# O-Snap: Optimization-Based Snapping for Modeling Architecture

MURAT ARIKAN
Vienna University of Technology
MICHAEL SCHWÄRZLER
VRVis Research Center
SIMON FLÖRY and MICHAEL WIMMER
Vienna University of Technology
and
STEFAN MAIERHOFER
VRVis Research Center

In this article, we introduce a novel reconstruction and modeling pipeline to create polygonal models from unstructured point clouds. We propose an automatic polygonal reconstruction that can then be interactively refined by the user. An initial model is automatically created by extracting a set of RANSAC-based locally fitted planar primitives along with their boundary polygons, and then searching for local adjacency relations among parts of the polygons. The extracted set of adjacency relations is enforced to snap polygon elements together, while simultaneously fitting to the input point cloud and ensuring the planarity of the polygons. This optimization-based snapping algorithm may also be interleaved with user interaction. This allows the user to sketch modifications with coarse and loose 2D strokes, as the exact alignment of the polygons is automatically performed by the snapping. The generated models are coarse, offer simple editing possibilities by design, and are suitable for interactive 3D applications like games, virtual environments, etc. The main innovation in our approach lies in the tight coupling between interactive input and automatic optimization, as well as in an algorithm that robustly discovers the set of adjacency relations.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Geometric algorithms, languages, and systems*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Modeling packages*

General Terms: Algorithms

Additional Key Words and Phrases: Surface reconstruction, geometric optimization, interactive modeling

**ACM Reference Format:**

Arikan, M., Schwärzler, M., Flöry, S., Wimmer, M., and Maierhofer, S. 2013. O-Snap: Optimization-based snapping for modeling architecture. ACM Trans. Graph. 32, 1, Article 6 (January 2013), 15 pages.
DOI = 10.1145/2421636.2421642
http://doi.acm.org/10.1145/2421636.2421642

6

## 1. INTRODUCTION

Modeling and reconstruction of buildings poses a challenge for both current research efforts as well as industrial applications. While data acquisition processes and techniques have experienced enormous advances over the last years, the ensuing processing and modeling steps are by far not as sophisticated and unproblematic to handle. Reasons for this are on the one hand the large size and noisiness of the point cloud data gathered from laser scans, photogrammetric approaches, or stereo cameras, and on the other hand the absence of suitable modeling tools and techniques that can handle the complexity of large 3D point clouds.

Converting raw input data into 3D models suitable for applications like games, GIS systems, simulations, and virtual environments therefore remains a complex and time-consuming task suitable for skilled 3D artists only. Modeling applications like Google SketchUp and its Pointools [2011] plugin address this issue by proposing simplified user interfaces and interoperability with other products like Street View (from where the artist can for example retrieve photogrammetric data). However, the modeling process remains cumbersome and time consuming, as the accuracy of the reconstruction depends on the skills and patience of the user. In contrast, our system automatically maintains the fitting to the input point cloud during the whole reconstruction and modeling process.

This work was partially supported by the Austrian Research Promotion Agency (FFG) through the FIT-IT project "Terapoints", project no. 825842. The competence center VRVis funded by BMVIT, BMWFJ, and City of Vienna (ZIT) within the scope of COMET – Competence Centers for Excellent Technologies. The program COMET is managed by FFG.
Authors' addresses: M. Arikan (corresponding author), Vienna University of Technology, Institute of Computer Graphics and Algorithms, Favoritenstrasse 9-11/E186, A-1040 Vienna, Austria; email: marikan@cg.tuwien.ac.at; M. Schwärzler, VRVis Research Center, Donau-City-Strasse 1, A-1220 Vienna, Austria; S. Flöry, M. Wimmer, Vienna University of Technology, Institute of Computer Graphics and Algorithms, Favoritenstrasse 9-11/E186, A-1040 Vienna, Austria; S. Maierhofer, VRVis Research Center, Donau-City-Strasse 1, A-1220 Vienna, Austria.

