Connect-and-Slice: an hybrid approach for reconstructing 3D objects

Hao Fang Florent Lafarge
Université Côte d'Azur, Inria
Firstname.Lastname@inria.fr

Abstract

Converting point clouds generated by Laser scanning, multiview stereo imagery or depth cameras into compact polygon meshes is a challenging problem in vision. Existing methods are either robust to imperfect data or scalable, but rarely both. In this paper, we address this issue with an hybrid method that successively connects and slices planes detected from 3D data. The core idea consists in constructing an efficient and compact partitioning data structure. The later is i) spatially-adaptive in the sense that a plane slices a restricted number of relevant planes only, and ii) composed of components with different structural meaning resulting from a preliminary analysis of the plane connectivity. Our experiments on a variety of objects and sensors show the versatility of our approach as well as its competitiveness with respect to existing methods.

1. Introduction

Reconstructing surfaces from 3D data is still one of the biggest challenges in computer vision. Most of existing methods, as the popular Poisson algorithm [15], are designed to approximate free-form shapes by dense triangular meshes. When the observed scenes contained geometric regularities, typically man-made objects and urban environments, these methods do not provide optimal results in terms of storage, rendering or editing capacity [2]. One prefers representing such scenes by more compact and structure-aware Computer-Aided Design style models, *i.e.* with concise polygon meshes in which each facet corresponds to a large polygon [4].

Reconstructing concise polygon meshes is usually operated in two steps. First, *planar primitives* are detected from input 3D data. A planar primitive is defined as the association of a plane and a subset of input data called inliers from which the plane has been fitted to. Primitives are disconnected from each others and constitute an intermediate representation between input 3D data and the output mesh. The second step then consists in assembling primitives into a surface mesh. This step constitutes the difficult part of the problem. One strategy is to connect the primitives using proximity and structural considerations [1, 8, 17, 29]. Despite being fast and scalable, this solution is not robust to defect-laden data, in particular when primitives are over- or under-detected or when erroneous connections between primitives exist. A more robust strategy consists in slicing a 3D domain by extending the primitives. This leads to the creation of a partition of polyhedral cells or polygonal facets [5, 7, 24, 34]. The surface is then extracted by labeling the cells as inside or outside the surface, or equivalently, by selecting facets to be part of the surface. Because each primitive exhaustively slices all the others, this solution is more robust to defect-laden data than the first strategy. However, its main shortcoming is the computational burden for slicing the primitives into a partition of atomic surface and volume elements, with typically unreasonable timing and memory issues when more than one hundred primitives are handled. None of these two strategies are both robust and scalable.

We address this problem with an hybrid method that successively connects and slices primitives in a robust and scalable manner. The method is built on three important technical ingredients.

First, our algorithm has a preliminary step that analyzes the connectivity of primitives in order to search for structurally-valid surface components. This allows us to quickly process a part of the input primitives and solve obvious primitive assembling situations.

Second, we construct a more flexible and lighter partition data structure than the slicing methods. Our data structure is spatially-adaptive in the sense that a primitive slices a restricted number of relevant primitives based on spatial proximity considerations. Moreover, its atomic elements have different structural meanings that will guide the extraction of the output surface.

Third, we propose a surface extraction mechanism that operates from such an irregular data-structure in which cells are not necessarily convex and free of intersection with other cells. In particular, we do not measure data fidelity to input 3D data directly, but instead to primitives: it allows output models not to suffer from artifacts frequently found with existing methods.

We demonstrate the potential of our algorithm in terms of flexibility, robustness and scalability on different types of objects, going from buildings to mechanical pieces through even free-form shapes. In particular, we show our algorithm is faster and more scalable than state-of-the-art methods by a significant margin.

2. Related works

Primitive detection. Detecting geometric primitives from 3D data is an instance of the general problem of fitting mathematical models to data. Region growing [21, 28] and Ransac [31] mechanisms constitute the most popular methods for detecting planes from 3D data. Some recent works detect and regularize primitives according to geometric relationships such as parallelism or orthogonality, either sequentially [20, 25] or simultaneously [22, 26]. Such regularities typically allow the complexity of the subsequent assembling step to be reduced. Other recent methods also allow the extraction of geometric primitives at key abstraction levels of the observed objects [12]. Although these methods work well in practice, primitive detection remains an ill-posed problem with no guarantee that the output configurations adequately describe the observed object [9].

Connectivity-based methods. Analyzing a connectivity graph to detect and link points intersecting plane triples [8, 29, 33] usually works well when the correct connectivity between primitives can be recovered. To be robust to challenging data, one interactive solution is to automatically snap primitives when the connectivity is obvious, and let the user complete the output surface for the conflicting situations [1]. Another solution consists in mixing polyhedral surface components with flexible free-form patches [17, 16]. Such a representation however does not offer the level of compactness and simplicity of CAD style models. Despite being fast, connectivity-based methods suffer from a lack of robustness to defect-laden data, in particular to over- and under-detection of primitives and erroneous connections between primitives. Our approach exploits some principles of these methods as a preliminary step to quickly solve obvious plane assembling situations and lighten the time-consuming slicing operations.

