Kinetic Shape Reconstruction

JEAN-PHILIPPE BAUCHET and FLORENT LAFARGE, Université Côte d'Azur, Inria, France

Converting point clouds into concise polygonal meshes in an automated manner is an enduring problem in computer graphics. Prior works, which typically operate by assembling planar shapes detected from input points, largely overlooked the scalability issue of processing a large number of shapes. As a result, they tend to produce overly simplified meshes with assembling approaches that can hardly digest more than 100 shapes in practice. We propose a shape assembling mechanism that is at least one order of magnitude more efficient, both in time and in number of processed shapes. Our key idea relies upon the design of a kinetic data structure for partitioning the space into convex polyhedra. Instead of slicing all the planar shapes exhaustively as prior methods, we create a partition where shapes grow at constant speed until colliding and forming polyhedra. This simple idea produces a lighter yet meaningful partition with a lower algorithmic complexity than an exhaustive partition. A watertight polygonal mesh is then extracted from the partition with a min-cut formulation. We demonstrate the robustness and efficacy of our algorithm on a variety of objects and scenes in terms of complexity, size, and acquisition characteristics. In particular, we show the method can both faithfully represent piecewise planar structures and approximate freeform objects while offering high resilience to occlusions and missing data.

CCS Concepts: • **Computing methodologies** → *Mesh models*; *Parametric curve and surface models*;

Additional Key Words and Phrases: Surface reconstruction, surface approximation, polygonal surface mesh, kinetic framework

ACM Reference format:
Jean-Philippe Bauchet and Florent Lafarge. 2020. Kinetic Shape Reconstruction. *ACM Trans. Graph.* 39, 5, Article 156 (June 2020), 14 pages.
https://doi.org/10.1145/3376918

1 INTRODUCTION

Polygon meshes are the most widely used representations of surfaces. They capture well the geometry of piecewise-planar objects, e.g., buildings, with ideally one facet for each planar component of an object. Polygonal meshes also approximate freeform shapes effectively [Botsch et al. 2010]. Although many mature algorithms exist to generate *dense* polygon meshes from data measurements such as Laser or MultiView Stereo (MVS) images [Berger et al. 2017], generating concise polygon meshes is still a scientific challenge. While user-guided Computer-Aided Design (CAD) enables the production of such quality meshes, ready for visualization, simulation or fabrication tasks, such interactive tools are labor-intensive and hence not suited to process large amounts of measurement data.

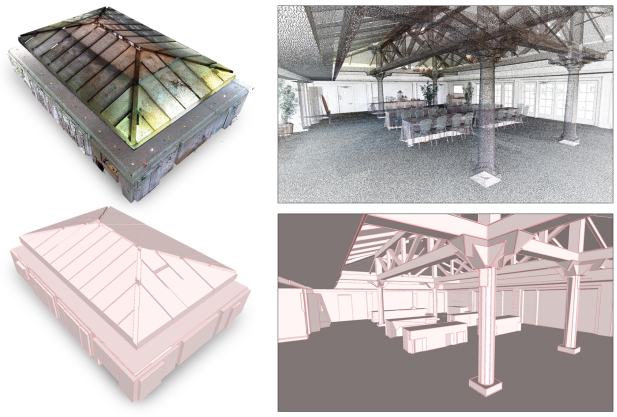


Fig. 1. Goal of our approach. Our algorithm converts a point cloud (top). Here a 3M points laser scan into a concise and watertight polygon mesh (bottom) in an automated manner. Planar components of the indoor scene (beams, walls, windows…) are captured by large facets, whereas freeform objects (columns, tables…) are approximated by a low number of facets. Small objects (chairs, plants…) are filtered out. Model: Meeting Room.

We consider the problem of converting unorganized 3D point clouds into surface polygon meshes with the following objectives:

- Fidelity. The output mesh must approximate well the input point cloud.
- Watertight. The output mesh is a watertight surface mesh, free of self-intersections, bounding a 3D polyhedron.
- Simplicity. The output mesh is composed of a small number of facets, ideally just enough facets given a user-defined tolerance approximation error.
- Efficiency. The algorithm should be fast, scalable, and proceed with a low memory consumption.
- Automation. The algorithm is fully automated and takes only few intuitive user-defined parameters.

Existing methods or pipelines [Boulch et al. 2014; Mura et al. 2016; Nan and Wonka 2017] only partially fulfill the above objectives. They commonly operate by assembling *planar shapes* previously detected from the input points. Planar shapes are sets of inlier points to which a plane is fitted. They are used as intermediate representation between the input points and the output mesh. In most cases, the assembling of the planar shapes consists in slicing the 3D space bounding the input point cloud into a partition of convex polyhedral cells with the (infinite) supporting plane of each shape. The output mesh is then extracted by labeling the cells as inside or outside the input 3D objects. This approach yields concise polygon meshes, but is too compute- and memory-intensive to deal with complex objects and scenes. In our experiments this

This project was partially funded by Luxcarta Technology.
Authors' addresses: J.-P. Bauchet and F. Lafarge, Université Côte d'Azur, Inria, 2004 route des Lucioles, 06902 Sophia Antipolis, France; email: {Jean-philippe.Bauchet, Florent.Lafarge}@inria.fr.

0730-0301/2020/06-ART156 $15.00
https://doi.org/10.1145/3376918

approach can only deal with scenes composed of less than 100 shapes when executed on a standard desktop computer.

To achieve the above objectives, we propose a shape assembling method that is at least one order of magnitude more efficient than existing methods. Our primary contribution hinges on the design and efficient implementation of a *kinetic* data structure for partitioning the 3D space into convex polyhedra. Instead of slicing the 3D space by intersecting all supporting planes with each other, we generate a partition induced by convex planar polygons that grow at constant speed until colliding and creating polyhedra. Such a kinetic approach yields lighter yet meaningful 3D partitions with a significantly lower algorithmic complexity than the exhaustive partitioning. Our secondary contribution is a min-cut formulation for extracting the output surface mesh from the 3D partition. Departing from prior work, our formulation relies upon a fast voting scheme that leverages the orientation of point normals. Finally, we evaluate and compare our approach with state-of-the-art methods against several quality and performance metrics on a collection of 42 datasets that differ in terms of complexity, size, and acquisition characteristics. This evaluation material will be released online. The goal of our method is illustrated in Figure 1.

2 RELATED WORK

Our review of previous work covers the detection of planar shapes from input points, the methods proceeding by assembling planar shapes, as well as alternative methods based on surface approximation or geometry simplification.

Planar shape detection. Planar shape detection consists in clustering the input point data into clusters of inlier points that lie near the same plane. Because we impose a minimal number of inliers for a planar shape, an input point is not necessarily associated to a shape. The common approaches operate by region growing [Marshall et al. 2001; Rabbani et al. 2006] or Ransac [Schnabel et al. 2007]. Region growing propagates a plane hypothesis to neighboring points until fitting conditions are no longer valid. Ransac proposes multiple plane hypotheses from some samples, verifies them against data, and selects the planes with the largest numbers of inliers. More advanced approaches detect shapes and regularize them according to geometric relationships such as parallelism or orthogonality, either sequentially [Li et al. 2011; Oesau et al. 2016] or simultaneously [Monszpart et al. 2015]. Shapes can also be detected by learning approaches trained from CAD model databases [Fang et al. 2018; Li et al. 2019]. We refer to a recent survey for more details [Kaiser et al. 2018]. Our goal is different, as we focus on assembling already detected planar shapes into a watertight polygonal mesh.

Shape assembling. Two main families of methods can be distinguished to assemble planar shapes into a polygon mesh.

Connectivity methods analyze an adjacency graph between planar shapes to extract the vertices, edges, and facets of the output mesh [Chen and Chen 2008; Schindler et al. 2011; Van Kreveld et al. 2011]. Although these methods are fast, they are likely to produce incomplete models on challenging data where adjacency graphs often contain linkage errors. One possible solution consists in completing the missing parts either interactively with user-guided mesh operations [Arikan et al. 2013] or automatically with dense triangle meshes [Holzmann et al. 2018; Lafarge and Alliez 2013]. Unfortunately, the first alternative does not offer a relevant solution for processing large volumes of data, and the second one does not output concise polygon meshes.

Slicing methods [Boulch et al. 2014; Chauve et al. 2010; Mura et al. 2016; Nan and Wonka 2017] are more robust to challenging data. They operate by slicing the input 3D space with the supporting planes of the detected shapes. The output partition is a decomposition into convex polyhedral cells. The output mesh is then extracted by labeling the cells as inside or outside the surface, or equivalently, by selecting the polygonal facets of the cells that are part of the surface. The main limitation is the high algorithmic complexity for constructing the 3D partition, commonly performed via a binary space partition tree updated at each plane insertion [Murali and Funkhouser 1997]. Such a data structure can take hours to construct when more than 100 shapes are involved. Moreover, the returned partitions are overly complex, composed of many small anisotropic cells that are hampering the subsequent surface extraction process. Other approaches [Schnabel et al. 2009; Verdie et al. 2015] discretize the partitioning space instead of computing the exact geometry of the whole partition. This option is less costly, but often yields geometric artifacts when the discretization is not fine enough. In contrast, we do not slice all the shapes exhaustively, and instead grow them until colliding them with each other: this simple and natural idea yields lighter yet meaningful partitions with a lower algorithmic complexity.

Surface approximation. An alternative way to produce concise polygonal meshes is to first reconstruct a smooth surface from the input points and then simplify it. The first step can rely upon mature algorithms from the geometry processing toolbox, such as the popular Poisson reconstruction method [Kazhdan et al. 2006; Kazhdan and Hoppe 2013]. We refer to a recent survey that discusses the main algorithms for smooth surface reconstruction [Berger et al. 2017]. Once extracted, the dense triangle mesh is simplified into a coarser mesh. The most common approach consists in contracting edges until reaching a target number of facets [Garland and Heckbert 1997; Lindstrom 2000]. To better preserve the piecewise-planar structure of objects, the edge contraction operators can account for the planar shapes detected in the dense mesh [Salinas et al. 2015]. However, on planar parts such approaches yield triangle meshes whose adjacent facets are unlikely to be coplanar and form a meaningful decomposition. Other methods [Chen et al. 2013; Cohen-Steiner et al. 2004] alleviate this problem by connecting planar shapes using the adjacency inferred from the input mesh. These surface approximation methods are in general effective, but they require as input a mesh that is both geometrically and topologically accurate to deliver faithful results. Unfortunately, such a requirement is rarely guaranteed from real-world data, which are often highly corrupted by noise, outliers, and occlusions.