DOI 10.1145/2421636.2421642
http://doi.acm.org/10.1145/2421636.2421642

Fully automatic reconstruction approaches may omit any user interaction, but can hardly deliver satisfying results in case of erroneous and/or partly missing data. We overcome these limitations by letting a user guide the geometry completion. Although our system is interactive, we strongly benefit from tight coupling of user interaction and automatic reconstruction techniques.

In this article, we present a new approach to modeling 3D buildings from measured point cloud data. Our approach is mainly based on the observation that many man-made objects, and especially buildings, can be approximated and modeled using planar surfaces with piecewise linear outlines as their primary elements, as long as the desired level of detail is not too high. There are two main insights that drove this research: (1) in order to create 3D models suitable for virtual environments, the system needs to propose an initial solution that already abstracts from the deficiencies of the input data, like noise, missing elements, etc. And (2), modeling requires a tight coupling of interactive input and automatic optimization. We therefore propose a modeling pipeline that first creates a coarse polygonal model from an input point cloud, based on the decomposition of the points into subsets associated with fitted planes and subsequent polygon boundary extraction. The most important step is an optimization-based algorithm that snaps adjacent parts of the model together. This algorithm is then repeatedly carried out during the interactive modeling phase to facilitate modeling. Our work exploits certain characteristics in the input data to provide an optimized reconstruction of piecewise planar surfaces with arbitrary topologies, which is precise on the one hand, and comprises a very low number of faces on the other hand.

Our main contributions are:

—a new polygonalization pipeline for point clouds that abstracts polygon outlines to reasonable shapes even in the presence of high amounts of noise and outliers, that is easy to implement, and most importantly maintains interactivity during the modeling process;

—a new optimization-based snapping algorithm for polygon soups, where the novelty lies in a robust discovery of adjacency relationships;

—a new interactive modeling paradigm based on 2D sketching combined with interactive optimization-based snapping, allowing the user to model with coarse strokes.

## 2. RELATED WORK

The challenge of quickly generating 3D models of architectural buildings from images, videos, or sparse point clouds has received tremendous interest lately. Although significant success has been achieved with both semi- and fully automatic systems such as from Werner and Zisserman [2002], Schindler and Bauer [2003], Chen and Chen [2008], Furukawa et al. [2009], and Vanegas et al. [2010], as well as interactive systems such as from Debevec et al. [1996], van den Hengel et al. [2007], Sinha et al. [2008], and Nan et al. [2010], these systems either require a greater amount of manual intervention or have strict assumptions on the model to be reconstructed. We refer the reader to the recent survey by Musialski et al. [2012] for a comprehensive overview of urban reconstruction algorithms.

The automatic reconstruction work of Chen and Chen [2008] introduces a method to reconstruct polygonal faces of the model by searching for Hamiltonian circuits in graphs. They assume the existence of the complete set of planes and their neighboring information. Two planes are assumed to be adjacent if the minimum distance between their corresponding point sets is within a distance parameter. Due to erroneous and missing data prevalent in real-world datasets, we believe that one of the most challenging problems in automatic reconstruction remains the determination of neighboring information and that more sophisticated algorithms are needed to solve this problem.

Image-based approaches [Werner and Zisserman 2002; Schindler and Bauer 2003] generate coarse approximations consisting of mutually orthogonal planes. The coarse models are then refined with predefined shapes to add details such as windows, doors, and wedge blocks.

In their seminal work, Debevec et al. [1996] introduced a hybrid method that combines geometry-based modeling with image-based modeling into one pipeline, in which the user matches edges in the photographs to the edges in the model. Parameters and relative positions of model components as well as camera parameters are computed by minimizing a nonlinear photogrammetric objective function.

Recently, Nan et al. [2010] presented a system, the so-called *SmartBoxes*, to quickly model architectural buildings directly over 3D point clouds. SmartBoxes assumes Manhattan world scenes [Furukawa et al. 2009; Vanegas et al. 2010], and is great to reconstruct facades with repetitive axis-aligned structures. Compared to SmartBoxes, our system doesn't make any assumptions on the shape of planar surfaces, their mutual alignments, and orientations.