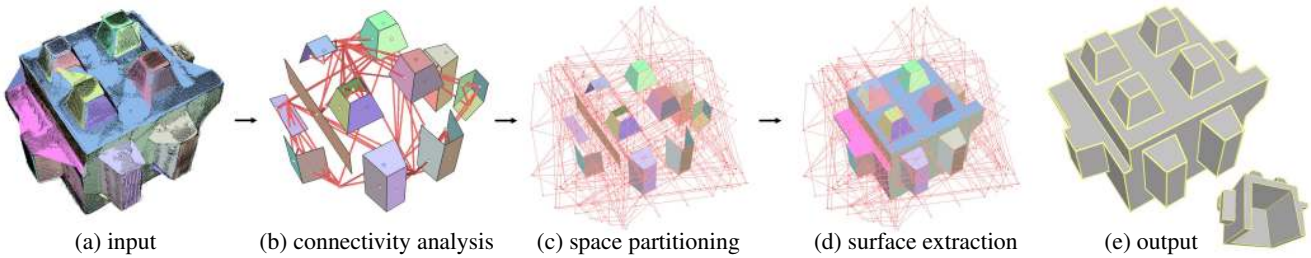
Slicing-based methods. The core of these methods consists in partitioning a 3D domain by extending primitives. The partitioning data-structure is typically a 3D tesselation of polyhedral cells, which are themselves composed of polygonal facets. The output surface is then extracted by selecting a subset of facets from the tesselation. Because each primitive naively slices all the others, such a data-structure, also called a plane arrangement [23], is particularly dense and time-consuming to compute. Some methods decompose the slicing operations into spatial blocks [7, 5]. Such piecewise partitions increase scalability by a good margin. However it creates geometric artifacts as blocks often do not align well with data. These methods also add artificial primitives along vertical and horizontal axes in the partition to be more robust to missing primitives, assuming the observed object aligns with these arbitrary directions. A discrete partitioning [30, 34] that avoids computing the exact geometry of the whole partition is a less costly option, but also engenders geometric artifacts when the discretization is not fine enough. Another possible solution consists in filtering and simplifying the input set of primitives to remove redundant planes and reduce the computational burden of the slicing operations [24]. Although these methods offer a good robustness to imperfect configurations of primitives, they do not scale well. Our approach proposes two key ingredients to solve this issue: a new light and spatially-adaptive partitioning data-structure and a preliminary connectivity analysis that reduces the number of primitives to be processed during slicing operations.

Methods with geometric assumptions. Some works also exploit strong geometric assumptions. The Manhattan-World assumption [10] enforces planes to follow only three orthogonal directions. This assumption reduces both the geometry of output 3D models and the solution space to explore. Such an assumption is interesting for modeling buildings [19] and approximating shapes very coarsely [14]. Another frequent geometric assumption is to restrict the output surface to a disk-topology with a 2.5D view-dependent representation. This is well adapted to reconstruct buildings from airborne data [35, 36, 27, 18], facades from street-side data [3], and indoor scenes from images [6]. Although these assumptions efficiently reduce the solution space in general, their use is restricted to specific applications. To the contrary, our approach remains generic.

3. Overview

Our algorithm takes as input a point cloud. It can also start from a dense triangle mesh. The algorithm returns as output a polygonal mesh which is 2d-manifold, watertight and intersection-free. Optionally, the user can relax these geometric guarantees. We first extract from the input 3D data a set of primitives by standard methods [28, 31]. For each detected primitive, we compute (i) a rough approximation of its boundaries using α-shape [11], and (ii) an oriented 2D bounding box, *i.e.* the smallest rectangle lying on the detected plane that contains all its projected inliers. We call a $\epsilon-$*bounding box*, the oriented 2D bounding box scaled up by an offset ϵ.

The algorithm operates in three steps illustrated in Figure 1. First, the connectivity relations between primitives



(a) input (b) connectivity analysis (c) space partitioning (d) surface extraction (e) output

Figure 1. Overview. Our algorithm starts from a point cloud and a set of primitives whose α-shapes are represented by colored polygons (a). By analyzing the connectivity graph of primitives (see red edges), we extract some structurally-valid facets represented by colored polygons with black edges (b). This quick connectivity analysis allows us to treat 35 of the 60 primitives of this model. We then build the partitioning data-structure (see the pink wireframe) by slicing the spatially-close unprocessed primitives while embedding the structurally-valid facets found in the previous step (c). The last step selects a subset of polygonal facets from the partition data-structure (d). The output is a 2d-manifold polygon mesh in which each facet is a polygon supported by one of the primitives (e).

are analyzed in order to search for structurally-valid surface components. This step, presented in Section 4, allows us to quickly process a part of the input primitives and solve obvious assembling situations before slicing operations. We then build the partitioning data-structure in Section 5 by slicing the spatially-close unprocessed primitives while embedding the structurally-valid components found in the previous step. Finally, the output surface is recovered by selecting a subset of polygonal facets from the partition data-structure using an energy minimization formulation presented in Section 6.

4. Connectivity analysis

The objective of the first step is to quickly solve obvious local assemblings of some primitives by analyzing the connectivity relations between them.

We define the notion of strong connectivity for characterizing primitives that are spatially very close. When detected from point clouds, two primitives are said *strongly-connected* if at least two inlier points fitted each to one of the two primitives are mutual neighbors in the k-nearest neighbor graph of the input points. In case of input meshes, two primitives are strongly-connected if at least one inlier facet from the first primitive share an edge with an inlier facet of the second primitive. We operate our analysis on the *connectivity graph* where each node is associated with a primitive, and each edge with a pair of strongly-connected primitives. From real-world data, such a graph usually contains errors with missing and invalid connections. Our strategy is to search for structurally-valid facets in this graph.

Extracting corners, creases and border polygons. We first detect all the 3-cycles in the connectivity graph, *i.e.* triples of primitives that are mutually connected. The point located at the intersection of the three corresponding planes is called a *corner* if it is close from the α-shapes of the three primitives. In practice, we impose a maximal distance of 5% of the 3D bounding box diagonal. This condition allows us to ignore a corner positioned far away from its primitives, which typically occurs when primitives are nearly parallel. We then detect *creases*, *i.e.* the line-segments linking pairs of corners which have exactly two primitives in common. Finally, we extract *border polygons* of each primitive, *i.e.* the simple cycles of creases lying on the primitive.