Geometry simplification. Some works further reduce the combinatorial problem by imposing geometric assumptions for the output surface. For instance, the Manhattan-World assumption [Coughlan and Yuille 2000] enforces the generation of polycubes [Huang et al. 2014; Ikehata et al. 2015; Tarini et al. 2004] by imposing output facets to follow only three orthogonal directions.

Another popular geometric assumption consists in constraining the output surface to specific disk-topologies. For instance, 2.5D view-dependent representations are typically well suited to buildings with airborne data [Musialski et al. 2013] and facades with streetside images [Bodis-Szomoru et al. 2015]. Some approaches also approximate surfaces with specific layouts of polygonal facets. Such polyhedral patterns are particularly relevant for architectural design [Jiang et al. 2015, 2014]. Unfortunately, such assumptions are only relevant for specific application domains. Besides piecewise-planar, our algorithm does not rely upon any geometric assumptions and remains generic.

3 OVERVIEW

Figure 2 illustrates the main steps of our method. Similarly to the popular Poisson algorithm [Kazhdan et al. 2006] and most of smooth surface reconstruction methods, our algorithm requires as input a point cloud with oriented normals. Point clouds are typically generated from MVS, laser scanning, or depth camera. Note that the oriented normals are commonly returned by these acquisition systems: normals are estimated by principal component analysis or more advanced techniques, whereas orientations are given by the point-to-camera center direction. Our algorithm also requires as input a configuration of planar shapes. The latter is extracted from the point cloud by region growing [Rabbani et al. 2006] or Ransac [Schnabel et al. 2007] in our experiments. These algorithms are standard geometric processing tools implemented in the CGAL library [Oesau et al. 2018]. More advanced methods exploiting filtering and regularization of planar shapes could be also considered. The shape extraction algorithms are typically controlled by two parameters: a fitting tolerance ϵ that specifies the maximal distance of an inlier to its planar shape, and a minimal shape size σ that allows the rejection of planar shapes with a low number of inliers.

As a preliminary step, we compute the convex hull of each planar shape, i.e., the smallest convex polygon containing the projection of the inlier points on the optimal plane of the shape.

The first step of our method consists in partitioning the bounding box of the object into polyhedra, each facet of a polyhedron being a part or an extension of the convex polygons. This step relies on a kinetic framework in which convex polygons extend at constant speed until colliding with each other. When a collision between two or more polygons occurs, we modify the evolution of these polygons by either stopping their growth, changing their direction of propagation, or splitting them. This set of polygons progressively partitions the 3D space into polyhedra, the final partition being obtained when no polygon can grow anymore.

The second step extracts the polygonal surface mesh from the partition by labeling polyhedra as inside or outside the reconstructed object. We formulate it as a minimization problem solved by min-cut. Our objective function, called an energy, takes into account both the fidelity to input points and the simplicity of output models.

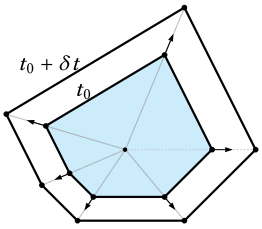
Our algorithm outputs a polygonal surface mesh whose facets lie on the supporting plane of the planar shapes. The output mesh is guaranteed to be watertight and intersection-free. Moreover, its volume is already decomposed into convex polyhedra, which is an interesting property for simulation tasks.

4 ALGORITHM

4.1 Kinetic Partitioning

Background on Kinetic frameworks. Kinetic data structures have been introduced in computational geometry to track physical objects in continuous motion [Guibas 2004; Guibas et al. 1983]. A kinetic data structure is composed of a set of geometric *primitives* whose coordinates are continuous functions of time. The purpose of kinetic frameworks is to maintain the validity of a set of geometric properties on the primitives, also called *certificates*. When a certificate becomes invalid, geometric modifications are operated on the primitives to make the geometric properties valid again. This situation, also called an *event*, happens, for instance, when several primitives collide. Kinetic frameworks show a strong algorithmic interest to dynamically order the times of occurrence of the events within a *priority queue*. A kinetic simulation then consists in unstacking this queue until no primitive moves anymore. Kinetic data structures have been mostly designed for moving 2D primitives in the plane. Typical examples include dynamic 2D Delaunay triangulations with moving vertices [Agarwal et al. 2010], dynamic planar graphs with the propagation of line segments [Bauchet and Lafarge 2018], or a pair of polygons in a motion [Basch et al. 2004]. To our knowledge, we are the first to design and implement a kinetic data structure that collides polygons in the 3D space.

Primitives and kinetic data structure. The primitives of our kinetic framework are polygons whose vertices move at constant speeds along given directions. The initial set of polygons is defined as the convex hulls of the planar shapes. Polygons grow by uniform scaling: Each vertex moves in the opposite direction to the center of mass of the initial polygon (see inset). The underlying kinetic data structure $\mathcal{P}$ is the set of these polygons that, we assume, do not intersect with each other. When two or more polygons intersect, we modify primitives to maintain valid the intersection-free property. For generality reasons, polygons extend at constant speed: No planar shape is favored over others.



Certificates. For each primitive i, we define the certificate function $C_i(t)$ as

$$C_i(t) = \prod_{\substack{j=1 \\ j \neq i}}^{N} Pr_{i,j}(t), \tag{1}$$

where N is the number of primitives of the kinetic data structure, and $Pr_{i,j}(t)$ the predicate function that returns 0 when primitive i collides with primitive j, i.e., when the minimal distance between the polygon boundary of primitive i and the closed polygon of primitive j equates zero for the first time, and 1 otherwise. Primitive i is called the *source primitive*, and j, the *target primitive*. Because the directions of propagation of polygons can change along time, the computation of this minimal distance is a difficult task in practice. We detail in Section 5 how we compute this distance efficiently.

Initialization. Before starting growing the set of convex hulls, we first need to decompose them into intersection-free polygons. As

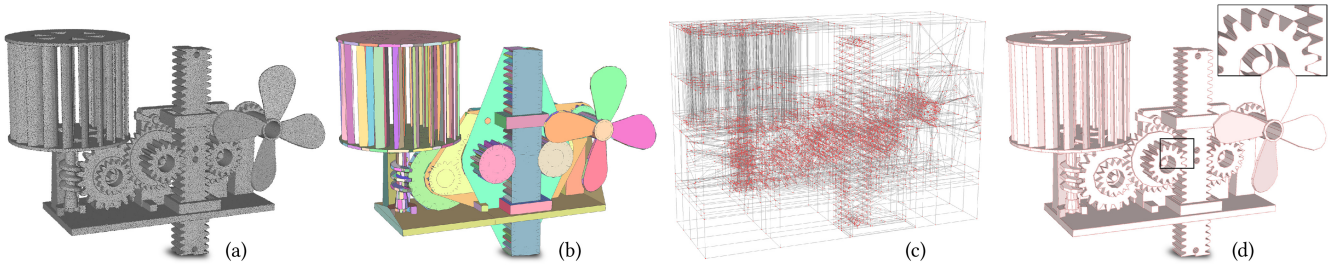


Fig. 2. Overview. Our algorithm takes as input a point cloud with oriented normals (a) and a configuration of convex polygons that approximate the object (b). Convex polygons are obtained through shape detection as the convex hulls of inlier points projected onto the shape. The 3D bounding box of the object is decomposed into polyhedra by kinetic partitioning from the convex polygons (c). The vertices and edges of polyhedra are displayed by red dots and grey line-segments, respectively. A concise polygonal mesh is then extracted by labeling each polyhedron as inside or outside the object (d). Model: FULL THING.

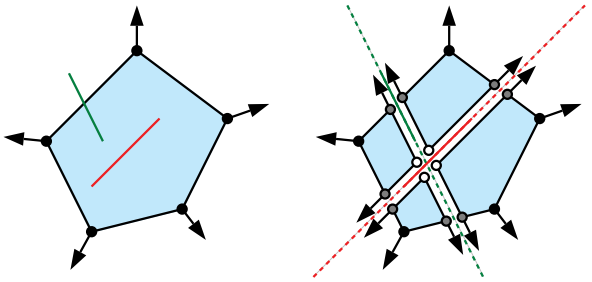


Fig. 3. Initialization. At $t = 0$, the blue polygon intersects two other polygons, the intersections being represented by the red and green line-segments (left). We decompose it into intersection-free polygons with cuts along the intersection lines (red and green dashed lines). The direction of propagation of original vertices (black points) are unchanged, while the new vertices are either kept fixed when located at the junction of several intersection lines (white points) or are moving along the intersection lines (grey points). We denote them as *frozen vertices* and *sliding vertices*, respectively (right). A polygon is *active* if not all its vertices are frozen.

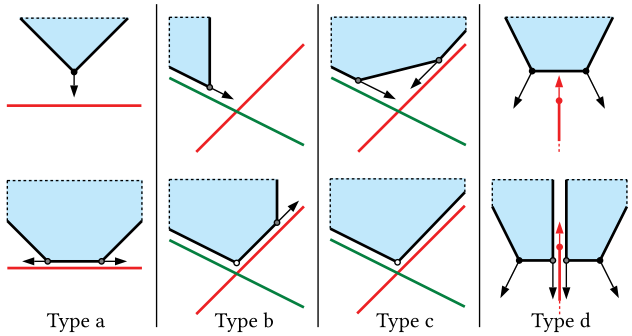


Fig. 4. Collision typology and corresponding primitive updates. A vertex or an edge of the source polygon (blue shape) typically collides with the target polygon (red segment) in four different manners (top). The corresponding updates consist in inserting sliding vertices and/or frozen vertices in the source polygon for types a, b, and c, or splitting polygons for type d (bottom).

illustrated in Figure 3, each non-intersection-free polygon is cut along the intersection lines with the other polygons. Because the propagation of polygons outside the bounding box is not relevant in practice, we also insert the six facets of the bounding box into the kinetic data structure. We denote by $\mathcal{P}_0$ this set of intersection-free polygons. In addition, we populate the priority queue by computing and sorting in ascending order the times of collision, i.e., times for which certificates $C_i(t) = 0$ for each polygon i of $\mathcal{P}_0$.

Updating operations. When a collision between primitives occurs, we need to update the kinetic data structure to keep the set of polygons free of intersection.

We first modify the source polygon. As illustrated in Figure 4, four collision types are distinguished:

- A vertex of the source polygon collides with the target polygon (type a). We replace the vertex by two sliding vertices that move along the intersection line in opposite directions.
- A sliding vertex of the source polygon collides with the target polygon (type b). We modify the direction of propagation of the sliding vertex to follow the intersection line with the target polygon. We also create a frozen vertex at the junction between the intersection line with the target polygon and the intersection line with the polygon supporting the sliding vertex before collision.
- A sliding vertex of the source polygon collides with the target polygon while a sliding vertex guided by the target polygon already exists (type c). We create a frozen vertex where the two intersection lines meet: The propagation of the source polygon is locally stopped.
- An edge of the source polygon collides with an edge or vertex of the target polygon (type d). Source and target polygons are split along their intersection line with the creation of eight sliding vertices (two per new polygon).