Sinha et al. [2008] introduce an interactive system to generate textured models of architectural buildings from a set of unordered photographs. The user sketches outlines of planar surfaces of the scene by directly drawing on top of photographs. The drawing process is made easier by snapping edges automatically to vanishing point directions and to previously sketched edges. Compared to our approach, their system still needs a precise drawing of polygons, since they do not automatically estimate polygon boundaries, and snapping is induced by a simple proximity criteria, while our system reliably extracts adjacency relations between elements of the polygons.

The VideoTrace system proposed by van den Hengel et al. [2007] interactively generates 3D models of objects from video. The user traces polygon boundaries over video frames. While in the work of Sinha et al. [2008] a single image is sufficient to accurately reconstruct polygonal faces, VideoTrace repeatedly reestimates the 3D model by using other frames.

A large number of mesh reconstruction methods have been proposed over the years: popular approaches include *extended marching cubes* by Kobbelt et al. [2001], the *Delaunay refinement paradigm* [Boissonnat and Oudot 2005], and *implicit* approaches [Kazhdan et al. 2006; Alliez et al. 2007; Schnabel et al. 2009]. Typically, all these methods need to generate high-resolution meshes in order to recover sharp edges. More recently, Salman et al. [2010] have improved the accuracy of reconstructions for a prescribed mesh size while ensuring a faithful representation of sharp edges. Still, their method does not yield models of low face count and visual quality as required in this work (refer to Figure 15). Instead of applying an extensive postprocessing pipeline (e.g., centered around Cohen-Steiner et al.'s [2004] shape approximation), we propose to integrate all requirements into a single optimized algorithm: our method fits median planes to the input data, accurately reconstructs sharp edges from the intersection of these planes, and generates coarse polygonal models (with the complexity defined by the number of planes). Most importantly, our method is good at handling surfaces with boundaries: all our input data is incomplete, with missing parts and holes due to the acquisition process.

Recent commercial systems such as SketchUp have been designed to quickly create 3D models from users' sketches. The Pointools [2011] plugin for SketchUp allows users to model directly over 3D point clouds, but the accuracy of the reconstruction

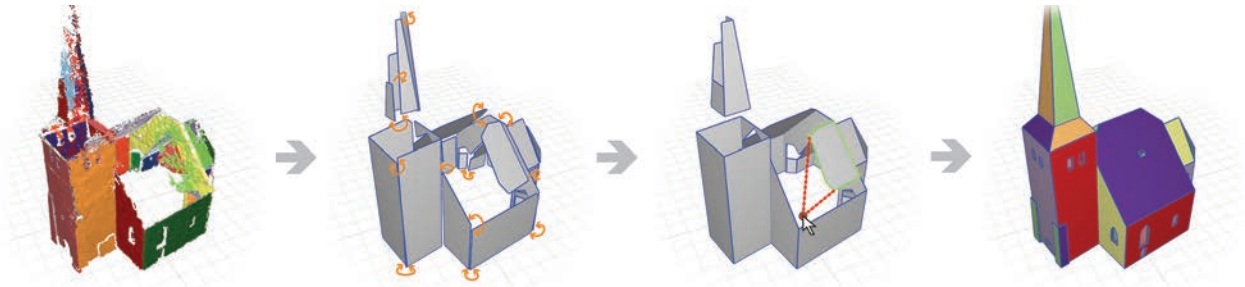

Fig. 1. An overview of our reconstruction and modeling pipeline, from left to right: Starting from a noisy and incomplete point cloud, we decompose the input data into subsets lying approximately on the same plane (left). The boundary points of each subset are extracted and used to estimate coarse polygons (middle left). Local adjacency relations are automatically discovered (middle left) and enforced via a nonlinear optimization to snap polygons together, providing an initial reconstruction (middle right) that can then be interactively refined by our optimization-aided sketch-based interface within a few clicks (middle right), yielding a coarse polygonal model that well approximates the input point cloud (right).

depends on the skills and patience of the user, since the sketched geometry has to be manually aligned to the point cloud by visual inspection. In addition, the plugin offers a simple snapping tool that allows snapping the endpoint of a sketched line to a nearby point in the point cloud. In practice, this feature can be used to coarsely sketch the floorplan of a building, but is impractical for more detailed modeling tasks due to the noise inherent in point clouds, as there is no way to align a primitive to a set of points. Furthermore, gaps that appear in the modeling process have to be closed manually by moving edges. In contrast, we optimally reconstruct planar primitives by least-median fitting to the point data. Moreover, we automatically discover adjacency relations, which allows us to run a planarity-preserving optimization-based snapping algorithm to close the model.