Extracting structural facets. A primitive with border polygons hosts a facet which is potentially a good candidate to be part of the output surface. In presence of one border polygon, this facet is simply defined as its inside surface. When two border polygons are nested, ie one of these polygons is contained in the second one, we define the facet as the surface in between the two polygons. When border polygons intersect, we do not create facet to avoid non-manifold degeneracies. Such a facet is called a *structural facet* if two conditions are respected:

• *Data consistency*: the facet must strongly overlap with the α-shape of the primitive,

• *Structural validity*: all the creases lying on a primitive must belong to the border polygons of that primitive.

The first condition checks whether the facet is well recovered by the α-shape of the primitive. In practice, we impose an overlapping ratio between the facet and the α-shape of the primitive higher than a threshold τ set to 0.9 in our experiments. The second condition guarantees that the facet is unique and connect in a 2d-manifold way with facets induced by the other primitives.

Structural facets connect between each others to form 2d-manifold polyhedral surface components that partially describe the observed object. The border edges of these components necessarily lie on the remaining primitives: we call them *anchor edges*. We impose the structural facets to be part of the final output mesh and discard their corresponding primitives in the following steps. We denote by $\mathcal{P}$, the set of remaining primitives.

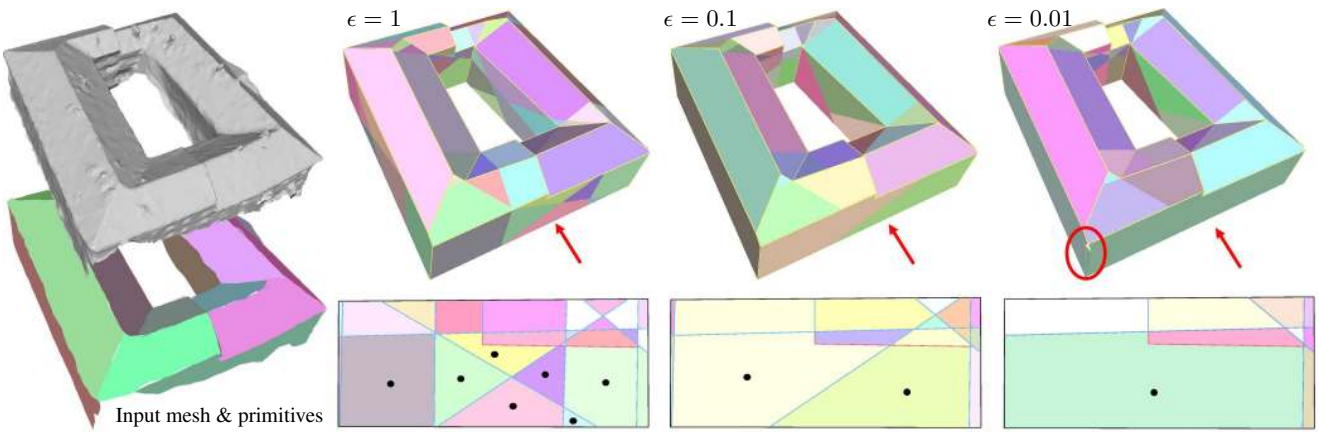
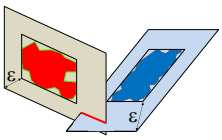


Figure 2. Soft-connectivity. When all the remaining primitives intersect with each others ($\epsilon = 1$), the 2D partition of the front facade of the building is over-fragmented (see colored polygons on the top right frame with the anchor edges in red and the intersection lines in blue; polygons with a black dot indicate they belong to the output surface on the left). Decreasing ϵ reduces the complexity of 2D partitions. In presence of holes in the input mesh, primitive intersections can be missed when ϵ is too low (see the missing intersection between the front and left facade in the case where $\epsilon = 0.01$). ϵ is expressed as a ratio of the bounding box diagonal of the scene.

5. Space partitioning

Primitive slicing is usually performed in a greedy manner in the literature. Typically, one first computes the *slicing domain* of each primitive, *i.e.* the polygon lying on the primitive plane and bounded by the 3D bounding box of the observed object. Then, the 3D bounding box is divided into polyhedra by inserting one per one each slicing domain in an arbitrary order: the first slicing domain splits the 3D bounding box into two polyhedra, the second slicing domain typically splits the two polyhedra into four polyhedra, etc. Because such a slicing strategy considers the intersection of all pairs of slicing domains, the number of polyhedra increases exponentially with respect to the number of primitives. In practice, only a small portion of these intersections is relevant. To reduce the computational burden of this operation, we restrict the pairs of primitives to be sliced. We define the notion of soft-connectivity to avoid intersecting slicing domains whose primitives are not close enough. Two primitives are said *softly-connected* if their ϵ-bounding boxes intersect inside the 3D bounding box of the observed object (see inset). This connectivity relationship is fast to compute and less restrictive than strong-connectivity.



In practice, we first intersect the slicing domains of softly-connected primitives to form a 2D partitions of polygonal facets. If anchor edges lie on primitives, they are inserted into the corresponding 2D partitions. The anchor edges whose extremities are not connected to other anchor edges are extended until meeting an intersection line or the border of the slicing domain. We finally split edges that cross anchor edges. Figure 3 illustrates these different slicing operations. Note that such a strategy generates a set of polygonal facets which can eventually intersect between each others without necessarily sharing an edge.

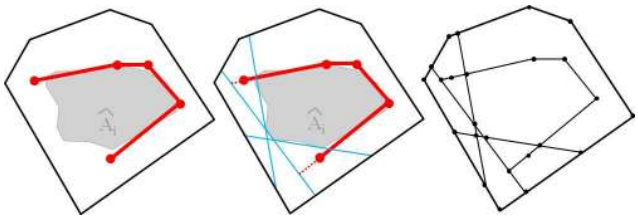


Figure 3. Slicing operations. Left: we first compute the slicing domain (back lines) of primitive i and insert the anchor edges associated with this primitive (red segments). Middle: we then insert line-segments defined as the intersection with the slicing domains of softly connected primitives (blue lines) and extend anchor edges whose extremities are not connected to other anchor edges (dashed red lines). Right: the intersections of these different lines and edges give the 2D partition of facets associated with primitive i.