These four collision types include all the possible cases of collisions between two polygons. In particular, collisions between two coplanar polygons are processed in types a and b. Also, the collision between an edge of the source polygon and the interior of the target polygon is simply treated as type a.

In many situations, it is interesting to extend the source polygon on the other side of the target polygon. To decide whether this should happen, we define a *crossing condition* that is valid if *(i)* the source polygon has collided a number of times lower than a user-specified parameter K, or else if *(ii)* relevant input points omitted during shape detection can be found in the other side of the target polygon. Parameter K is a trade-off between the complexity of the

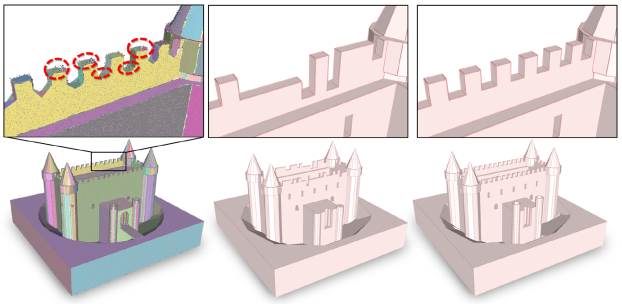
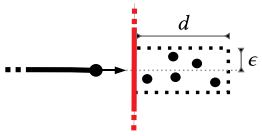
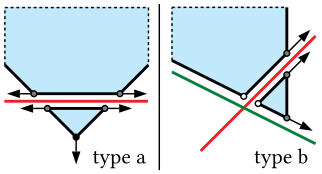


Fig. 5. Data-driven criterion. Small planar parts are often missed during shape extraction (see missing shapes on the left closeup). The output surface then becomes overly simplified (middle). The data-driven criterion allows us to recover the correct geometry when these planar parts belong to a repetitive structure such as the ramparts (right). Model: Castle.

polyhedron partition and the robustness to missing data. In particular, $K = 1$ yields the simplest partition, whereas $K = \infty$ produces the partition returned by the exhaustive plane slicing approach. Parameter K is typically set to 2 in our experiments. The second criterion is data-driven: It counts the number of input points contained in a box (black dashed rectangle in the inset) located behind the target polygon (red segment) and aligned with the source polygon (black segment). In the inset, this number is 5. The thickness of the box is twice the fitting tolerance ϵ used for detecting planar shapes while its depth d is chosen experimentally as 10 times the average distance $\widehat{d}$ between neighbors of the k-nearest neighbor graph of the input points. Assuming the average density of points sampling the object surface is approximately $\widehat{d}^{-2}$, we validate the criterion when the density of points in the box is higher than half of $\widehat{d}^{-2}$ in practice. This data-driven criterion is useful to recover planar parts missed during shape detection, as shown in Figure 5.



If the crossing condition is valid, a new polygon that extends the source primitive on the other side of the target polygon must be inserted into the kinetic data structure. This situation can only happen to collision types a and b in Figure 4. The new polygon is initialized as a triangle composed of two sliding vertices and one original vertex for type a, or of two sliding vertices and one frozen vertex for type b (see inset).



Finally, we update the priority queue. We first remove the processed event between the source and the target polygons from the queue. If the two polygons have been modified, we recompute their times of collision with the active polygons if not all their vertices are frozen, or remove their times of collision with the active polygons from the queue otherwise. If new polygons have been created (type d or types a/b with the crossing condition valid), we compute their times of collision with the active polygons and insert them into the priority queue.

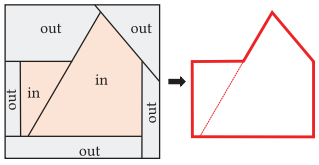
Finalization. When the priority queue is empty, the kinetic data structure does not evolve anymore. The output polygons are composed of frozen vertices only and form a partition of polyhedra. We extract this partition as a 3D combinatorial map [Damiand 2018]. The 2-cells of the combinatorial map correspond to the union of the internal facets, which are the final set of intersection-free polygons $\mathcal{P}$ and the external facets located on the sides of the 3D bounding box. Polyhedra, or 3-cells of the combinatorial map, are then extracted as the smallest closed sets of adjacent facets. Note that the polyhedra are convex by construction. The global mechanism of the kinetic partitioning is summarized in Algorithm 1. A detailed pseudo-code is also provided in supplementary material.

ALGORITHM 1: Kinetic partitioning

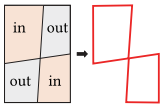
```
Initialize the kinetic data structure P to P_0
Initialize the priority queue Q
while Q ≠ ∅ do
    Pop the source and target primitives from Q
    Test the crossing condition of the source primitive
    Update P
    Update Q
end while
Finalize the tessellation
```

4.2 Surface Extraction

We now wish to extract a surface from the partition of polyhedra. We operate a min-cut to find an inside-outside labeling of the polyhedra, the output surface being defined as the interface facets between inside and outside (see the red polygon in the inset). Because the kinetic partition is a valid polyhedral embedding, this strategy guarantees the output surface to be watertight (i.e., composed of close surfaces) and intersection-free, similarly to min-cut formulations operating on 3D triangulation [Labatut et al. 2009].



Moreover, it provides a decomposition of the output model into convex polyhedra. The output surface mesh is, however, not guaranteed to be 2-manifold. As illustrated in the inset, two polyhedra labeled as inside can be connected along one edge or at one vertex while being facet-adjacent to polyhedra all labeled as outside.



The methods relying upon this approach [Boulch et al. 2014; Chauve et al. 2010; Verdie et al. 2015] typically assign an inside-outside guess on each polyhedron by ray shooting: This is both imprecise in presence of missing data and computationally costly. We instead propose a faster voting scheme that exploits the oriented normals of inlier points to more robustly assign a guess on a portion of the polyhedra only.

We denote by C the set of polyhedra of the tessellation, and by $x_i = \{in, out\}$, the binary label that specifies whether polyhedron i is inside ($x_i = in$) or outside ($x_i = out$) the surface. We measure the quality of a possible output surface $\mathbf{x} = (x_i)_{i \in C}$ with a two-term energy of the form

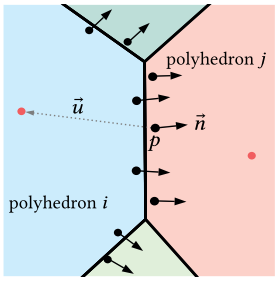
$$U(\mathbf{x}) = D(\mathbf{x}) + \lambda V(\mathbf{x}), \tag{2}$$

where $D(\mathbf{x})$ and $V(\mathbf{x})$ are terms living in $[0,1]$ that measure data fidelity and surface complexity, respectively. $\lambda \in [0,1]$ is a parameter balancing the two terms. The optimal output surface that minimizes U is found by a max-flow algorithm [Boykov and Kolmogorov 2004].

Data fidelity $D(\mathbf{x})$ measures the coherence between the inside-outside label of each polyhedron and the orientation of normals of inlier points. Similarly to signed distances proposed in smooth surface reconstruction, we assume here that the normals point towards the outside. After associating each inlier point with the facet of the partition that contains its orthogonal projection on the infinite plane of its planar shape, we express data fidelity by a voting on each inlier point under the form

$$D(\mathbf{x}) = \frac{1}{|\mathcal{I}|} \sum_{i \in C} \sum_{p \in \mathcal{I}_i} d_i(p, x_i), \tag{3}$$

where $|\mathcal{I}|$ is twice the total number of inlier points, $\mathcal{I}_i$ is the set of inlier points associated with all the facets of polyhedron i, and $d_i(p, x_i)$ is a voting function that tests whether the orientation of inlier point p is coherent with the label x_i of polyhedron i. This function is defined by $d_i(p, in) = 1_{\{\vec{n}\cdot\vec{u}>0\}}$ and $d_i(p, out) = 1_{\{\vec{n}\cdot\vec{u}<0\}}$ where $1_{\{.\}}$ is the characteristic function, $\vec{n}$ the normal vector of inlier point p, and $\vec{u}$ the vector from p to the center of mass of polyhedron i. In the inset example, we prefer assigning label *in* to polyhedron i and label *out* to polyhedron j. In particular, $d_i(p, x_i = out)$ and $d_j(p, x_j = in)$ return a penalty of 1 whereas $d_i(p, x_i = in) = d_j(p, x_j = out) = 0$. Because the voting function d_i is binary, normals only need to point towards the right half-space separating the facet. This brings robustness to imprecise normal directions.



The second term $V(\mathbf{x})$ is more conventional: It measures the complexity of the output surface by its area, where lower is simpler. It is expressed by

$$V(\mathbf{x}) = \frac{1}{A} \sum_{i \sim j} a_{ij} \cdot 1_{\{x_i \neq x_j\}}, \tag{4}$$

where $i \sim j$ denotes the pairs of adjacent polyhedra, a_{ij} represents the area of the common facet between polyhedra i and j, and A is a normalization factor defined as the sum of the areas of all facets of the partition. As shown in Figure 6, this term avoids the surface zigzagging. Giving a too-high importance to this term, however, shrinks the surface. In our experiments, we typically set λ to 0.5.

5 IMPLEMENTATION DETAILS

Our algorithm has been implemented in C++ using the CGAL library. In particular, we used the exact predicates, exact constructions kernel for the computation of distances and times of collision. We now explain several important implementation details that improve the efficiency, stability, and scalability of our method.

Reformulation as collaborative 2D collision problems. At each update of the kinetic data structure, the propagation of the source polygon is likely to be modified. Consequently, the update of the priority queue requires the distance of this polygon to each other active polygon to be recomputed. To avoid such time-consuming operations, we algorithmically reformulate the 3D collision problem as N collaborative propagations of polygons in 2D controlled by a global priority queue, N being the number of planar shapes. The intuition behind this reformulation is that two non-coplanar polygons cannot collide somewhere else than along the 3D line intersecting their respective planes. This 3D line being represented by a 2D line in the planes containing the two polygons (see dashed lines in the inset), we can then use a simple point-to-line distance in 2D to compute the times of collisions. Although more events must be processed with this reformulation, the update of the priority queue is drastically simplified.