*GlobFit*, recently introduced by Li et al. [2011], iteratively learns mutual relations among primitives obtained by the RANSAC algorithm [Schnabel et al. 2007]. Their system seems to be complementary to ours: While they focus on discovering global relations (like orthogonality, coplanarity, etc.) among parts of the model to correct the primitives, our strength lies in the automatic discovery of local adjacency relations between polygon elements. To produce a final model (which is not their primary goal), Li et al. [2011] only extrapolate and compute pairwise primitive intersections. While it is relatively easy to locally extend individual polygons, intersections of multiple primitives can be highly complex and reconstruction becomes nontrivial.

## 3. OVERVIEW

Our system takes as input a set of 3D points, for example, from a laser scanner, photogrammetric reconstruction, or similar source. The goal is to create a polygonal model suitable for interactive applications and not influenced by the noise and holes inherent in the input data. The reconstructed polygonal model will be watertight wherever feasible; in this article we shall denote such a model a *closed model*. It is important to emphasize that we do not assume our target surfaces to be closed in a topological sense.

Modeling consists of two phases: an automatic phase that creates an initial model, and an interactive phase that is aided by optimization-based snapping.

### 3.1 Automatic Phase

In the automatic phase, the input data is decomposed into subsets lying approximately on the same plane (Figure 1, left). Throughout the article, we refer to these subsets as *segments*. For each such segment, boundary polygons are estimated in the *polygonalization* step (Section 4, Figures 1 middle left and 3). The resulting model still has holes and is not well aligned.

We therefore introduce an intelligent *snapping algorithm* (Section 5) that constrains and optimizes the locally fitted planes and their corresponding polygons. The local fit of the planes is determined by how well the planes approximate the observed point cloud data, while the mutual spatial relations, that is, adjacency relations between polygon elements, are iteratively computed and enforced through a nonlinear optimization. This "intelligent snapping" is a crucial part of our approach: Instead of simply snapping to existing geometry or features within a given distance [Sinha et al. 2008; van den Hengel et al. 2007], we define a feature-sensitive matching and pruning algorithm to discover a robust set of adjacency relations among parts of the polygons (Figure 1, middle left). The polygons are then aligned by enforcing the extracted relations, while best fitting to the input data and maintaining the planarity of the polygons (Figure 1, middle right).

Note that while a completely automatic reconstruction of a whole building can hardly be achieved due to erroneous and missing data, our automatic phase produces results that are comparable to previous automatic systems, for example Chen and Chen [2008], who assume the existence of a complete set of planes.

### 3.2 Interactive Phase

Since the initial geometry proposal from the automatic phase can not guarantee a perfect solution in all cases, the interactive phase provides a simple and intuitive *sketch-based user interface* (Section 6) that directly interoperates with the optimization routines. Even though such manual interventions cannot be completely omitted, the novel system differs significantly from other 3D modeling techniques by exploiting the previous analysis: Due to the known supporting planes, the modeling complexity is reduced from a 3D to a 2D problem. This allows the user to model the necessary changes with a few simple and loose strokes on a flat layer, as the exact alignment is performed interactively by the optimization-based snapping algorithm (Figure 1 middle right).