The value of ϵ controls the complexity of the partitioning data-structure, as illustrated in Figure 2. Choosing a low ϵ value gives a set of light 2D partitions, low running time and low memory consumption, but is less likely to be robust to missing data.

We denote by $\mathcal{F}$, the set of polygonal facets contained in all the 2D partitions, and $\mathcal{E}$, the set of edges. Note that edges are typically adjacent to four facets, except in case of anchor edges and rare situations where at least three primitives intersect along the same line.

6. Surface extraction

Contrary to existing methods [7, 24, 34], our set of polygonal facets $\mathcal{F}$ and edges $\mathcal{E}$ does not necessarily constitute a regular partition of polyhedral cells in the sense that cells can overlap and polygonal facets can intersect between each others. Traditional polyhedron labeling methods by Graph-Cut [7, 34] being not applicable to our partition, we adopt a more flexible facet selection approach inspired by the integer programming formulation of [24]. In particular, such a formulation allows us to impose some geometric constraints on the expected solution, *e.g.* the intersection-free guarantee. Contrary to [24], our energy model (i) operates on an irregular partition that requires additional linear constraints and (ii) does not directly depend on time-consuming measurements to input data.

We denote by $x_i = \{0, 1\}$ the activation state of facet $i \in \mathcal{F}$, and by $\mathbf{x} = (x_i)_{i \in \mathcal{F}}$ a configuration of activation states for all facets in $\mathcal{F}$. The set of active facets, *i.e.* so that $x_i = 1$, constitutes the facets of the output surface.

Energy. We measure the quality of a configuration $\mathbf{x}$ with a two-term energy of the form

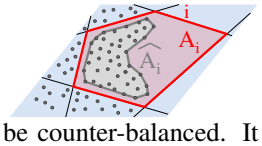
$$U(\mathbf{x}) = (1 - \lambda)D(\mathbf{x}) + \lambda V(\mathbf{x}) \tag{1}$$

where $D(\mathbf{x})$ and $V(\mathbf{x})$ are terms living in $[0, 1]$ measuring data consistency and surface complexity. $\lambda \in [0, 1]$ is parameters balancing these two terms.

The linear term $D(\mathbf{x})$ encourages facets recovered by inliers to be activated

$$D(\mathbf{x}) = (1-\beta)\left(\sum_{i \in \mathcal{F}} \frac{A_i - \widehat{A_i}}{A - \widehat{A}}\, x_i\right) + \beta\left(1 - \sum_{i \in \mathcal{F}} \frac{A_i}{A}\, x_i\right) \tag{2}$$

where A_i is the area of facet i, $\widehat{A_i}$ the area of α-shape of the inliers falling in facet i, A the sum of areas of all facets, $\widehat{A}$ the sum of areas of all primitive α-shapes. The first part of the expression encourages the activation of facets homogeneously recovered by data. The second part of the expression is required to penalize the non-activation of facets. β is a parameter living in $[0, 1]$ that allows these two opposite forces to be counter-balanced. It acts as a trade-off between local correctness of facets and global coverage. In our experiments, we set β to 0.5, except for inputs with missing data where the value is increased to 0.7.



The quadratic term $V(\mathbf{x})$ favors low complexity output surface in a similar way than the one proposed by [24]

$$V(\mathbf{x}) = \frac{1}{|\mathcal{E}_\sim|} \sum_{(i,j) \in \mathcal{E}_\sim} \mathbb{1}_{\{i \bowtie j\}} x_i x_j \tag{3}$$

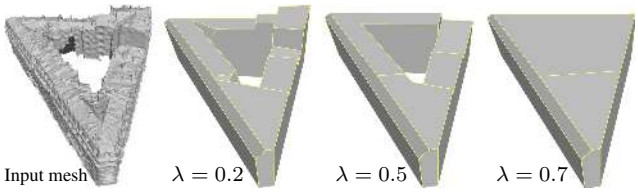


Figure 4. Impact of parameter λ. Increasing λ reduces the complexity of the output model. At $\lambda = 0.7$, only a small portion of the 21 detected primitives plays a role in the output model. Note how the inner courtyard disappears.

where $\mathcal{E}_\sim$ is the set of pairs of facets in $\mathcal{F}$ that share an edge, $|\mathcal{E}_\sim|$ its cardinality, $\mathbb{1}_{\cdot}$ the Heaviside function, and $i \bowtie j$ the geometric relationship which is true when facets i and j are not coplanar. This term favors output surfaces with large facets by penalizing the presence of creases, as illustrated in Figure 4.

Constraints. We introduce three linear constraints in order to impose some geometric guarantees on the output surface.

• *Structural constraint* imposes the structural facets to be active, *i.e.* part of the output surface (Eq. 4):

$$x_i = 1, \quad \forall i \in \mathcal{F}_s \tag{4}$$

where $\mathcal{F}_s$ corresponds to the set of structural facets.

• *2d-manifold and watertight constraint* traditionally imposes each edge to be shared by zero or two facets. As input points have often missing parts on their 3D bounding box (see for instance *Church* and *Face* in Figure 7), we relax the watertight constraint on edges lying on the 3D bounding box. This allows us to avoid either shrinking the output surface or increasing computational complexity by adding facets of the six sides of the 3D bounding box in $\mathcal{F}$. Note that, for a strict watertightness, such border edges can be easily filled in as post-processing. We formulate this constraint as

$$\sum_{k \in F_e} x_k = 0 \text{ or } 1, \quad \forall e \in \mathcal{E}_{border} \tag{5}$$

$$\sum_{k \in F_e} x_k = 0 \text{ or } 2, \quad \forall e \in \mathcal{E}_{\overline{border}} \tag{6}$$

where F_e is the set of facets adjacent to edge e, $\mathcal{E}_{border}$ is the set of edges lying on one of the six sides of the 3D bounding box, and $\mathcal{E}_{\overline{border}}$ its complementary set in $\mathcal{E}$.