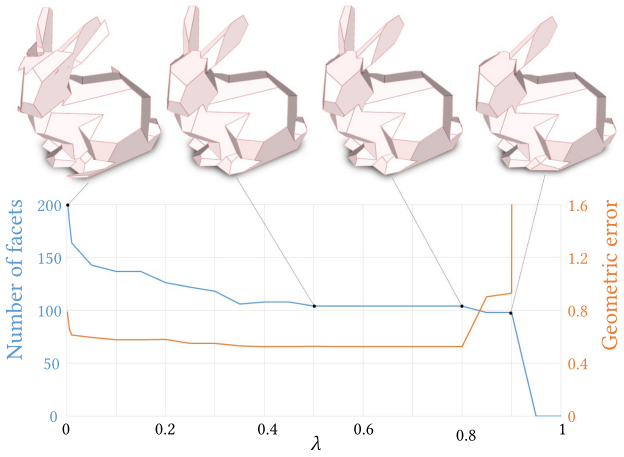
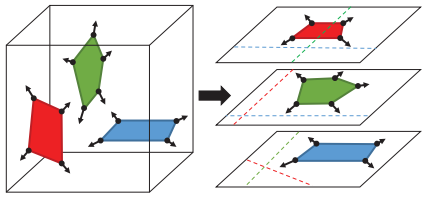


Fig. 6. Impact of parameter λ. Increasing λ simplifies the output surface by reducing the number of facets (blue curve). A too-high value, however, makes object parts disappear and increases the geometric error (red curve). The best compromise between simplicity and accuracy is returned between 0.4 and 0.8. The geometric error is computed as the symmetric mean Hausdorff distance between input points and output mesh. Model: Stanford bunny.



Computation with rationals. Our implementation relies upon the exact predicates, exact constructions kernel of the CGAL library. All distances and times of collisions are computed with rational numbers after converting the input planar primitives with decimal coordinates into rationals. The final kinetic partition is then guaranteed to be a valid polyhedron embedding, and the output mesh to not contain degenerated facets. Note that, because the vertices of the kinetic partition are necessarily intersections of 3 initial supporting planes, the growth of the rational representation remains low. Once extracted by the min-cut formulation, the output mesh is exported with a double precision whose approximation error with the rational numbers is upper-bounded by a user-defined value (set to 10^{-12} in our experiments using the Lazy_exact_nt CGAL number type). In addition, the orientation of the input planar shapes is expressed to the nearest tenth of a degree so the orientation of two

non-coplanar shapes necessarily differs from at least 0.1 degree. These two ingredients are an efficient way to avoid potential imprecision during the floating point conversion in practice: Among all our runs, we never experimented imprecision cases leading to self-intersecting facets in the kinetic partition.

Priority queue. Although all possible collisions should be inserted into the priority queue, we observe that an active vertex often collides with only the three closest lines. To reduce memory consumption and processing time, we thus restrict the number of collisions per vertex in the priority queue to three. If the three collisions occur and the vertex is not frozen, we insert the three next collisions of this vertex in the priority queue and repeat this operation as many times as necessary. This reduces running time and memory consumption by a factor close to 1.5 and 3, respectively.

Simultaneous collisions. Our implementation sequentially processes the collisions between pairs of primitives and does not directly handle situations where three primitives or more mutually collide at the same time. We treat these particular situations by simply applying an infinitesimal perturbation on the initial coordinates of the polygon vertices while maintaining the vertices in the supporting planes. This simple trick avoids a laborious implementation of the numerous cases of simultaneous collisions between k primitives while preserving the efficiency of the algorithm. This technical choice is also based on the observation that such simultaneous multiple collisions is improbable in practice when planar shapes are detected with a floating precision.

Spatial subdivision. To increase the scalability of our algorithm, we offer the option to subdivide the bounding box of the object into uniformly-sized 3D blocks in which independent kinetic partitionings are operated. For each block, we collect the closest planar shapes, i.e., those whose initial convex polygon is either inside the block or intersecting at least one of its sides. We then operate the kinetic partitioning inside the block from this subset of planar shapes. Once all blocks are processed, we merge the polyhedral partitions. An important speed-up factor lies into the fact that only a portion of planar shapes is involved in the partitioning of each block. Figure 7 plots the time and memory savings against the number of planar shapes for several subdivision schemes. While the savings are significant, the use of spatial subdivision produces a simplified polyhedral partition as polygons do not propagate to neighboring blocks. This may affect the quality of the output surface with typically the presence of extra facets at the borders of blocks. Figure 8 illustrates this compromise between algorithm efficiency and surface quality. Our implementation treats blocks sequentially, but one could process them in parallel for even better performances.

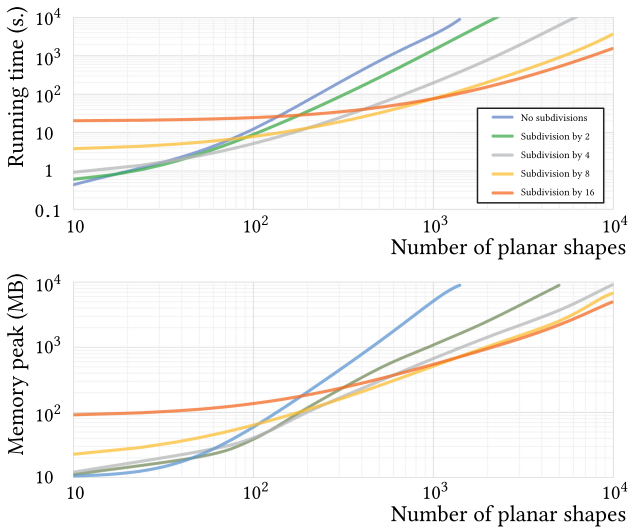


Fig. 7. Performances with spatial subdivision. The lowest curve in each graph indicates the most efficient subdivision scheme given the number of planar shapes. For instance, 1K planar shapes can be processed in one minute by subdividing the 3D bounding box into 8^3 blocks (yellow curve in top graph), whereas half an hour is required without subdivision (blue curve). Memory peak evolves similarly to running time. The performances have been obtained from 100K input points uniformly sampled on a sphere.

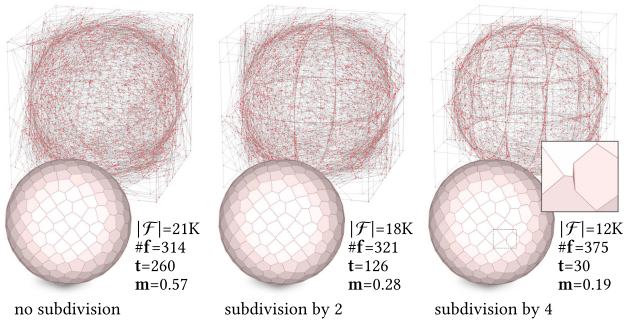


Fig. 8. Spatial subdivision: efficiency vs quality. Our algorithm produces an ideal mesh with 314 polygonal facets from 314 planar shapes approximating a sphere (left). Subdividing the bounding box into blocks reduces running time and memory consumption and produces simplified polyhedra partitions (top). This option may degrade the quality of the output meshes (bottom) with typically the presence of extra facets at the block borders (see closeup). $|\mathcal{F}|$, #**f**, **t**, and **m** correspond to the number of facets in the polyhedral partition, the number of polygonal facets in the output mesh, running time (in sec), and memory peak (in GB), respectively.

6 RESULTS

We evaluated our algorithm with respect to our fidelity, simplicity, and efficiency objectives. We measure the fidelity to 3D data by the symmetric mean Hausdorff (SMH) distance between input point cloud and output mesh. The simplicity of output representation is quantified by the ratio between the number of output facets to the number of initial planar shapes. We measure the efficiency of the algorithm by both running time and memory peak. In addition to these four measures, we also evaluate the scalability of algorithms by the maximal number of planar shapes that can be processed without exceeding both 10^5 seconds on a single computer with a core i9 processor clocked at 2.9 GHz, and 32 GB memory consumption. Input planar shapes were extracted by the region-growing implementation of the CGAL library [Oesau et al. 2018]. The fitting tolerance ϵ were typically set to 0.2% of the bounding box diagonal on large scenes and 1% on more simple objects and multi-view stereo acquisitions. The minimal shape size σ ranged from 0.001% to 1% of the total number of input points, depending

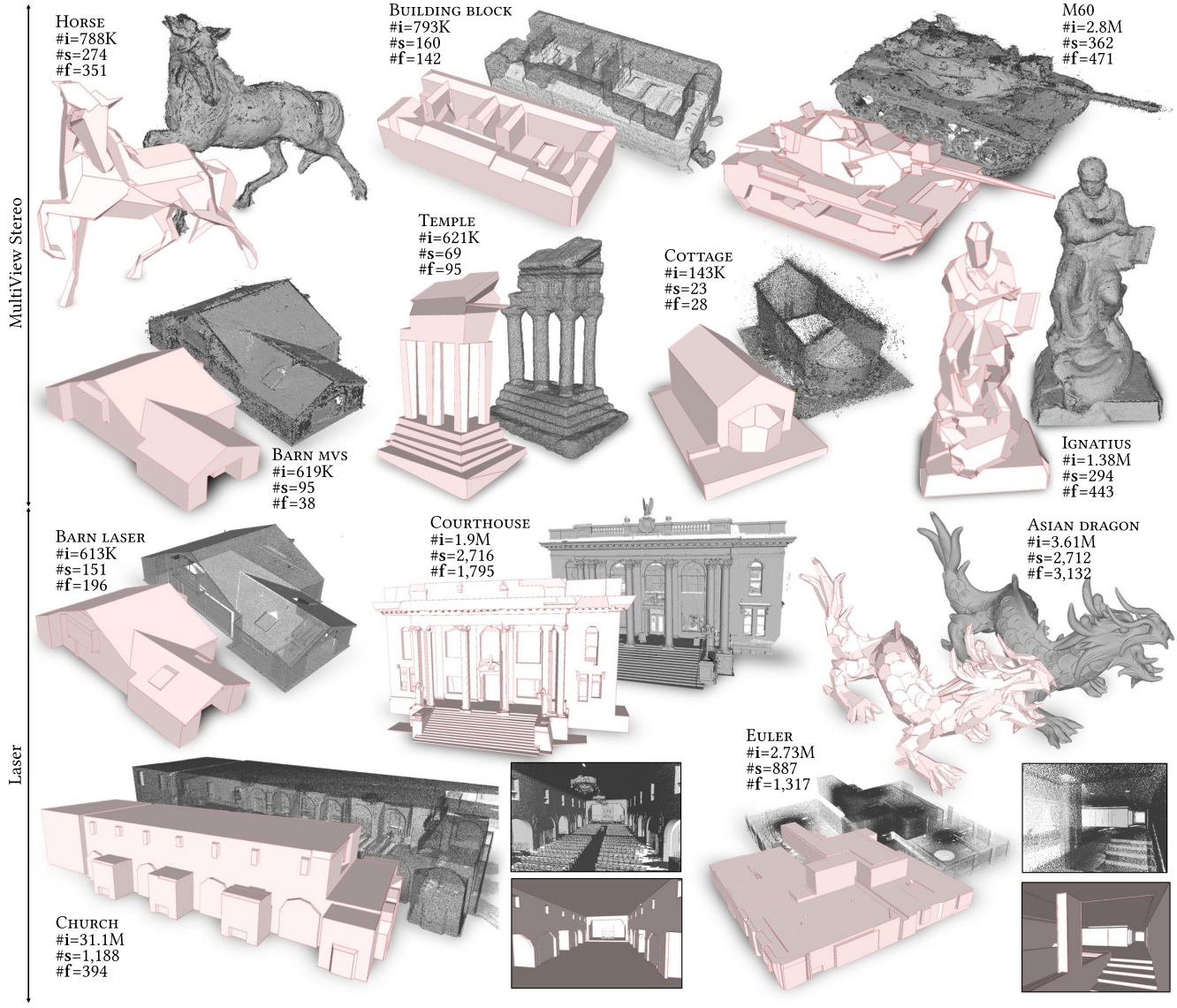


Fig. 9. Reconstructions from various multi-view stereo and laser datasets. Our algorithm produces concise polygonal meshes for both freeform objects such as Horse, Ignatius, and Asian dragon and piecewise-planar structures such as Church, Barn, and Euler. Complex objects such as M60 are approximated with a good amount of details by a few hundred facet mesh only. Because of a more accurate detection of planar shapes, our algorithm typically performs better from laser datasets with more detailed reconstructed models (see Barn mvs and Barn laser models). #i, #s, and #f correspond to the number of input points, planar shapes, and output facets, respectively.

