts planes by randomly picking lines, we only extract planes that are supported by mutually parallel or perpendicular lines. This is derived from an important observation that many lines that form Manhattan-world planes are regularized. This constraint may be strict but very efficient and practical in terms of urban scenes.

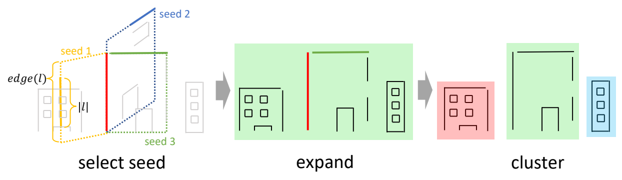

Figure 3: *Main steps of plane detection. Firstly, for a given line l_i (red line shown in the figure), the seed plane with the highest confidence (tightly wrapped by the bounding box, e.g., seed 3 shown in the figure) is selected from candidate seed planes. Then the seed plane is expanded by adding more coplanar lines. Lastly, a clustering operation is performed to break the plane into several subplanes.*

Figure 3 illustrates the main steps of plane detection. We iteratively search for a candidate plane formed by two coplanar parallel or perpendicular lines. The candidate plane is expanded by adding more coplanar parallel or perpendicular lines and then divided into several sub-planes via line clustering. The above steps are performed iteratively until no more planes are generated. We describe the technical details below. Initially, the set of detected planes $\mathcal{S}$ is set to empty and all lines are initialized as free lines. For each non-structural line (i.e., free line or textural line) l_i in $\mathcal{L}$, we iteratively search for another non-structural line l_j that is parallel or perpendicular meanwhile coplanar to l_i. Then l_i and l_j form a seed plane P_{l_i,l_j}. To avoid the duplicate detection of existing planes, if the seed plane P_{l_i,l_j} is nearly parallel and close to an existing plane in $\mathcal{S}$, then this seed plane is considered as a duplicate plane and is discarded. Next, we calculate 2 main directions for this seed plane as $d_1(P_{l_i,l_j}) = normalize(\mathbf{l_i})$ and $d_2(P_{l_i,l_j}) = normalize(\mathbf{l_i} \times \mathbf{N})$, where $\mathbf{N}$ is the plane normal. Then we calculate the bounding boxes of these 2 seed lines whose axes are aligned with the 2 main directions. We also calculate the confidence for this seed plane as $min(|l|/edge(l)), l \in \{l_i, l_j\}$, where $edge(l)$ denotes the length of the bounding box edge in the direction that is parallel to l (see Figure 3). We find all eligible seed planes for l_i and select the one with the highest confidence as the candidate plane. High confidence indicates these 2 lines are tightly wrapped by the bounding box and are empirically likely to form a reasonable plane, e.g., a window or a wall, rather than an accidentally coplanar one. Once a candidate plane is selected, we expand the plane by adding more coplanar non-structural lines that are parallel or perpendicular to the 2 main directions.

Lines in the expanded candidate plane may be pretty scattered. Thus a DBSCAN-like line clustering is applied to break P_{l_i,l_k} into several more compact meanwhile meaningful sub-planes. Compared to the original DBSCAN algorithm, the clustering targets are changed from points to lines, and the distance of 2 points is replaced by the minimal distance of 2 lines. We define the score of a plane P as the sum of the lengths of all lines supporting the plane. Planes with scores below a threshold T_p, i.e., planes supported by very few and small lines, are removed. In our implementation, T_p is set to 0.05 times the average score of all current planes $\mathcal{S}$. Finally, the remaining sub-planes are added to plane set $\mathcal{S}$ and states (i.e., free line, textural line, or structural line) of lines that support these

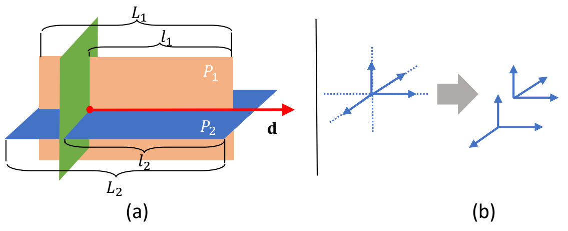

Figure 4: *Corner generation from an intersection point. We first determine valid individual directions (a), then choose all combinations of valid orthogonal directions to form corners (b).*

sub-planes are updated. We continue to add planes for each l_i until it's already a structural line or there is no candidate plane available.