• *Intersection-free constraint.* As $\mathcal{F}$ can contain facets that intersect, we impose such pairs not to be active at the same time:

$$x_i + x_j = 0 \text{ or } 1 \;\; \forall (i, j) \in \mathcal{I} \tag{7}$$

where $\mathcal{I}$ is the set of pairs of facets in $\mathcal{F}$ that intersect. Note that when ϵ is set to 1, this constraint is not necessary as the

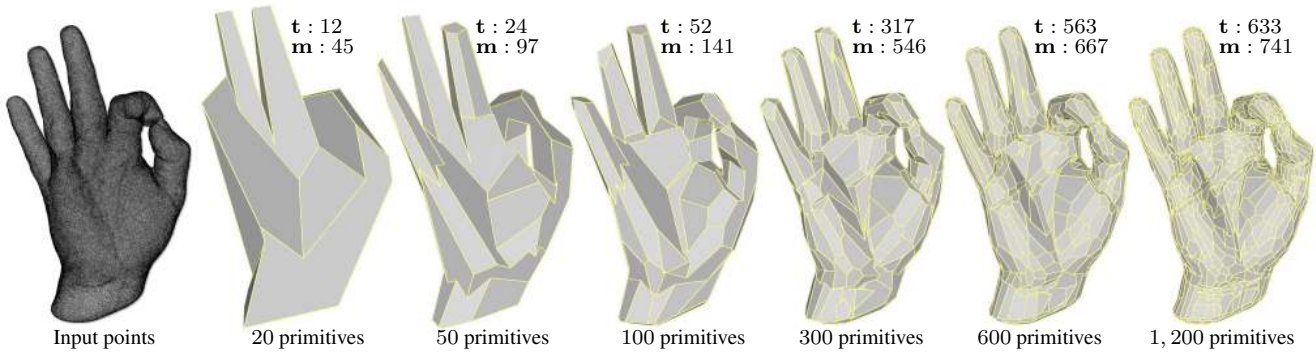


Figure 5. Reconstruction of a free-form object at different levels of details. Our algorithm can approximate free-form objects by piecewise planar representations. Detecting primitives with an increasing precision gives us a set of polygon meshes at different levels of detail. $\mathbf{t}$ and $\mathbf{m}$ refer to running time (in second) and memory peak (in MB) respectively.

partition is guaranteed to be free of intersecting facets by construction.

Optimization. We search for the configuration $\mathbf{x}$ that minimizes the energy U while imposing Eq. 4, 5, 6 and 7 to be true. We solve this quadratic optimization problem under linear constraints using a standard integer programming library [13]. In practice, we turn it into a linear optimization problem by inserting the extra-variables $y_k = x_i x_j$.

7. Experiments

The algorithm has been implemented in C++, using the Computational Geometry Algorithms Library [32] which provides the basic geometric tools for mesh-data structures.

Flexibility. The algorithm has been tested on a variety of data from urban and indoor structures to mechanical pieces through free-form objects. Although it performs best on piecewise-planar objects and scenes, our algorithm can handle a large number of primitives necessary to approximate free-form shapes at different levels of detail, as illustrated in Figure 5. Different types of acquisition systems have been used to generate the datasets, including Laser, *e.g. Euler* and *Hand*, multi-view stereo, *e.g. Cottage* and *Building block*, and Kinect, *e.g. Rubbish bin* and *Couch*. Because the data term of our energy measures surface consistency with respect to primitives directly, our algorithm is weakly affected by the type of acquisition systems as long as primitives fit well to input data.

Robustness. Our algorithm is relatively robust to noise as long as primitives can be decently detected. When data contain holes and missing areas as in *Euler* in Figure 7, the connectivity analysis typically returns few structural facets, but the subsequent slicing mechanism achieves to fill in the missing areas. In practice, our algorithm cannot handle large holes for which an extension of detected primitives is not sufficient to describe the missing part. The exploitation of defect-laden data typically requires to increase the value of ϵ from 0.1 (default value) to typically 0.3.

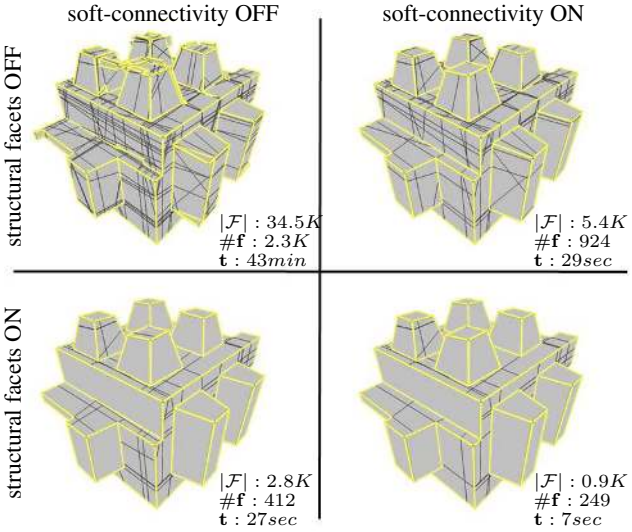


Figure 6. Ablation study. Without soft-connectivity ($\epsilon = 1$) and connectivity analysis step, the number of facets $|\mathcal{F}|$ in the partition is huge, leading to high running times $\mathbf{t}$ and a complex output surface likely to contain artifacts (top left). Activating soft-connectivity ($\epsilon = 0.1$) reduces the complexity of the partition while improving the quality of the output surface (top right). When a fair number of structural facets are detected during the connectivity analysis step (here 35 structural facets over 60 primitives), the partition is even more compact as only a part of primitives are sliced (bottom right). The facets in the output sufaces are represented through yellow (border) and black (internal) edges, and their number is given by $\#\mathbf{f}$.