on the complexity of scenes. The exact values are given in supplementary material for the 42 tested models. Globally speaking, this pair of values must be chosen according to the desired abstraction level where the higher the values of ϵ and σ, the more concise the output model. Note that the optimal choice for these two values can also be learned from training sets in which each point cloud is associated with a targeted configuration of planar shapes [Fang et al. 2018].

Flexibility. Our algorithm has been tested on a variety of objects and scenes. Freeform objects such as Stanford Bunny, Hand, Horse, or Ignatius are approximated by compact polygonal meshes. For instance, each digit of Tower of Pi is represented by only a few polygonal facets in Figure 10. Piecewise-planar structures such as Barn, Euler, Meeting room, or Full thing are reconstructed with fine details as long as planar shapes are correctly detected from data. We also tested our algorithm on different acquisition systems. MultiView Stereo (MVS) datasets presented in Figure 9 have been generated by COLMAP [Schönberger and Frahm 2016] from image sequences mostly provided by the Tanks and Temples benchmark [Knapitsch et al. 2017]. Because of the high amount of noise, these point clouds are particularly challenging to reconstruct. Point clouds generated by

Fig. 10. Reconstruction of Tower of Pi. 10.8K planar shapes are detected from 2.9M input points (left) and assembled by our algorithm into a concise polygonal mesh of 12.1K facets (middle). Each digit is nicely represented by a few facets (see closeups). In contrast, the mesh produced by the traditional approximation pipeline (Screened Poisson [Kazhdan and Hoppe 2013] then QEM edge collapse [Garland and Heckbert 1997]) at the same output complexity fails to preserve the structure of the digits (right).

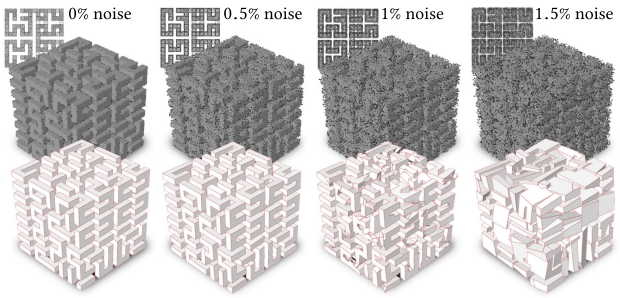


Fig. 11. Robustness to noise. Our algorithm is robust to noise as long as planar shapes can be decently extracted from input points. At 1% noise, some shapes become missing or inaccurately detected, leading to an overly complex output mesh. At 1.5%, the poorly extracted shapes do not allow us to capture the structure of the cube anymore. Note that normals have been recomputed for each degraded point cloud. Model: Hilbert Cube.

laser scanning are geometrically more accurate, but suffer from missing data and heterogeneous point density. As illustrated with Barn model, our algorithm typically returns more accurate output meshes with laser acquisition. MVS point clouds are often too noisy to capture fine details with small planar shapes. Conversely, frequent occlusions contained in laser scans are effectively handled by our kinetic approach that naturally fills in empty space in between planar shapes. Some point clouds have also been sampled from CAD models. This is the case of Full thing, Castle, Tower of Pi, and Hilbert cube, whose CAD models originate from the Thingi10k database [Zhou and Jacobson 2016].

Robustness to imperfect data. Figure 11 shows the robustness of our algorithm to noise. Below 1% noise (with respect to the bounding box diagonal), our algorithm outputs accurate meshes. Above 1%, planar shapes become missing or inaccurately detected. Because our goal is not to correct or complete the shape configuration, the output mesh cannot preserve the structure of the object anymore. Besides noise, the propagation of planar shapes within

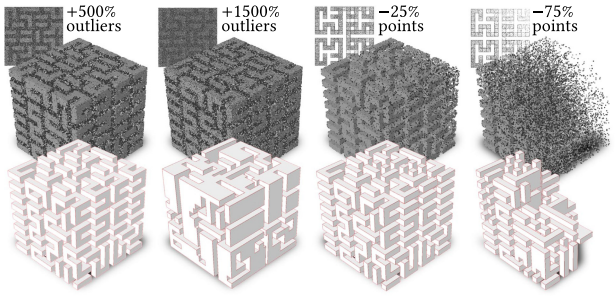


Fig. 12. Robustness to outliers and heterogeneous sampling. Our algorithm returns an accurate model when adding 500% of outliers in the object bounding box, but starts compacting the structure around 1500% of added outliers. In the two right examples, the point clouds have been progressively sub-sampled from the bottom left to the top right of the cube in a linear manner. Our algorithm is resilient to such heterogeneous sampling as long as shapes can be retrieved in the low density of points.

Table 1. Performances of Our Algorithm on Various Models

	Meeting room	Full thing	Tower of Pi	Temple	Church
# input points	3.07M	1.38M	2.9M	621K	31.1M
# planar shapes	1,655	1,648	10.8K	69	1,188
#output facets	1,491	1,807	12,059	95	394
Subdivision	4	4	8	no	2
kinetic partitioning (sec)	197	443	1,696	18	310
surface extraction (sec)	84	49	105	17	717
memory peak (MB)	642	552	3,864	69	325

the kinetic algorithm offers high resilience to occlusions, especially when parameter K is strictly greater than 1. This allows, for instance, to recover the skylights on Barn laser and the rampart structures on Castle. As illustrated in Figure 12, our algorithm is also robust to heterogeneous sampling and outliers by inheriting the good behavior of shape detection methods with respect to these two defects.

Performances. Table 1 presents the performances of our algorithm in terms of running time and memory consumption from various models. Timings and memory consumption of all the results presented in the article are provided in supplementary material. Kinetic partitioning is typically the most time-consuming step. In particular, around 90% of computing efforts focus on the processing of the priority queue. The total number of collisions depends on multiple factors, which include the number of initial convex polygons, their number of vertices, their mutual positioning within the bounding box, as well as parameter K. For instance, 53K collisions are processed for the 75 shapes of the Hand model shown in Figure 16. The most frequent collision types are b and c with an occurrence of 51% and 30%, respectively. Collision type a is the most time-consuming update because of the costly insertion of several sliding vertices in the kinetic data structure.

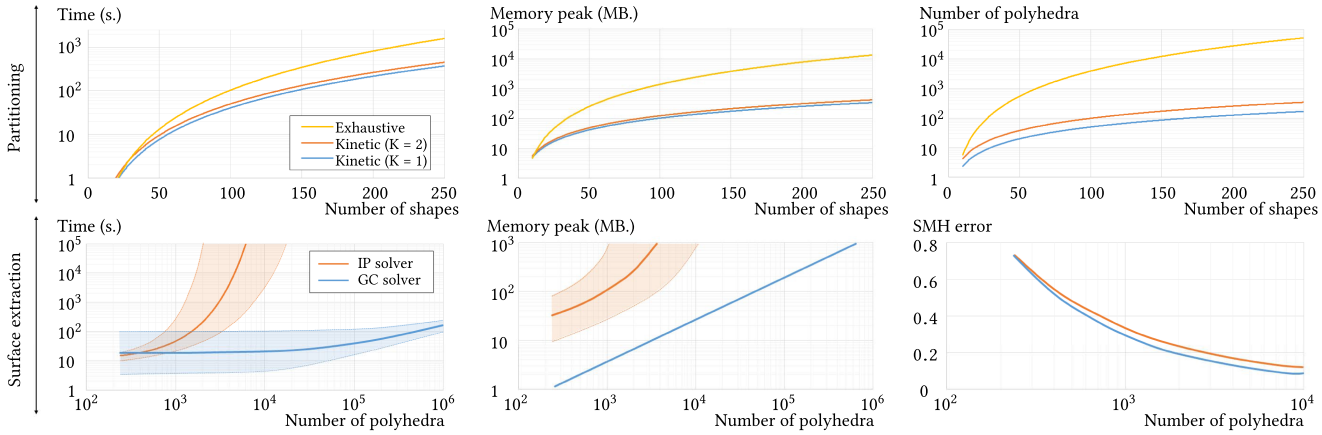


Fig. 13. Performances of partitioning and surface extraction. Kinetic partitioning produces more compact sets of polyhedra in a more time- and memory-efficient manner than exhaustive partitioning (top row). The gain becomes significant from 150 planar shapes with one (respectively, two) order of magnitude saving in memory consumption (respectively, number of polyhedra). When partitions contain less than 1K polyhedra, our graph-cut (GC) formulation for surface extraction and the integer programming (IP) solver of Nan and Wonka [2017] have similar running times and geometric errors (bottom row). However, the IP solver consumes more memory and can hardly digest partitions with more than 10K polyhedra. Moreover, the IP solver is less stable as running times and memory peaks can unpredictably vary (see light orange bands in bottom left/middle graphs). In contrast, the variation in time observed for our GC formulation only depends on the number of input points, the lower (respectively, upper) bound of the light blue band corresponding to the model with the lowest (respectively, highest) number of points. Statistics were drawn from a collection of seven different models with input point sizes ranging from 100K to 5M.