Lastly, we set the boundary for each plane in $\mathcal{S}$ 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 expand the boundary of each plane a little bit by adding a margin around it to facilitate the following corner generation step.

Corner generation. Once planes $\mathcal{S}$ 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 all the boundaries of three planes are counted as valid points since the plane's boundary represents the physical range of the corresponding plane in the real world, and intersection points outside any boundary tend to be nonexistent. Three orthogonal planes intersect and generate three intersecting lines. Let intersection point be the origin and three intersecting lines be the axes, we have a coordinate with six extending directions. By choosing three orthogonal directions from these six directions, a corner can be formed. We first determine which directions are valid or not. If a direction is valid, this means we can explore the rest parts of a cube along that direction. As shown in Figure 4(a), to check whether a direction $\mathbf{d}$ is valid, let P_1 and P_2 be the two neighboring planes of $\mathbf{d}$, we calculate the confidence of this direction $\mathbf{d}$ as $p(\mathbf{d}) = \sum_{i\in\{1,2\}} min(max(\frac{l_i}{L_i}, \frac{l_i}{\lambda}), 1)$, where l_i is the distance from the origin to the boundary of P_i along direction $\mathbf{d}$, L_i is the width of P_i in direction $\mathbf{d}$, and λ is a good enough length that we are likely to accept $\mathbf{d}$ immediately when $l_i > \lambda$, which indicates that the direction $\mathbf{d}$ points to a fairly large plane area and is a promising exploration direction. 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 $(l_1 + l_2)/2$. As shown in Fig 4(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 $\mathbf{c_i} \in \mathcal{C}$ consists of one origin position $\mathbf{o_i}$, three axis directions $\mathbf{d_i^1}, \mathbf{d_i^2}, \mathbf{d_i^3}$ and their corresponding initial axis lengths l_i^1, l_i^2, l_i^3. Figure 5 shows the result of corner generation.

3.3. Cube fitting

In this part, we fit cubes to point cloud $\mathcal{P}$ to obtain cube representations of buildings. A cube representation $\mathbf{r_i}$ is a cube shape

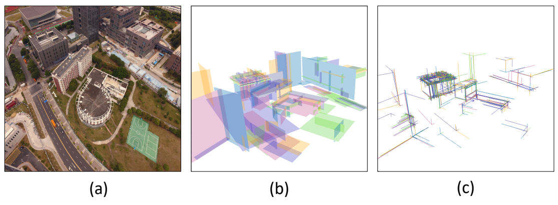

Figure 5: *The result of corner generation. (a) is the scene image, (b) is the planes detected from the line cloud, and (c) is the generated corners.*

parameterized by a fixed corner which contains one origin position $\mathbf{o^i}$ and three axis directions $\mathbf{d_i^1}, \mathbf{d_i^2}, \mathbf{d_i^3}$ and their corresponding fixed axis lengths l_i^1, l_i^2, l_i^3. The inputs of this step are point cloud $\mathcal{P}$ and corners $\mathcal{C}$ generated in the previous step. The output is cube representations of buildings $\mathcal{R}$.

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

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

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

Coverage term. This term rewards points that roughly lie on the cube's surfaces and penalizes points inside the cube, thus encourages axes to prolong to find more points on the cube, or shrink at a reasonable cube size. The smaller this term is, the better the cube fits point cloud. The coverage term is defined as

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

where $d(p,\mathbf{c_i})$ is the nearest distance from point p to the cube determined by $\mathbf{c_i}$. $d(p,\mathbf{c_i})$ is negative when p is 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 if the distance from a point to certain plane is within thresh-

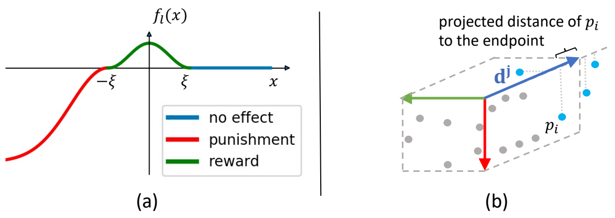

Figure 6: *(a) The overall shape of loss function $f_l(x)$. Points that roughly lie on the cube surfaces get a reward (green line). Points inside the cube get a punishment (red line). Others make no contributions (blue line). (b) An example of the calculation of $\pi(\mathbf{d^j}, N)$ in regularization term where $N = 4$, the blue points are the N points used to calculate projected distance.*