Performance. Our algorithm is designed to be scalable and fast through two keys ingredients: a connectivity analysis to quickly process obvious assembling situations, and a slicing mechanism operated on softly-connected

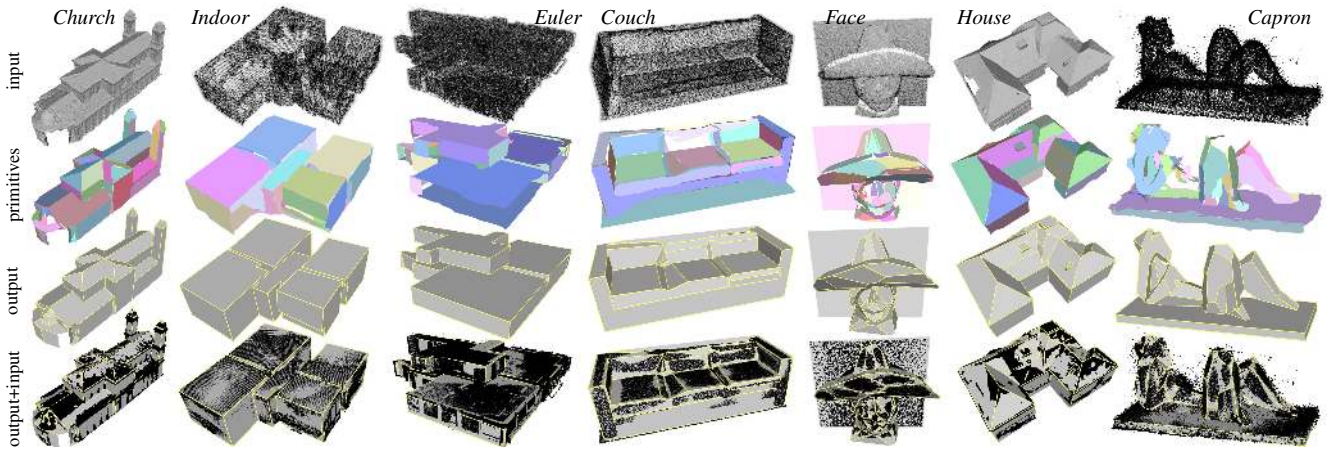


Figure 7. Results on different man-made objects. Our algorithm offers a good versatility by operating on different types of objects and scenes without any specific geometric assumption. Note, in particular, that primitives are neither regularized nor filtered.

primitives only. Figure 6 shows the impact of these two ingredients on output complexity and running time. Used simultaneously, they allow us to strongly reduce running time from 43 minutes to 7 seconds on the shown example. Running time does not depend only on the size of input data and the number of primitives, but also on the amount of structural facets. The latter is high typically on data weakly corrupted by defects and with few free-form components. In such cases, the algorithm is faster. As illustrated in Figure 5, running time and memory peak do not skyrocket when the number of primitives increases. More details on performances are presented in supplementary material.

Comparisons with reconstruction methods. We compared our algorithm with the connectivity-based method Structuring [17] and the slicing-based methods Polyfit [24] and Polyhedral Cell Complex (PCC) [7]. For the latter, no primitive has been artificially added along vertical and horizontal axes in order to fairly compare the assembling mechanisms. As illustrated in Figure 8 and Table 1, we deliver similar results than Polyfit and PCC on simple examples requiring few primitives as Cottage, while being slightly faster. On more challenging datasets where hundred primitives are necessary to decently approximate the objects as *Rubbish bin*, our algorithm performs better in terms of visual quality, output complexity and running time. Structuring is fast and scalable but the mixture of large polygonal facets with fine triangular meshes leads to complex output models which is not a simple assembling of planes. Polyfit and PCC which rely on greedy slicing mechanisms are relatively slow and have memory consumption problems. In particular, the number of primitives for Polyfit on *Rubbish bin* has been reduced to run the algorithm under reasonable time. PCC and Polyfit produce models with visual artifacts when approximating the free-form shapes of *Standford bunny*. On such an object, many planes share almost straight angles, leading to a mislabeling of facets or cells when the set of candidates

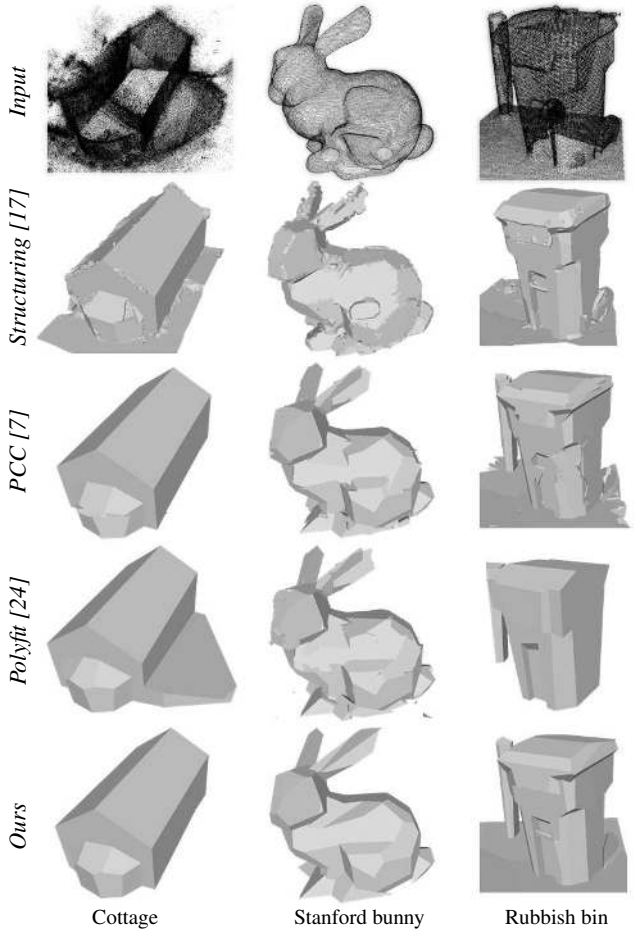


Figure 8. Visual comparison with reconstruction methods. Given similar configurations of primitives, our algorithm produces artifact-free models with a lower complexity in shorter running times than Structuring, PCC and Polyfit (see also Table 1).