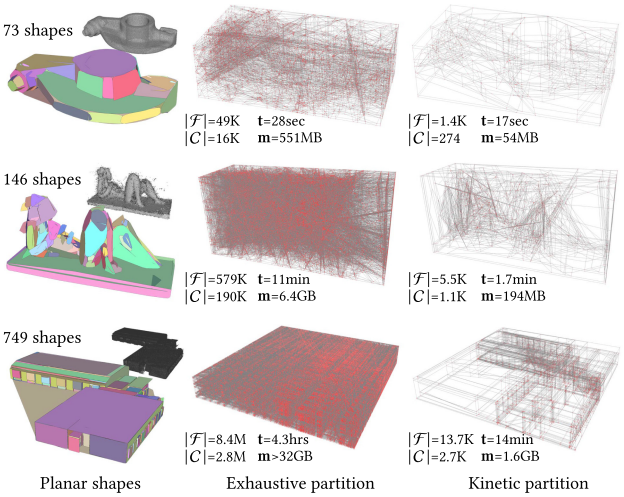


Fig. 14. Exhaustive *vs* kinetic partitionings. By naively slicing all planes supporting the shapes, exhaustive partitions are more complex than our kinetic partitions. The construction of kinetic partitions is also faster and consumes less memory, especially from complex configurations of planar shapes (see middle and bottom examples). $|\mathcal{F}|$, $|C|$, **t**, and **m** refer to the number of facets in the partition, the number of polyhedra, the construction time of the partition, and the memory consumption, respectively. Models from top to bottom: Rocker arm, Capron, and Navis.

The costly operation for the surface extraction step is the computation of the voting function $d_i(p, x_i)$, which must be performed for each inlier point. The shape detection step, which is not a contribution of our work, typically requires few seconds for processing several millions of input points. In terms of scalability, our algorithm can handle tens of thousands of planar shapes on a standard computer without parallelization schemes.

Ablation study. We evaluated the impact of the kinetic partitioning and the surface extraction on the performances of our algorithm.

First, we compared the performances of exhaustive and kinetic partitionings. The top graphs of Figure 13 show the evolution of their construction time, memory-consumption, and complexity in function of the number of planar shapes. The construction of kinetic partitions is faster and consumes less memory. In particular, memory consumption is reduced by more than one order of magnitude from 150 planar shapes. Kinetic partitions are also significantly lighter with a reduction of the number of polyhedra by a factor close to the number of shapes when parameter K is fixed to 1. Figure 14 illustrates this gap of performances with visual results obtained from 73, 146, and 749 planar shapes.

We also measured from our kinetic partitions the efficiency of our graph-cut-based surface extraction against the integer programming formulation proposed in Polyfit [Nan and Wonka 2017], optimized with the Gurobi solver [Gurobi Optimization 2019]. Running time, memory consumption, and SMH error of output models in function of the number of polyhedra in the partition are plot in the bottom graphs of Figure 13. While SMH errors are similar from light partitions, our graph-cut formulation is significantly more scalable and stable. The integer programming solver typically fails to deliver results under reasonable time when partitions contain more than 10K polyhedra. Moreover, the convergence time of its branch-and-bound optimization strategy is hardly predictable: small variations on the energy parameters can increase running times by more than three orders of magnitude for a given partition. In contrast, our solver can digest millions of polyhedra in a few minutes, and running times only depend on the numbers of polyhedra and input points. Our graph-cut solver also requires less memory with a consumption that evolves linearly at a rate of 2 KB per polyhedron.

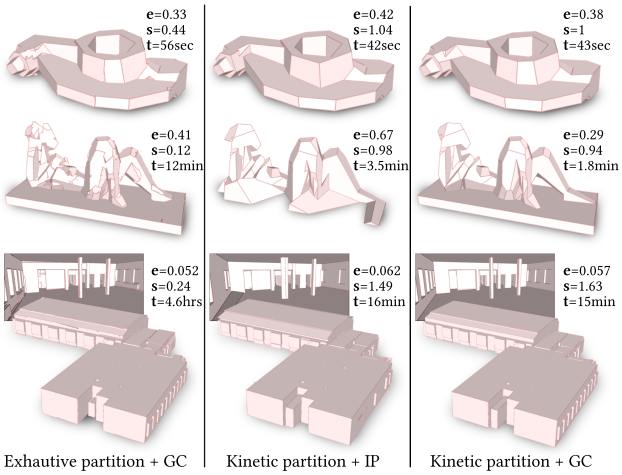


Fig. 15. Ablation study. Combining exhaustive partitions with our graph-cut (GC) solver produces accurate models resulting from the exploration of a large solution space. However, this combination is not time-efficient and returns models with a low simplicity score illustrated by numerous visual artifacts (left). Plugging the integer programming (IP) solver of Polyfit on kinetic partitions is a more efficient—yet less accurate—option (middle). Our framework offers the best performances and also the best compromise between accuracy and output simplicity (right). Models from Figure 14.

Finally, we measured the quality of output models obtained from different combinations of partitioning schemes and surface extraction solvers in Figure 15. The combination of kinetic partitioning with our graph-cut solver delivers the best results in terms of output simplicity and performances, especially when more than 100 shapes are handled. Using exhaustive partitions with our graph-cut solver allows to strongly extend the solution space: This typically produces more accurate models, but at the expense of output simplicity, performance, and scalability. Plugging the integer programming solver of Polyfit [Nan and Wonka 2017] on kinetic partitions is an efficient option. However, this solver operates on restricted solution spaces in which candidate facets cannot lie on the object bounding box. This restriction both prevents us from using spatial subdivision schemes for increasing scalability and strongly decreases geometric accuracy of output models in case of occlusions located on the object borders (see the missing ground in the Capron model). The ablation study also shows that the performance gain obtained by our solution mainly comes from the kinetic partitioning step. Note that operating the Polyfit solver from exhaustive partitions was not considered in this study: This combination is clearly less efficient than the three tested ones. In particular, it cannot handle more than 100 shapes in practice.

Comparisons with surface reconstruction methods. We compared our algorithm with Polyfit [Nan and Wonka 2017] and Chauve's method [Chauve et al. 2010], which are arguably the two most robust existing methods in the field. To fairly compare the reconstruction mechanisms, we used the same configuration of planar shapes for all methods. In particular, we deactivated the shape completion module in Chauve's method. Figure 16 shows the results obtained on Hand from 75 planar shapes. Because Polyfit and Chauve's method naively slice the object bounding box by the planar shapes, they produce overly dense partitions. For example, their exhaustive partition for Hand is composed of 30K cells and 91K facets, whereas our partition contains 0.7K cells and 3.5K facets only. Extracting output surface from a much lighter partition thus becomes faster and less memory-consuming. Our algorithm also outperforms Polyfit and Chauve's method in

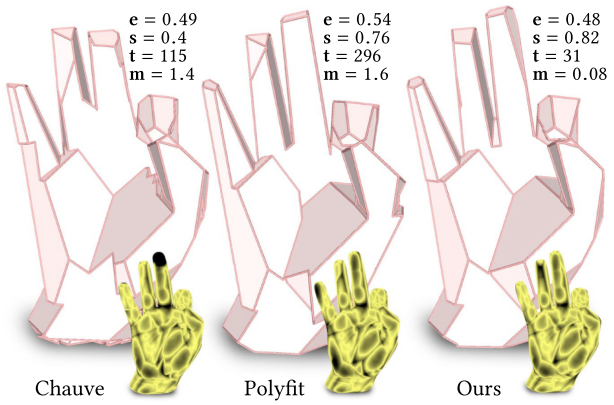


Fig. 16. Comparisons with reconstruction methods. Starting from the same 75 planar shapes, Chauve's method [Chauve et al. 2010] and Polyfit [Nan and Wonka 2017] are both more time- and memory-consuming than our algorithm while delivering less concise polygonal meshes, both less accurate and less compact. **e**, **s**, **t**, and **m** correspond to the symmetric mean Hausdorff error (in % of the bounding box diagonal), simplicity, running time (in sec) and memory peak (in GB), respectively. The colored point clouds show the error distribution (yellow=0, black≥ 1% of the bounding box diagonal).

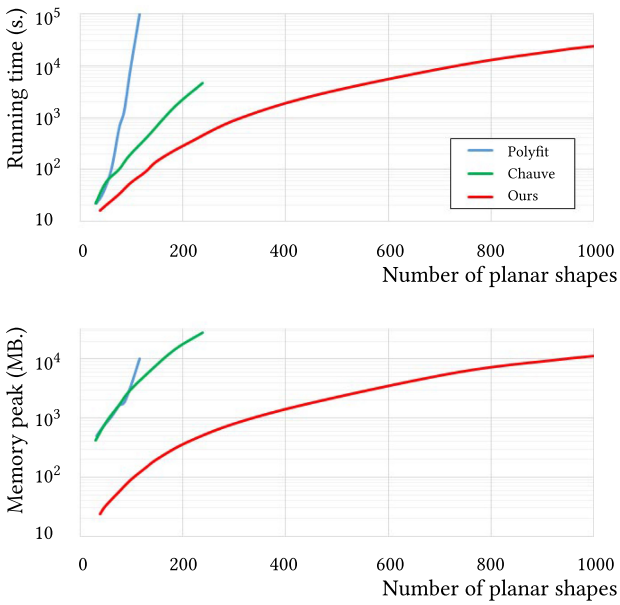


Fig. 17. Performances of surface reconstruction methods. Polyfit [Nan and Wonka 2017] requires days of computing for assembling 100 shapes, whereas Chauve's method [Chauve et al. 2010] exceeds 32 GB memory for slightly more than 200 shapes. Our algorithm is one order of magnitude more scalable than these two methods. Tests have been performed on the Hand model (Figure 16) without using subdivision schemes.