old ξ, we accept this point as a part of the plane. η and σ control the punishment. Here, a small punishment is preferred for the sparse and noisy data and a larger one is suitable for the dense point clouds. The overall shape of $f_l(x)$ is depicted in Figure 6(a). The design of $f_l(x)$ is based on an observation that most buildings are solid which means the building interior shouldn't contain points except outliers. The more outliers close to the center, the heavier punishments of these points are. We also set a convergent limit for the punishment to avoid good cube assumptions are rejected by a large punishment caused by outliers that accidentally lie on the cube center. A point that perfectly lies on the cube surfaces is counted as ONE support and thus gets a reward score of 1. The more a point away from the cube, the fewer reward it gets. Points that are away from the cube with a distance more than threshold ξ are considered neutral points that neither support nor deny the cube. Since we formulate the fitting process as a non-linear least-square optimization problem, a higher score should correspond with a smaller loss. A mapping function $\phi(x)$ is used to map scores from $[-\infty, +\infty]$ to $[0, +\infty]$ which is defined as

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

where s controls convergence speed and is set to 0.4.

Regularization term. This term prevents the axes of a corner from excessive prolonging and limits the shift of origin and axis directions from their initial poses. To calculate this term, we firstly project all the points that have positive reward (i.e., points whose distance to the cube is within ξ) on each axis $\mathbf{d^j}$ and select N points with the shortest projected distances to the axis endpoint to calculate the average projected distance $\pi(\mathbf{d^j}, N)$ (Figure 6(b)). Here a small N is recommended for a dense point cloud while a larger one (e.g., $N = 10$) for noisy data. The regularization term is defined as

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

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

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

where γ denotes the regularization term punishment and t is set to

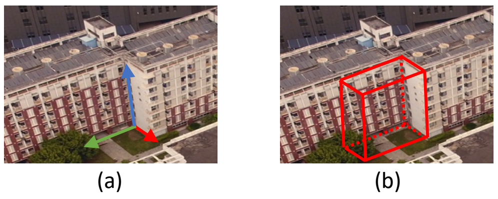

Figure 7: *An example of a virtual cube generated in the cube fitting step. (a) is the corner and (b) is the fitted cube that doesn't exist in the real world.*

0.2 by default. $f_r(x)$ has a tolerance for small excessive prolonging and significant punishment for a large one. γ controls the overall punishment strength on the excessive axis prolonging. Large γ leads to more compact cubes but also limits the exploration in the axis direction. $f_d(\mathbf{c_i}, \mathbf{c_{i0}})$ describes the difference between current parameter $\mathbf{c_i}$ and initial parameter $\mathbf{c_{i0}}$ and is defined as

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

$f_d(\mathbf{c_i}, \mathbf{c_{i0}})$ keeps the fitted cube not shift too much away from the original corner.

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

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

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 the aforementioned steps we don't apply any visibility examination, some corners that don't belong to any actual building can be generated and their corresponding fitted cubes are *virtual*, as illustrated in Figure 7. In this step, these virtual cubes are removed. For each surface s_i^j of corresponding cube of each $\mathbf{r_i} \in \mathcal{R}$, we count the numbers of points that roughly lie on s_i^j, denoted by $N_p(s_i^j)$, and the number of cameras that are able to see s_i^j, denoted by $N_c(s_i^j)$. 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 $D(\mathbf{r_i}) = \sum_{j=1}^{6} |R_p(s_i^j) - R_c(s_i^j)|$ for each $\mathbf{r_i} \in \mathcal{R}$. $\mathbf{r_i}$ that holds condition $D(\mathbf{r_i}) > N_t$ ($N_t = 8$ 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. Virtual cubes usually have more points on the back faces that are seen less by the cameras and have a large rank difference.

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.