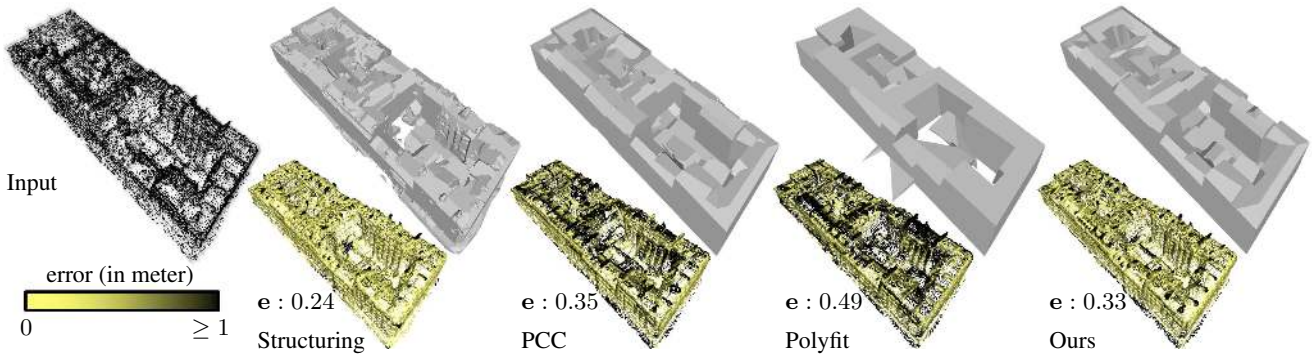


Figure 9. Geometric accuracy on *Building block*. The yellow-to-black colored points represents the Hausdorff distance from the input points to output surface. Structuring obtains the best RMS error $\mathbf{e}$, but the model is not compact as underlined in Table 1. Our error is the second best, outclassing PCC and Polyfit which are penalized by dense space partitions and data consistency terms weakly robust to noise.

	Cottage				*Stanford bunny*				*Rubbish bin*				*Building block*			
	$\#\mathbf{P}$	$\lvert\mathcal{F}\rvert$	$\#\mathbf{f}$	$\mathbf{t}$(s)	$\#\mathbf{P}$	$\lvert\mathcal{F}\rvert$	$\#\mathbf{f}$	$\mathbf{T}$(s)	$\#\mathbf{P}$	$\lvert\mathcal{F}\rvert$	$\#\mathbf{f}$	$\mathbf{T}$(s)	$\#\mathbf{P}$	$\lvert\mathcal{F}\rvert$	$\#\mathbf{f}$	$\mathbf{T}$(s)
Structuring [17]	19	272K	34K	21	100	63K	12K	6	100	115K	20K	11	150	360K	43K	7
PCC [7]	21	3.7K	288	8	100	165K	10.5K	58	100	211K	7.9K	91	150	651K	22K	387
Polyfit [24]	23	1.8K	112	19	100	147K	5.6K	2449	30	3.9K	0.5K	57	54	6.4K	0.9K	1267
Ours	21	0.9K	83	8	100	5.7K	0.6K	22	100	8.7K	1.2K	14	150	12.4K	1.2K	126

Table 1. Quantitative evaluation for models presented on Figures 8 and 9. $\#\mathbf{P}$, $\lvert\mathcal{F}\rvert$ $\#\mathbf{f}$ and $\mathbf{t}$ refer to the number of primitives, the number of candidate facets in $\mathcal{F}$, the number of facets in the output model, and the running time respectively.

is high (the number of candidate facets for PCC and Polyfit is approximatively 30 times higher than with our method) and the data consistency term of the labeling energy does not rely on primitives only. Yet, these methods do not offer a special treatment to scalability, but rather focus on improving the quality of primitives. Figure 9 shows the geometric accuracy of these methods on a complex block of buildings. Our algorithm outclasses Polyfit and PCC while being faster and delivering a more compact output model.

Limitations. Our work focuses on primitive assembling, not on primitive detection or primitive completion. As a result, if primitives are badly detected from input points, we do not offer a special treatment to repair them, contrary to PCC or Polyfit. This typically happens when inputs contain large missing parts. When no detected primitive can decently fill in the missing parts, our algorithm typically shrinks the surface. PCC which artificially adds primitives on these parts along vertical and horizontal directions is then a more suitable choice. Similarly, Polyfit delivers more regularized surfaces for simple objects thanks to its filtering of primitives. Yet, the primitive treatments of these two methods could be also employed with our work in order to rectify our input primitives. Also, the connectivity analysis step typically retreives less structural facets from defect-laden data. The performances of our method are then reduced in this case.

8. Conclusion

We proposed a polygonal surface reconstruction algorithm from 3D data that connects and slices primitives in a robust and scalable manner. The algorithm is built on several key technical ingredients that allows us to operate on an efficient and compact partitioning data-structure. We proposed (i) the principle of soft-connectivity that avoids slicing improbable pairs of primitives, (ii) a preliminary analysis of the connectivity of primitives in order to quickly solve obvious primitive assembling situations, and (iii) a surface extraction energy which estimates the quality of a solution without operating time-consuming measurements to input 3D data. Our algorithm outperforms state-of-the-art methods on challenging input data in terms of performance and output complexity.

Parameter ϵ specifying the soft-connectivity relationship plays an important role in controlling how far the connected primitives can be located from each others. In future work we would like to investigate on its automatic selection. We also wish to understand the hierarchical relationships between primitives in order to detect and exploit high order structural information as symmetry.