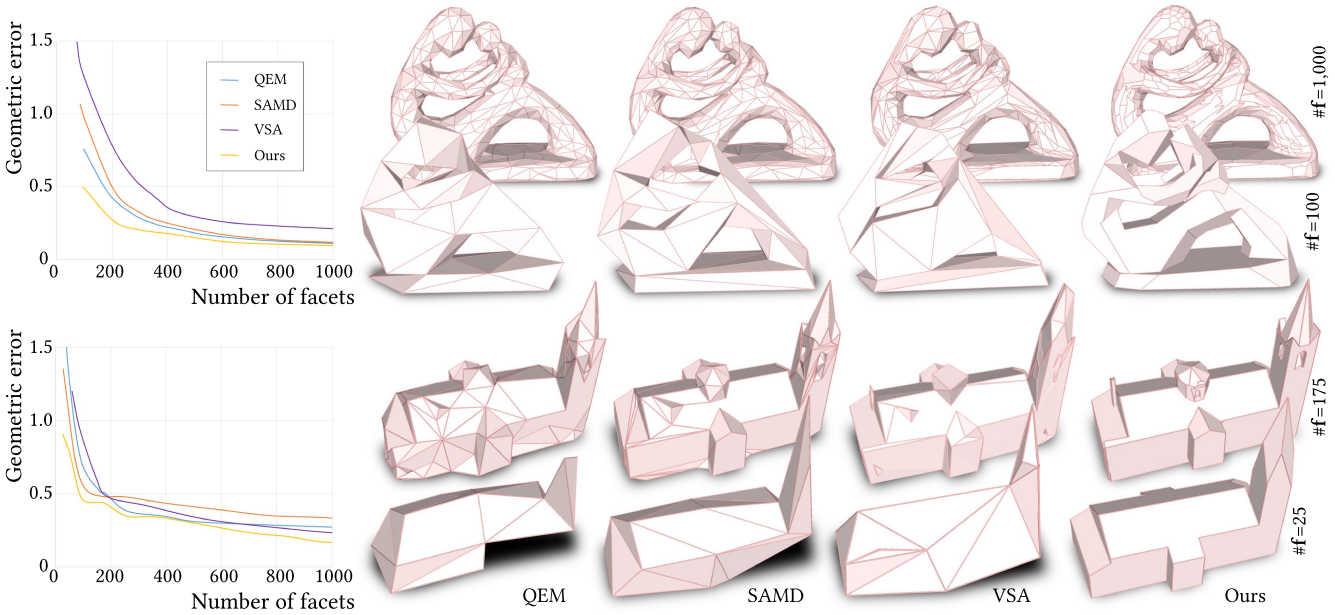


Fig. 18. Comparisons with surface approximation methods. The left curves, which measure the geometric error in function of the number of facets of the output mesh, show that our algorithm outclasses simplification methods for both a freeform object (Fertility, top) and a nearly piecewise planar structure (Lans, bottom). QEM [Garland and Heckbert 1997] cannot output large meaningful facets, whereas SAMD [Salinas et al. 2015] and VSA [Cohen-Steiner et al. 2004] fail to correct the geometric inaccuracies contained in the dense triangle mesh. For each row, the four meshes contain the same number of facets #f. The geometric error is measured as the symmetric mean Hausdorff distance between input points and output mesh.

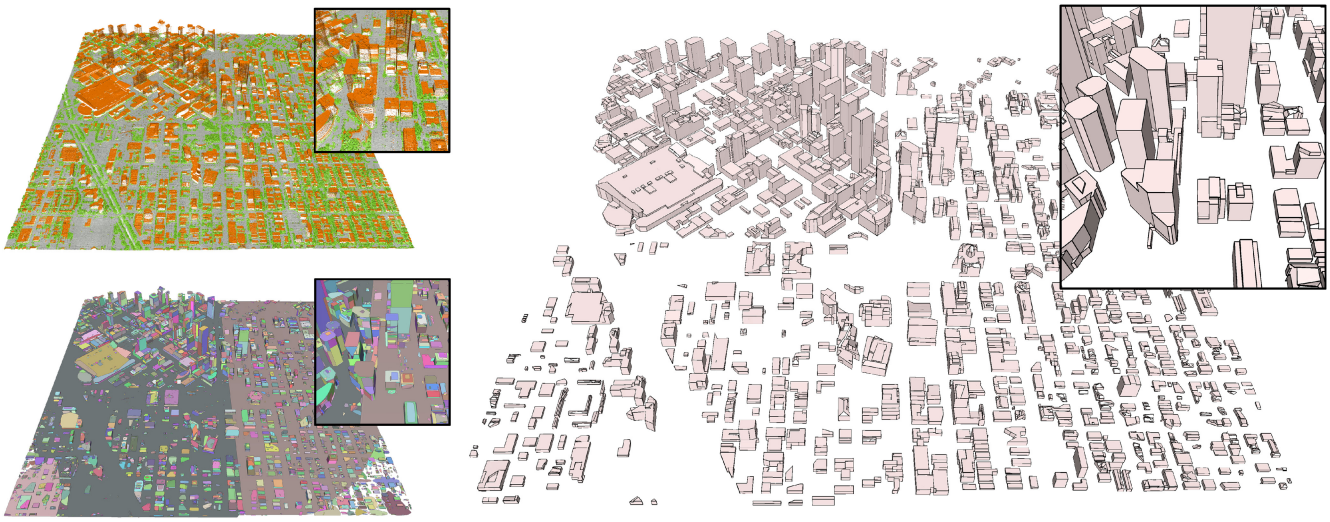


Fig. 19. Application to city modeling. We use our algorithm to reconstruct the downtown of Denver, Colorado, in the U.S., from an airbone Lidar scan of 15.1M points. After classifying input points into *ground* in grey, *building* in red, and *other* in green (top left point cloud) and after detecting 13K planar shapes from *building* and *ground* points (bottom left set of polygons), our algorithm reconstructs the 703 buildings of the scene with a CityGML-LOD2 representation (right). The output mesh is composed of 12.9K facets only.

terms of fidelity and simplicity. Our surface extraction is more efficient in balancing between fidelity and simplicity. The main reason relies on the use of a simpler data term where only inlier points are taken into account to measure the faithfulness. This technical choice offers more robustness to imperfect data than the visibility criterion of Chauve's method and more stability than the three-term energy of Polyfit.

Graphs presented in Figure 17 compared the scalability of the methods. Polyfit in its Gurobi solver version may require days of computing when more than 100 planar shapes are handled, whereas Chauve's method exceeds 32 GB memory for processing slightly more than 200 shapes. In particular, the former exploits a time-consuming integer programming solver while the latter suffers from a memory-consuming visibility estimation. Conversely,

our algorithm with no subdivision scheme can process 1K planar shapes under more reasonable processing times and without exceeding 32 GB memory. More qualitative and quantitative results are provided in supplementary material and in Bauchet [2019].

Comparisons with surface approximation pipelines. We also compared our algorithm with the surface approximation methods QEM [Garland and Heckbert 1997], SAMD [Salinas et al. 2015], and VSA [Cohen-Steiner et al. 2004]. Because these methods operate from meshes, we first reconstructed a dense triangle mesh from input points using the screened Poisson algorithm [Kazhdan and Hoppe 2013]. Both a freeform object (Fertility) and a nearly piecewise-planar structure (Lans) were used for this comparison. As shown in Figure 18, polyhedral meshes produced by our algorithm are geometrically more accurate than those returned by these three methods at a similar mesh complexity, i.e., with the same number of output facets. The accuracy gain is particularly high at low mesh complexity where approximation methods cannot capture correctly the structure of the object and tend to shrink the output surface. By operating directly from input points, our algorithm does not depend on an intermediate dense mesh reconstruction step in which geometric and topological errors are likely to occur. Moreover, the large polygonal facets returned by our algorithm approximate the object more effectively than the triangle facets returned by edge contraction (QEM and SAMD) or constrained Delaunay triangulation built from a primitive connectivity graph (VSA). In terms of performance, the approximation pipelines have similar processing times and memory consumption than our algorithm when producing low complexity meshes. For instance, the four methods process the Lans model with 25 output facets (see bottom row in Figure 18) in approximately half a minute. High complexity meshes, i.e., with more than 200 facets, are produced more quickly by the approximation pipelines than by our algorithm. More qualitative and quantitative results can be found in supplementary material.

Limitations. Our work focuses on assembling planar shapes, not on detecting them. We assume planar shapes can be accurately detected from input points by standard algorithms. However, this assumption might be wrong in presence of data highly corrupted by noise and severe occlusions. For such cases, configurations of planar shapes returned by existing algorithms are typically inaccurate and incomplete. Our kinetic algorithm can only correct configurations where missing planar shapes are parts of a repetitive structure as illustrated in Figure 5. Our surface extraction solver is also not designed to reconstruct non-manifold geometry where some object parts have no thickness. By using an inside-outside labeling, our solver forces each object part to have a non-zero thickness. Consequently, a non-manifold object will be either shrunk or overly-simplified around its zero thickness components. For such objects, the flexibility of an integer programming solver is probably a better choice here. Finally, the initialization of non-convex shapes by their convex hulls can create extra cuttings. One solution to build even lighter partitions could be to decompose their alpha-shapes into convex polygons. Increasing the number of polygons would however decrease performances.

7 APPLICATION TO CITY MODELING

We now show how our algorithm can be used in a concrete application such as city modeling from airborne Lidar data. City modeling mainly consists in reconstructing buildings with a piecewise-planar geometry [Musialski et al. 2013]. Before applying our algorithm, we first need to distinguish buildings from other urban objects contained in Lidar scans. We relied on the classification package of the CGAL Library [Giraudot and Lafarge 2019] for this task. We considered three classes of interest: *building*, *ground,* and *other*. We ran the Random Forest classifier with the default geometric features. As second step, we then detected planar shapes from points labeled as *building*. Because Lidar scans are typically acquired at nadir, facades are mostly occluded and not captured by planar shapes. We thus consolidated the set of detected shapes by adding artificial vertical planar shapes at the building contours. We computed an elevation map (or Digital Surface Model) from the raw Lidar scan by planimetric projection and detect line-segments located along high gradients by the LSD algorithm [Von Gioi et al. 2010]. The 2D line-segments were finally lifted into vertical rectangles to form the set of extra planar shapes. The last step consisted in applying our kinetic partitioning and surface extraction from the consolidated set of planar shapes. Figure 19 illustrates these steps on the city of Denver, Colorado, in the U.S. The output mesh is both compact with only 12.9K facets for describing 703 buildings, and accurate with a mean Hausdorff distance to the input points labeled as *building* of 0.52 meter.

8 CONCLUSION

We proposed an algorithm for converting point clouds into concise polygonal meshes. The key technical ingredient of the algorithm is a kinetic data structure where planar shapes grow at constant speed until colliding and partitioning the space into a low number of polyhedra. This idea does not only bring efficiency and scalability with respect to existing methods, it also allows us to deliver more concise polygonal meshes extracted from lighter yet more meaningful partition of polyhedra. We demonstrated the robustness and efficiency of our algorithm on a variety of objects and scenes in terms of complexity, size, and acquisition characteristics. We also showed the applicability of our algorithm in city modeling for reconstructing large-scale airborne Lidar scans.

In future work, we will generalize the kinetic partitioning algorithm to non-planar shapes using NURBS and other parametric functions. Although the collision detection for such shapes is even more challenging to compute efficiently, this perspective would allow us to reconstruct freeform objects with a better complexity-distortion tradeoff. We also plan to investigate the potential of our kinetic data structure for repairing CAD models.