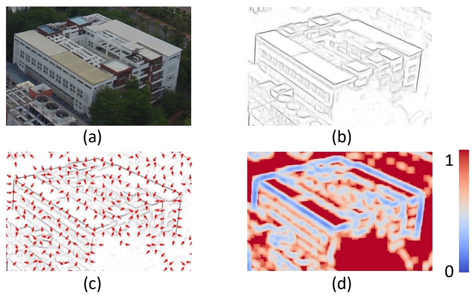

Figure 8: *The registration step. (a) the original image. (b) the detected edge. (c) an illustration of the calculated guiding vectors. (d) the calculated step map, where red denotes larger steps and blue denotes smaller steps.*

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 sizes, 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 search 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 each pixel p_i, we calculate its guiding vector as

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

where $w_{ji} = \exp(-|\mathbf{p_j p_i}|)$ 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. The closer the pixel to the edge, the smaller the step is. This property helps pixels to approach a salient edge gradually rather than oscillating around it. The moving step of each pixel p_i is calculated as

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

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

Correspondence search. To find the correspondence of an edge e on the edge map, we project e to the image I and sample points along it 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. Starting 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 $v_i + \mathbf{g_i} \cdot step(v_i)$ and repeat until a stopping point is found. Once reached a stopping point, we search for the corresponding point p_j of v_i near the stopping point that makes following score reach maximum.

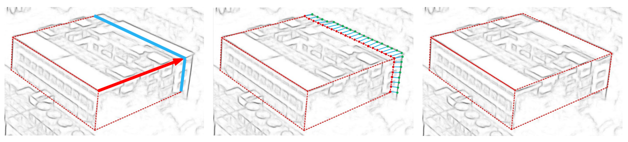

Figure 9: *Searching for corresponding points of free edges. (a) is a cube that not converged to proper size due to data missing, red arrow is the axis direction* **d** *and blue lines represent free edges of* **d**. *(b) shows the sampled points on free edges and corresponding points searched. (c) is the newly re-estimated cube size.*

$$S(v_i, p_j) = \frac{{s_j}^a}{|\mathbf{v_i p_j}|^b} \cdot \exp\Big(-\frac{(1 - |\mathbf{t_i} \cdot \mathbf{t_j}|)^2 + (1 - |\mathbf{v_i v_{i-1}}|/|\mathbf{p_i p_{i-1}}|)^2}{2\sigma^2}\Big) \tag{11}$$

Here, $\mathbf{t_i}$ is the edge orientation at v_i and is calculated as $\mathbf{t_i} = normalize(\mathbf{v_i v_{i-1}})$, $\mathbf{t_j}$ is the edge orientation at p_j and is obtained in edge detection step using structured forests [DZ13]. The positions of pixels used for calculating vector $\mathbf{v_i p_j}$ are normalized into [0,1]. 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 takes 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. We call edge e a *free edge* of axis direction **d** when e is on the perpendicular cube face that **d** points to, that is to say, the position of e is affected by the axis length on axis direction **d**, as an example shown in Figure 9(a). To estimate a new length for axis direction **d**, we find all free edges of **d** that are visible on image I and sample points uniformly on these free edges to obtain the point set $\mathcal{V}_f$, as shown in Figure 9(b). Then we calculate the axis length offset $\delta(I,\mathbf{d})$ which indicates how much the axis length should adjust to make the free edges of **d** align with matched edges on I. The offset $\delta(I,\mathbf{d})$ is calculated as

$$\delta(I,\mathbf{d}) = \frac{\sum_{v \in \mathcal{V}_f} (\mathbf{vm}(\mathbf{v}) \cdot \mathbf{d}')}{|\mathcal{V}_f|} \cdot \frac{l}{l'}, \tag{12}$$

where $\mathbf{d}'$ is the normalized projection of **d** on the image I, l is the axis length of **d** and l' is the projected length of l on image I. The new length of the axis on direction **d** is estimated as $l = l + \delta(I,\mathbf{d})$. We repeat correspondence search and new length estimating until the axis length on **d** converges. We apply the steps mentioned above for all axes of all cube representations, thus ensure all cubes are in the correct sizes. Lastly, we convert all cube representations to polygonal surface models.

4. Experiments and discussion

Dataset. We tested our framework 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 different styles and shapes.

Implementation detail. We implemented our approach using C++. In the cube fitting step (Section 3.3), we solve the optimization problem using Google Ceres solver [AMO10]. 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 others are removed to avoid redundancy. After generating polygonal models, we assign textures for each triangle from the input images to yield better visual effects.