Acknowledgments

We thank CSTB for supporting this work, Sven Oesau for technical discussions and the anonymous reviewers for their inputs.

References

[1] M. Arikan, M. Schwarzler, S. Flory, M. Wimmer, and S. Maierhofer. O-snap: Optimization-based snapping for modeling architecture. *Trans. on Graphics*, 32(1), 2013.

[2] M. Berger, A. Tagliasacchi, L. Seversky, P. Alliez, G. Guennebaud, J. Levine, A. Sharf, and C. Silva. A survey of surface reconstruction from point clouds. *Computer Graphics Forum*, 36(1), 2017.

[3] H. Bodis-Szomoru, Riemenschneider and L. Van Gool. Superpixel meshes for fast edge-preserving surface reconstruction. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2015.

[4] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Lévy. *Polygon Mesh Processing*. AK Peters, 2010.

[5] A. Boulch, M. De La Gorce, and R. Marlet. Piecewise-planar 3d reconstruction with edge and corner regularization. *Computer Graphics Forum*, 33(5), 2014.

[6] R. Cabral and Y. Furukawa. Piecewise planar and compact floorplan reconstruction from images. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2014.

[7] A.-L. Chauve, P. Labatut, and J.-P. Pons. Robust piecewise-planar 3D reconstruction and completion from large-scale unstructured point data. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2010.

[8] J. Chen and B. Chen. Architectural modeling from sparsely scanned range data. *International Journal of Computer Vision (IJCV)*, 78(2-3), 2008.

[9] T.-J. Chin, Z. Cai, and F. Neumann. Robust fitting in computer vision: Easy or hard? In *Proc. of European Conference on Computer Vision (ECCV)*, 2018.

[10] J. Coughlan and A. Yuille. The manhattan world assumption: Regularities in scene statistics which enable bayesian inference. In *NIPS*, 2000.

[11] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *Trans. on Information Theory*, 29(4), 1983.

[12] H. Fang, F. Lafarge, and M. Desbrun. Planar Shape Detection at Structural Scales. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2018.

[13] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2016.

[14] J. Huang, T. Jiang, Z. Shi, Y. Tong, H. Bao, and M. Desbrun. l_1-based construction of polycube maps from complex shapes. *Trans. on Graphics*, 33(3), 2014.

[15] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Symposium on Geometry Processing*, 2006.

[16] P. Labatut, J.-P. Pons, and R. Keriven. Hierarchical shape-based surface reconstruction for dense multi-view stereo. In *Proc. of International Conference on Computer Vision (ICCV) workshops*, 2009.

[17] F. Lafarge and P. Alliez. Surface reconstruction through point set structuring. In *Computer Graphics Forum*, volume 32, 2013.

[18] F. Lafarge and C. Mallet. Building large urban environments from unstructured point data. In *Proc. of International Conference on Computer Vision (ICCV)*, 2011.

[19] M. Li, P. Wonka, and L. Nan. Manhattan-world urban reconstruction from point clouds. In *Proc. of European Conference on Computer Vision (ECCV)*, 2016.

[20] Y. Li, X. Wu, Y. Chrysanthou, A. Sharf, D. Cohen-Or, and N. Mitra. Globfit: Consistently fitting primitives by discovering global relations. *Trans. on Graphics*, 30(4), 2011.

[21] D. Marshall, G. Lukacs, and R. Martin. Robust segmentation of primitives from range data in the presence of geometric degeneracy. *Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 23(3), 2001.

[22] A. Monszpart, N. Mellado, G. Brostow, and N. Mitra. Rapter: Rebuilding man-made scenes with regular arrangements of planes. *Trans. on Graphics*, 34(4), 2015.

[23] T. Murali and T. Funkhouser. Consistent solid and boundary representations from arbitrary polygonal data. In *Proc. of Symposium on Interactive 3D Graphics (SI3D)*, 1997.

[24] L. Nan and P. Wonka. Polyfit: Polygonal surface reconstruction from point clouds. In *Proc. of International Conference on Computer Vision (ICCV)*, 2017.

[25] S. Oesau, F. Lafarge, and P. Alliez. Planar shape detection and regularization in tandem. In *Computer Graphics Forum*, volume 35, 2016.

[26] T.-T. Pham, T.-J. Chin, K. Schindler, and D. Suter. Interacting geometric priors for robust multimodel fitting. *Trans. on Image Processing*, 23(10), 2014.

[27] C. Poullis and S. You. Automatic reconstruction of cities from remote sensor data. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2009.

[28] T Rabbani, F van Den Heuvel, and G Vosselman. Segmentation of point clouds using smoothness constraint. *ISPRS*, 36(5), 2006.

[29] F. Schindler, W. Forstner, and J.-M. Frahm. Classification and reconstruction of surfaces from point clouds of man-made objects. In *Proc. of International Conference on Computer Vision (ICCV) Workshops*, 2011.

[30] R. Schnabel, P. Degener, and R. Klein. Completion and reconstruction with primitive shapes. In *Computer Graphics Forum*, volume 28, 2009.

[31] R. Schnabel, R. Wahl, and R. Klein. Efficient ransac for point-cloud shape detection. In *Computer graphics forum*, volume 26, 2007.

[32] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.11 edition, 2017.

[33] M. van Kreveld, T. van Lankveld, and R. Veltkamp. On the shape of a set of points and lines in the plane. *Computer Graphics Forum*, 30, 2011.

[34] Y. Verdie, F. Lafarge, and P. Alliez. LOD Generation for Urban Scenes. *Trans. on Graphics*, 34(3), 2015.

[35] V. Verma, R. Kumar, and S. Hsu. 3D building detection and modeling from aerial LIDAR data. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2006.

[36] L. Zebedin, J. Bauer, K.F. Karner, and H. Bischof. Fusion of feature- and area-based information for urban buildings modeling from aerial imagery. In *Proc. of European Conference on Computer Vision (ECCV)*, 2008.