## 4. POLYGONALIZATION

We use a local RANSAC-based method [Schnabel et al. 2007] to decompose the input point cloud into subsets (referred to as *segments*), each lying approximately on a plane, and a set of unclaimed

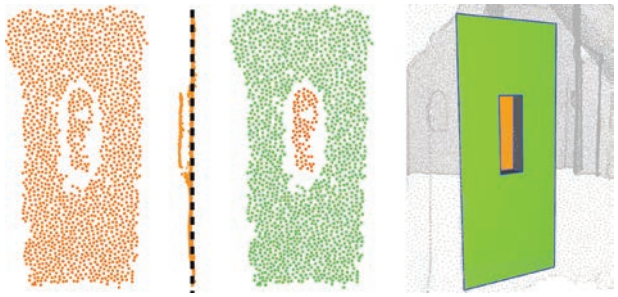

Fig. 2. The first step of the automatic reconstruction pipeline, a local RANSAC-based method, may capture nearby parallel structures (e.g., windows and facade) as a single segment (left). The least-median of squares method is used to fit a plane to the points of the dominant structure (middle left). By applying k-means clustering (k = 2) (and subsequent automatic polygonalization and optimization) in the interactive modeling phase, a more detailed hierarchical reconstruction (middle right, right) is achieved.

points. Even though the decomposition requires normal information, the correctness of normals close to sharp edges is not critical for the subsequent estimation of plane primitives (Section 4.1.1) and the rest of our pipeline. Hence, our implementation approximates 3D normal vectors from point positions by applying a local PCA [Jolliffe 2002] with fixed size neighborhoods.

A segment may consist of multiple connected components (e.g., front faces of all individual balconies on a facade), which are later separated by the boundary extraction algorithm (Section 4.1.2).

The goal of the polygonalization step is to divide the segments from the RANSAC stage into connected components, and approximate their outlines by coarse polygons. In the first step, we divide each segment into one or several connected components, and extract their ordered boundary points (Figure 3 top left), which act as initial polygons. Since the extracted boundaries are generally noisy, we cannot assume that we have high-quality vertex normal orientations. In the second step, we compute a smooth region around each point (Figure 3 top right), to which we then apply a local PCA to estimate initial 2D vertex normals (Figure 3 middle left). Finally, we reconstruct 2D vertex normals based on their initial values and a neighborhood relationship derived from the smooth regions (Figure 3 middle right), and use the reconstructed normal vectors to compute consistent vertex positions (Figure 3 bottom left). This process straightens the initial boundaries and thus provides a polygonal approximation of the 2D components on a coarse scale (Figure 3 bottom right).

## 4.1 Initialization

4.1.1 *Plane Fitting.* The polygonalization is computed in 2D space defined by the segment plane. The first step is therefore fitting a plane to all the points contained in a segment obtained by RANSAC. As depicted in Figure 2 (left), nearby parallel structures (e.g., main facade and windows) may have been detected as a single segment. We therefore apply a *Least Median of Squares* (LMS) estimator [Rousseeuw and Leroy 1987], which consistently finds the main structure (see Figure 2 middle left), as it is capable of fitting a model to data that contains up to 50% outliers.

4.1.2 *Boundary Extraction.* In order to divide each segment into connected components and extract their ordered boundary points, we employ *2D $\alpha$-shapes* [Edelsbrunner and Mücke 1994] (with $\alpha$ controlling the number of connected components and the level of detail of their boundaries) on the segment points projected