Reconstruction results. Our approach is designed to reconstruct Manhattan-world buildings. We have tested our approach on the aforementioned datasets. The reconstruction results are shown in Figure 11. Datasets (a)-(g) are captured from the real world and datasets (f)-(g) are captured from synthetic scenes. In Table 1 we list some statistics on the tested datasets. Due to the complexity of real-world scenes, the reconstructed point clouds are relatively sparse and both line clouds and point clouds contain significant amounts of outliers. Our method still roughly reconstructed these buildings and yielded good approximations by leveraging the structural information from line clouds and the range information from point clouds. In Figure 11(e), 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 Figure 12, we show the details of our reconstructed results. In the nasty cases shown in the top view Figure 12 a(1), the building highlighted in the red box, whose back faces are entirely missing, still got reconstructed. In the dataset *IT lab*, though the building (Figure 12 a(2)) is not in a cuboid shape 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 (Figure 12 a(3)), our approach failed in handling non-Manhattan-world buildings in Figure 12 a(4), where 2 cylinder shape buildings are completely missing. In Table 1, we show the statistics of our reconstruction results. Our reconstruction results have low face numbers while preserving the shape features of buildings, which is very suitable for large-scale urban reconstruction.

Plane detection. Figure 10 shows the results of various plane detection methods. Due to the sparsity of the points, only a few planes are detected from the point cloud and some planes are completely missing (Figure 10(b)). By using line cloud as input, the RANSAC method is able to detect more planes (Figure 10(c)). However, these unbound planes are unrecognizable and may intersect and generate a significantly large number of meaningless corners. Our plane clustering step limits the boundaries of the planes and fits planes to their actual sizes thus reduces the generation of redundant corners (Figure 10(d)). Besides, the Manhattan-world regularization constraint introduced in our method, i.e., only parallel or perpendicular lines are selected to support planes, can reduce the detection of insignificant and virtual planes (Figure 10(e)). While planes that contain many lines can be easily detected using the random seed plane picking mechanism, planes supported by a few lines are always missing in the RANSAC way. Our confidence-based seed

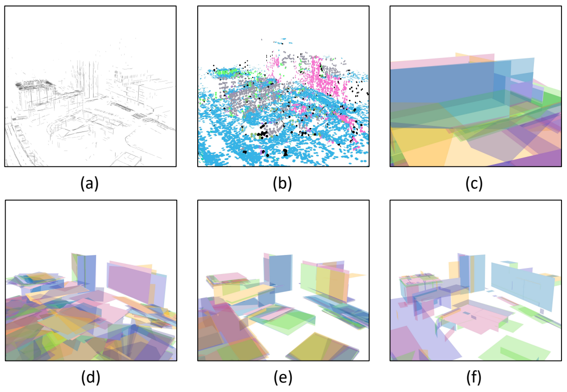

Figure 10: *Plane detection results of different methods. (a) The input line cloud. (b) planes detected by RANSAC from the point cloud. (c) planes detected by RANSAC. (d) planes detected by RANSAC with plane clustering. (e) planes detected by RANSAC with plane clustering and Manhattan-world regularization constraint. (f) planes detected by our method.*

plane selection strategy ensures even the small planes can be detected efficiently (Figure 10(f)).

Robustness of cube fitting. The critical step of our approach is cube fitting, and the outputs of this step largely dominate the result of reconstruction. We commit several experiments on how effective and robust our cube fitting step is. As shown in Figure 11, most cubes are fitted to the correct size in most cases. For a given corner, our objective function encourages the corner to explore the rest part and prevent over-extending at the same time. Recall that in the corner generation step (Section 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. Though our method does 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 a desirable size.

Effect of registration. In most cases, our registration step can adjust cubes into the proper size. Since our registration algorithm is running in a greedy strategy, cubes might be aligned to close but incorrect edges. Another factor that impacts registration results is the radius used to compute the guiding vector and the step. A larger radius leads to a larger searching range and more robust corresponding searching, meanwhile, the computation time also increases.

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. Table 1 illustrates the computation times on various datasets. The most time-consuming part of our approach is cube fitting (Section 3.3) which mostly takes about 70% of total running time. The fitting time is linearly related