ACKNOWLEDGMENTS

We thank our anonymous reviewers for their input, Qian-Yi Zhou and Arno Knapitsch for providing us datasets from the Tanks and Temples benchmark (Meeting Room, Horse, M60, Barn, Ignatius, Courthouse, and Church), and Pierre Alliez, Mathieu Desbrun, and George Drettakis for their help and advice. We are also grateful to Liangliang Nan and Hao Fang for sharing comparison materials. Datasets Full thing, Castle, Tower of Pi,

and Hilbert cube originate from Thingi10K, Hand, Rocker Arm, Fertility, and Lans from Aim@Shape, and Stanford Bunny and Asian Dragon from the Stanford 3D Scanning Repository.

REFERENCES

Pankaj K. Agarwal, Jie Gao, Leonidas J. Guibas, Haim Kaplan, Vladlen Koltun, Natan Rubin, and Micha Sharir. 2010. Kinetic stable Delaunay graphs. In *Proceedings of the Symposium on Computational Geometry (SoCG'10)*.

Murat Arikan, Michael Schwarzler, Simon Flory, Michael Wimmer, and Stefan Maierhofer. 2013. O-Snap: Optimization-based snapping for modeling architecture. *Trans. Graph.* 32, 1 (2013).

Julien Basch, Jeff Erickson, Leonidas J. Guibas, John Hershberger, and Li Zhang. 2004. Kinetic collision detection between two simple polygons. *Comput. Geom.* 27, 3 (2004).

Jean-Philippe Bauchet. 2019. *Kinetic Data Structures for the Geometric Modeling of Urban Environments*. PhD thesis. Université Côte d'Azur, France.

Jean-Philippe Bauchet and Florent Lafarge. 2018. KIPPI: KInetic Polygonal Partitioning of Images. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR'18)*.

Matthew Berger, Andrea Tagliasacchi, Lee M. Seversky, Pierre Alliez, Gael Guennebaud, Joshua A. Levine, Andrei Sharf, and Claudio T. Silva. 2017. A survey of surface reconstruction from point clouds. *Comput. Graph. Forum* 36, 1 (2017).

Andras Bodis-Szomoru, Hayko Riemenschneider, and Luc Van Gool. 2015. Superpixel meshes for fast edge-preserving surface reconstruction. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR'15)*.

Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. 2010. *Polygon Mesh Processing*. CRC Press.

Alexandre Boulch, Martin De La Gorce, and Renaud Marlet. 2014. Piecewise-planar 3D reconstruction with edge and corner regularization. *Comput. Graph. Forum* 33, 5 (2014).

Yuri Boykov and Vladimir Kolmogorov. 2004. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *Trans. Pattern Anal. Mach. Intell.* 26, 9 (2004).

Anne-Laure Chauve, Patrick Labatut, and Jean-Philippe Pons. 2010. Robust piecewise-planar 3D reconstruction and completion from large-scale unstructured point data. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR'10)*.

Desai Chen, Pitchaya Sitthi-Amorn, Justin T. Lan, and Wojciech Matusik. 2013. Computing and fabricating multiplanar models. *Comput. Graph. Forum* Vol. 32.

Jie Chen and Baoquan Chen. 2008. Architectural modeling from sparsely scanned range data. *Int. J. Comput. Vis.* 78, 2–3 (2008).

David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. 2004. Variational shape approximation. In *Proceedings of the SIGGRAPH Conference*.

James M. Coughlan and Alan L. Yuille. 2000. The Manhattan world assumption: Regularities in scene statistics which enable Bayesian inference. In *Proceedings of the Conference on Neural Information Processing Systems*.

Guillaume Damiand. 2018. Combinatorial maps. In *CGAL User and Reference Manual* (4.14 ed.). CGAL Editorial Board.

Hao Fang, Florent Lafarge, and Mathieu Desbrun. 2018. Planar shape detection at structural scales. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR'18)*.

Michael Garland and Paul S. Heckbert. 1997. Surface simplification using quadric error metrics. In *Proceedings of the SIGGRAPH Conference*.

Simon Giraudot and Florent Lafarge. 2019. Classification. In *CGAL User and Reference Manual* (4.14 ed.). CGAL Editorial Board.

Leonidas J. Guibas. 2004. Kinetic data structures. In *Handbook of Data Structures and Applications*. Routledge.

Leonidas J. Guibas, Lyle Ramshaw, and Jorge Stolfi. 1983. A kinetic framework for computational geometry. In *Proceedings of the Symposium on Foundations of Computer Science*.

Gurobi Optimization LLC. 2019. Gurobi Optimizer Reference Manual. http://www.gurobi.com.

Thomas Holzmann, Michael Maurer, Friedrich Fraundorfer, and Horst Bischof. 2018. Semantically aware urban 3D reconstructionwith plane-based regularization. In *Proceedings of the European Conference on Computer Vision (ECCV'18)*.

Jin Huang, Tengfei Jiang, Zeyun Shi, Yiying Tong, Hujun Bao, and Mathieu Desbrun. 2014. L_1-based construction of polycube maps from complex shapes. *Trans. Graph.* 33, 3 (2014).

Satoshi Ikehata, Hang Yan, and Yasutaka Furukawa. 2015. Structured indoor modeling. In *Proceedings of the International Conference on Computer Vision (ICCV'15)*.

Caigui Jiang, Chengcheng Tang, Amir Vaxman, Peter Wonka, and Helmut Pottmann. 2015. Polyhedral patterns. *Trans. Graph.* 34, 6 (2015).

Caigui Jiang, Jun Wang, Johannes Wallner, and Helmut Pottmann. 2014. Freeform honeycomb structures. *Comput. Graph. Forum* 33 (2014).

Adrien Kaiser, Jose Alonso Ybanez Zepeda, and Tamy Boubekeur. 2018. A survey of simple geometric primitives detection methods for captured 3D data. *Comput. Graph. Forum* 37 (2018).

Michael Kazhdan, Matthew Bolitho, and Hughes Hoppe. 2006. Poisson surface reconstruction. In *Proceedings of the Symposium on Geometry Processing*.

Michael Kazhdan and Hugues Hoppe. 2013. Screened Poisson surface reconstruction. *Trans. Graph.* 32, 3 (2013).

Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. 2017. Tanks and temples: Benchmarking large-scale scene reconstruction. *Trans. Graph.* 36, 4 (2017).

Patrick Labatut, Jean-Philippe Pons, and Renaud Keriven. 2009. Robust and efficient surface reconstruction from range data. *Comput. Graph. Forum* 28, 8 (2009).

Florent Lafarge and Pierre Alliez. 2013. Surface reconstruction through point set structuring. *Comput. Graph. Forum*, Vol. 32.

Lingxiao Li, Minhyuk Sung, Anastasia Dubrovina, Li Yi, and Leonidas Guibas. 2019. Supervised fitting of geometric primitives to 3D point clouds. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR'19)*.

Yangyan Li, Xiaokun Wu, Yiorgos Chrysathou, Andrei Sharf, Daniel Cohen-Or, and Niloy J. Mitra. 2011. GlobFit: Consistently fitting primitives by discovering global relations. *Trans. Graph.* Vol. 30.

Peter Lindstrom. 2000. Out-of-core simplification of large polygonal models. In *Proceedings of the SIGGRAPH Conference*.

David Marshall, Gabor Lukacs, and Ralph Martin. 2001. Robust segmentation of primitives from range data in the presence of geometric degeneracy. *Trans. Pattern Anal. Mach. Intell.* 23, 3 (2001).

Aron Monszpart, Nicolas Mellado, Gabriel J. Brostow, and Niloy J. Mitra. 2015. Rapter: Rebuilding man-made scenes with regular arrangements of planes. *Trans. Graph.* 34, 4 (2015).

Claudio Mura, Oliver Mattausch, and Renato Pajarola. 2016. Piecewise-planar reconstruction of multi-room interiors with arbitrary wall arrangements. *Comput. Graph. Forum* 35, 7 (2016).

T. Murali and Thomas Funkhouser. 1997. Consistent solid and boundary representations from arbitrary polygonal data. In *Proceedings of the Symposium on Interactive 3D Graphics*.

Przemyslaw Musialski, Peter Wonka, Daniel G. Aliaga, Manuel Wimmer, Luc Van Gool, and Werner Purgathofer. 2013. A survey of urban reconstruction. *Comput. Graph. Forum* 32, 6 (2013).

Liangliang Nan and Peter Wonka. 2017. PolyFit: Polygonal surface reconstruction from point clouds. In *Proceedings of the International Conference on Computer Vision (ICCV'17)*.

Sven Oesau, Florent Lafarge, and Pierre Alliez. 2016. Planar shape detection and regularization in tandem. *Comput. Graph. Forum* Vol. 35.

Sven Oesau, Yannick Verdie, Clément Jamin, Pierre Alliez, Florent Lafarge, and Simon Giraudot. 2018. Point set shape detection. In *CGAL User and Reference Manual* (4.14 ed.). CGAL Editorial Board.

Tahir Rabbani, Frank Van Den Heuvel, and George Vosselman. 2006. Segmentation of point clouds using smoothness constraint. *Proceedings of the International Society for Photogrammetry and Remote Sensing Conference* 36, 5 (2006).

David Salinas, Florent Lafarge, and Pierre Alliez. 2015. Structure-aware mesh decimation. *Comput. Graph. Forum* 34, 6 (2015).

Falko Schindler, Wolfgang Worstner, and Jan-Michael Frahm. 2011. Classification and reconstruction of surfaces from point clouds of man-made objects. In *Proceedings of the International Conference on Computer Vision (ICCV'11) Workshops*.

Ruwen Schnabel, Patrick Degener, and Reinhard Klein. 2009. Completion and reconstruction with primitive shapes. *Comput. Graph. Forum* Vol. 28.

Ruwen Schnabel, Roland Wahl, and Reinhard Klein. 2007. Efficient RANSAC for point-cloud shape detection. *Comput. Graph. Forum* Vol. 26.

Johannes Lutz Schönberger and Jan-Michael Frahm. 2016. Structure-from-motion revisited. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR'16)*.

Marco Tarini, Kai Hormann, Paolo Cignoni, and Claudio Montani. 2004. PolyCube-maps. *Trans. Graph.* 23, 3 (2004).

Marc Van Kreveld, Thijs Van Lankveld, and Remco C. Veltkamp. 2011. On the shape of a set of points and lines in the plane. *Comput. Graph. Forum* 30 (2011).

Yannick Verdie, Florent Lafarge, and Pierre Alliez. 2015. LOD generation for urban scenes. *Trans. Graph.* 34, 3 (2015).

Rafael Von Gioi, Jeremie Jakubowicz, Jean-Michel Morel, and Gregory Randall. 2010. LSD: A fast line segment detector with a false detection control. *Trans. Pattern Anal. Mach. Intell.* 32, 4 (2010).

Qingnan Zhou and Alec Jacobson. 2016. Thingi10K: A dataset of 10, 000 3D-printing models. *arXiv:1605.04797* (2016).

Received October 2019; revised March 2020; accepted April 2020