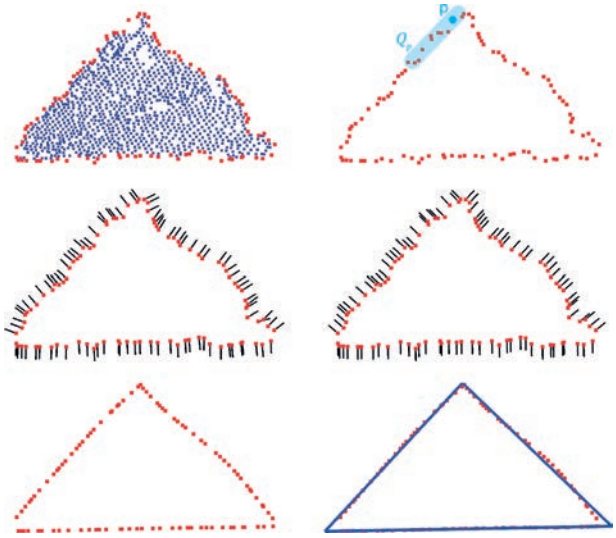

Fig. 3. Overview of our polygonalization pipeline. Ordered boundary points (initial polygon) of a connected component are extracted (top left). By applying PCA to smooth regions $Q$ (top right), vertex normals are initialized (middle left). Initial vertex normals are smoothed over neighboring vertices (with **p** and **q** neighboring if they are mutually contained in their respective smooth regions) (middle right) and used to compute consistent vertex positions (bottom left). Finally, a corner detection algorithm extracts the approximating polygon.

to the median plane. The family of 2D $\alpha$-shapes of the set of projected segment points $S$ is implicitly represented by the Delaunay triangulation of $S$. Each element (vertices, edges, and faces) of the Delaunay triangulation is associated with an interval that specifies for which values of $\alpha$ the element belongs to the $\alpha$-shape. The connectivity of the Delaunay triangulation and the classification of its elements with respect to the $\alpha$-shape is then used to extract the connected components and their ordered boundary points. We estimate $\alpha$ using the average distance between neighboring points.

## 4.2 Polygon Straightening

The goal of this step is to robustly estimate for each boundary a coarse polygon that approximates the original shape's outline. Most surfaces used in architecture, especially at the level of detail relevant for polygonal modeling, are bounded by straight lines that meet in sharp corners. We also require from our method that it is fast, since we need interactivity during the modeling process (see Section 6.6).

Our method is inspired by the $\ell_1$-sparse method [Avron et al. 2010] for the reconstruction of piecewise smooth surfaces. However, we found the $\ell_1$ formulation computationally too expensive, compared to our $\ell_2$ approach counting only a few or even no reweighted iterations (the $\ell_1$ formulation's second-order cone programming solver needs to solve a series of linear systems of comparable dimension to our setting). We tackle stability issues inherent to least-squares approaches (such as sensitivity to outliers) with statistical methods. In particular, we rely on the *forward search* method [Atkinson and Riani 2000] and present a novel method combining the simplicity of least-squares minimization with the strength of robust statistics.

4.2.1 *Neighborhood Estimation.* Similar to Fleishman et al.'s [2005] approach, we classify a locally smooth region around each

vertex by applying the forward search method, which preserves sharp features and is robust to noise and outliers. The main idea in forward search is to start from a small outlier-free neighborhood $Q$ and to iteratively extend the set $Q$ until a termination criterion is met. Starting from $Q$ and the model (in our case, a line) that is fitted to the points in $Q$, one iteratively adds one point to the set $Q$ (the point with lowest residual) and updates the model at each iteration, until the diameter of $Q$ exceeds a threshold $d_{max}$. Figure 3, top right, shows an example of a smooth region around a point $\mathbf{p}$ computed with forward search. Two polygon vertices $\mathbf{p}$ and $\mathbf{q}$ are said to be neighboring, if $\mathbf{q} \in Q_p$ and vice versa. The diameter threshold $d_{max}$ is the only parameter that affects the output of our polygonalization method. Since $d_{max}$ controls the local region sizes, we avoid high values of $d_{max}$ (usually set to minimum expected feature size) to prevent oversmoothing of sharp features.

4.2.2 *2D Normal Estimation.* We then estimate consistently oriented vertex normal vectors (Figure 3, middle right) based on the neighborhood relationship computed by the prior step. As in Avron et al. [2010], our least-squares minimization to reconstruct vertex normals consists of two terms and is formulated as

$$E_1 = \sum_{(p,q)\in N} w_{p,q} \|\mathbf{n}_p - \mathbf{n}_q\|^2 + \lambda \sum_p \|\mathbf{n}_p - \mathbf{n}_p^0\|^2. \quad (1)$$

The first term minimizes the normal differences and extends over the set $N$ of all neighboring vertices. The second term prevents the vertex normals $\mathbf{n}$ from deviating too much from their initial orientations $\mathbf{n}^0$. The weighting function $w_{p,q}$ in Eq. (1) penalizes variations in normal directions, and is given by the Gaussian filter

$$w_{p,q} = e^{-(\theta_{p,q}/\sigma)^2}, \quad (2)$$

where $\theta_{p,q}$ is the angle between the normal vectors $\mathbf{n}_p$ and $\mathbf{n}_q$ and $\sigma$ is a parameter set to 20 degrees in all our examples.