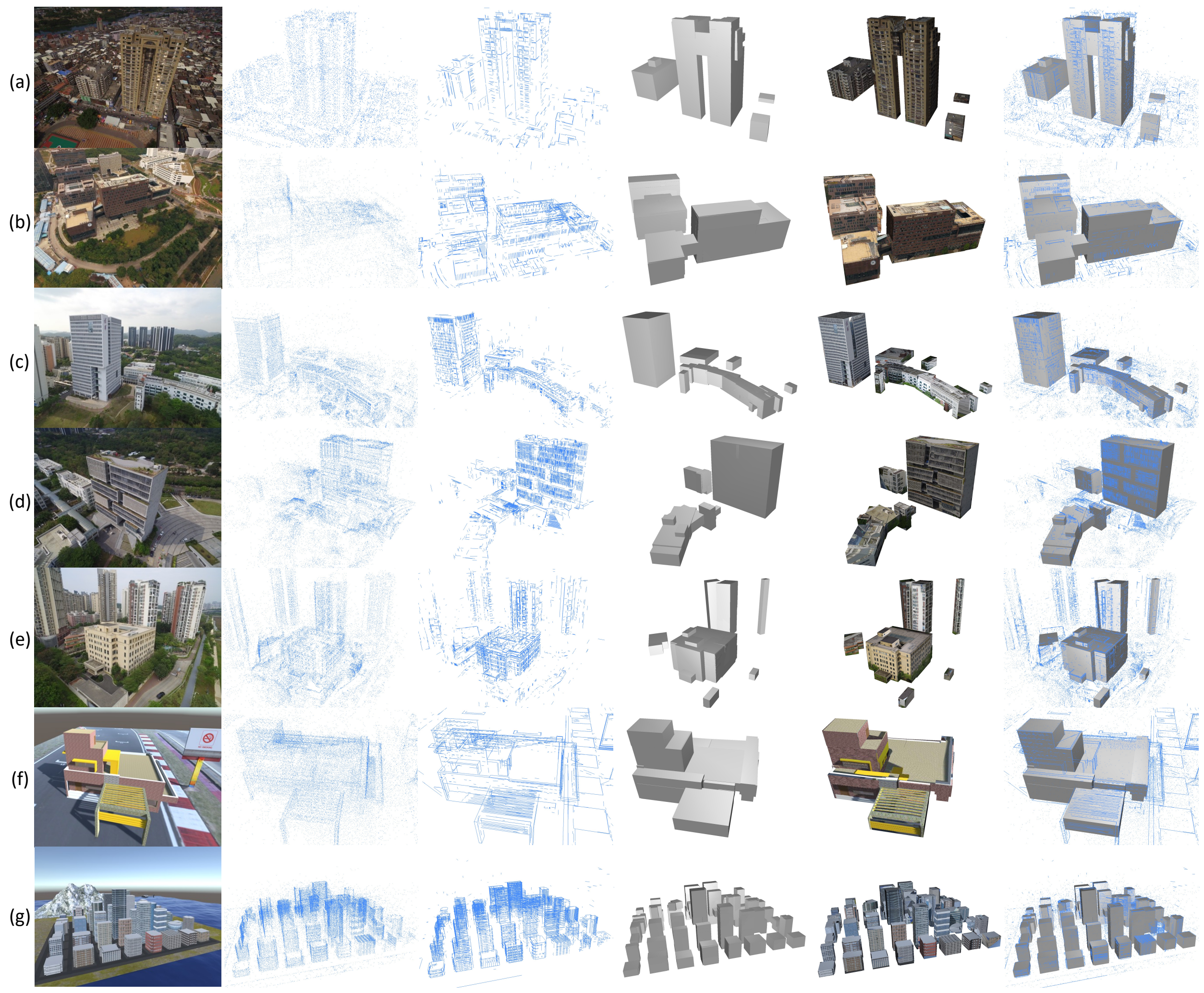

Figure 11: *Reconstruction results on various datasets. From left to right: reference image, point cloud, line cloud, fitted cubes, fitted cubes with textures, fitted cubes overlaid with the point cloud.*

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 thus fewer cubes and shorter running time got in return. So it requires users to adjust parameters according to the dataset to strike a balance between quality and speed. In the registration step, most time is spent on the edge detection and computation for guiding vector and step map. 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 from a few minutes to several hours depending on the number of images.

Comparison with other methods. Figure 13 shows the comparison results with 6 mainstream methods on several datasets. We first follow the typical photogrammetry pipeline using SfM technique to recover camera poses and sparse point cloud. Then we run MVS to recover dense point cloud. Screened Poisson reconstruction [KH13] is applied to reconstruct surfaces from both the sparse (Figure 13(a)) and the dense point cloud (Figure 13(b)). A mesh simplification using quadric metric [Hop99] (Figure 13(c)) is performed on the model reconstructed from the dense point cloud to reduce face number to 0.01 times of original number. The sparse point cloud is inputted to other two methods Polyfit [NW17] (Figure 13(d)) and Li et al [LWN16] (Figure 13(e)). The line cloud is inputted to one line-based surface reconstruction method proposed by Langlois et al [LBM19] (Figure 13(f)). The Screened Poisson reconstruction on the sparse point cloud is pretty bumpy and lacks details. Screened Poisson reconstruction on the dense point cloud recovered significant details, and yet the models contain millions of triangles and occupy large storage. While the simplified model con-

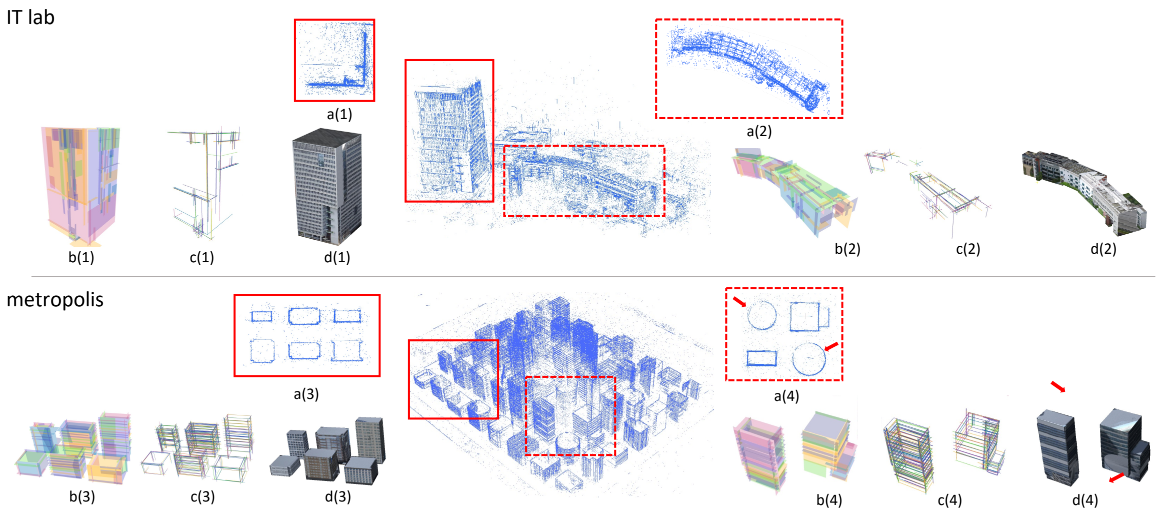

Figure 12: *Reconstruction details on datasets IT lab and metropolis. The image in the middle of each row is the sparse point cloud and line cloud. a(1-4) are the top view of the target buildings in the red boxes, b(1-4) are detected planes, c(1-4) are generated corners and d(1-4) are reconstructed models.*

tains fewer triangles, some features of buildings are severely lost. Other lightweight reconstruction methods Polyfit [NW17] (Figure 13(d)) and Li et al [LWN16] (Figure 13(e)) that heavily rely on plane detection results, failed to reconstruct faithful models due to data missing and unreliable plane detection from point clouds. Compare to methods merely using point cloud, line-based methods are more robust in the plane detection (Figure 13(f-g)). However, Langlois et al [LBM19] don't take plane's boundary into consideration. These planes partition space into massive cells, which not only takes a long time to process (usually takes over 30min) but also produces artifact cells easily . Besides, this method is still vulnerable to the missing of key planes (e.g., back faces of buildings). Thanks to fitting cubes from corners, our approach can reconstruct Manhattan-world buildings even when only a few planes are available, as shown in Figure 13(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 urban reconstruction. Statistics on face number and running time of the results in Figure 13 are available in Table 2.

Limitation. The result of our approach may heavily rely on the outputs of COLMAP and Line3D++ systems. Therefore, the quality of point clouds and line clouds 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 Figure 12a(4), wireframe structures), limiting our approach's applicability.

5. Conclusion and future work

We introduced a novel approach to address the challenging task that reconstructing Manhattan-world buildings from low-quality data. Our approach extracts planes from the line cloud and generates corners where we fit cubes to the point cloud. Lastly, lightweight models are generated after a registration step. Experiments on both real and synthetic datasets show that our approach is robust and efficient in reconstructing cuboid-shaped buildings from the incomplete and sparse point clouds. Our approach breaks the limitation of the previous approaches that they are vulnerable to plane missing.

In future work, we would like to explore the possibility of fitting more primitives and reconstructing buildings in arbitrary shapes. Another possible extension is using more semantic information obtained from images to improve the reconstruction results.

Table 1: *Statistics on the datasets in Figure 11*

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

Acknowledgements

This work was partially supported by NSFC (61972388), Shenzhen Basic Research Program (JCYJ20180507182222355), the Leading Talents of Guangdong Program (00201509), and the CAS grant (GJHZ1862).

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.

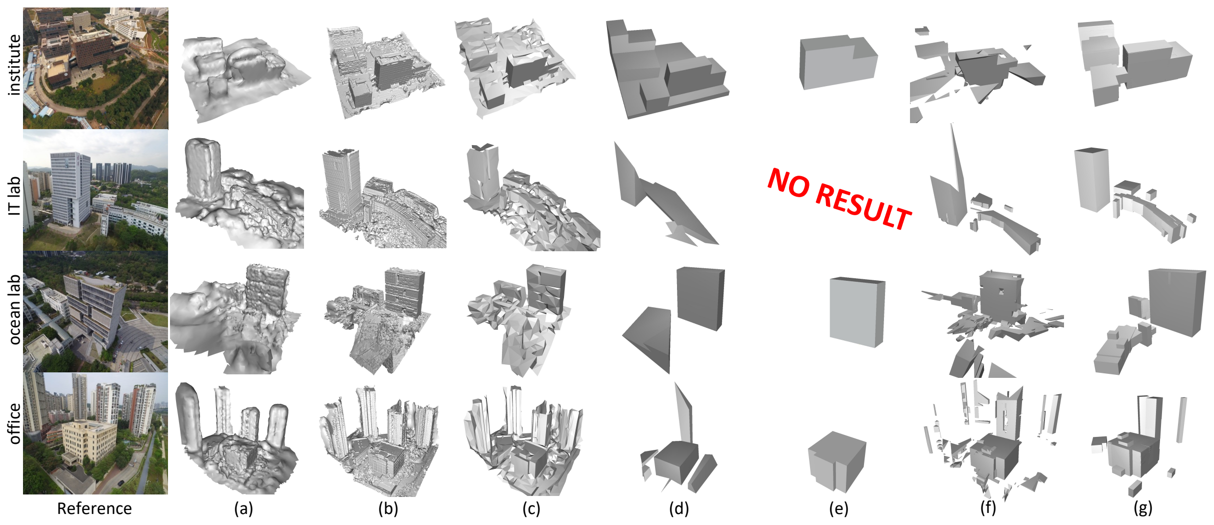

Figure 13: *Comparison with 6 methods on several datasets. (a) sparse point cloud + Screened Poisson [KH13]. (b) dense point cloud + Screened Poisson [KH13] then (c) simplified using quadric metric [Hop99]. (d) Polyfit [NW17]. (e) Li et al [LWN16]. (f) Langlois et al [LBM19]. (g) ours.*

Table 2: *Face number and full running time of methods shown in Figure 13. (a) sparse point cloud + Screened Poisson [KH13]. (b) dense point cloud + Screened Poisson [KH13] then (c) simplified using quadric metric [Hop99]. (d) Polyfit [NW17]. (e) Li et al [LWN16]. (f) Langlois et al [LBM19]. (g) 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)	9.2K	108min	8.8K	62min	10K	101min	9.9K	81min
(g)	768	5.5min	924	6.9min	612	9min	2.3K	13.5min

[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. *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.

[FL20] Fang H., Lafarge F.: Connect-and-slice: an hybrid approach for reconstructing 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 13490–13498.

[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.

[HFB16] Holzmann T., Fraundorfer F., Bischof H.: Regularized 3d modeling from noisy building reconstructions. In *2016 Fourth International Conference on 3D Vision (3DV)* (2016), IEEE, pp. 528–536.

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

[HMFB18] HOLZMANN T., MAURER M., FRAUNDORFER F., BISCHOF H.: Semantically aware urban 3d reconstruction with plane-based regularization. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 468–483.

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

[HOP*17] HOLZMANN T., OSWALD M. R., POLLEFEYS M., FRAUNDORFER F., BISCHOF H.: Plane-based surface regularization for urban 3d construction. In *Proceedings 28th British Machine Vision Conference, 2017 (BMVC)* (2017), pp. 1–9.

[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.

[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.

[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.

[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.

[ZSGH18] ZHU L., SHEN S., GAO X., HU Z.: Large scale urban scene modeling from mvs meshes. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 614–629.