The normal vectors are initialized (refer to Figure 3 middle left) by applying PCA to the local smooth regions computed by the prior step. The consistency of the initial normal orientations is provided by the order of the boundary points.

4.2.3 *2D Polygon Smoothing.* We then compute consistent vertex positions (Figure 3, bottom left) by displacing polygon vertices in normal direction, that is,

$$\mathbf{p}' = \mathbf{p} + t_p \mathbf{n}_p.$$

Similar to Avron et al. [2010], the new vertex positions $\mathbf{p}'$ are computed as the minimizer of the energy function

$$E_2 = E_{2,s} + E_{2,init} \quad (3)$$

with

$$E_{2,s} = \sum_{(p,q)\in N} w_{p,q} \left( \left| (\mathbf{p}' - \mathbf{q}') \cdot \mathbf{n}_q \right|^2 + \left| (\mathbf{q}' - \mathbf{p}') \cdot \mathbf{n}_p \right|^2 \right)$$

and

$$E_{2,init} = \mu \sum_p t_p^2.$$

The first term smoothes (straightens) the polygon boundary by minimizing the deviation of $\mathbf{q}$ from the tangent through $\mathbf{p}$ weighted according to the confidence measure $w_{p,q}$ (Eq. (2)) and vice versa. The second term prevents the polygon vertices from deviating too much from their initial positions and as a consequence avoids shrinking of the polygon.

Both functionals $E_1$ and $E_2$ (Eqs. (1) and (3)) are minimized using a Gauss-Newton method. Since we use forward search to reconstruct outlier-free regions, we require only three reweighted iterations to minimize $E_1$ and a single iteration to minimize $E_2$. The weight parameters $\lambda$ and $\mu$ are set to 0.1 in our datasets.

### 4.3 Polygon Extraction

The process described before provides us with a set of straightened boundary points with high-quality vertex normals, which are then used by a simple corner detection algorithm to extract approximating coarse polygons (Figure 3, bottom right).

We classify a vertex as an edge vertex if its normal vector is almost parallel to the normal vector of the succeeding and preceding vertex, respectively. Then we approximate sequences of edge vertices in a least-squares sense and obtain edge lines. A corner is detected at the intersection point of two successive lines. Note that vertices that are isolated in our neighborhood relationship graph are potential outliers or influenced by noise inherent in the initial boundaries, and thus not used by the corner detection algorithm.

## 5. POLYGON SOUP SNAPPING

The result of the polygonalization step is a soup of unconnected polygons $\mathcal{P} = \{P_1, \ldots, P_n\}$. The snapping process aims at closing the holes between the polygons of the polygon soup, and is used both in the initial automatic reconstruction and during interactive modeling. Snapping iteratively pulls polygon vertices towards other polygons, while simultaneously refitting $\mathcal{P}$ to the underlying point cloud and preserving the planarity of polygons. Each iteration consists of the following two steps.

—*Robust search for adjacencies*, which for each vertex identifies the possible matches to other vertices, edges, or faces and discards false ones.

—*Optimization*, which enforces the set of discovered adjacency relations to snap the polygon soup together.

The process terminates when the polygon soup stabilizes, that is, it becomes a closed model and satisfies the requirements given by the constraints. Our snapping process is related to Botsch et al. [2006] and Kilian et al. [2008], however, our system requires the optimization for various other constraints and in particular, the relations between the polygons are not known a priori. We now describe the steps in detail.

### 5.1 Robust Search for Adjacencies

The problem of matching the elements of the polygon soup to a closed model in a feature-aware manner is inherently ill-defined. The expected bad quality of the real-world datasets and the lack of any high-level input to the reconstruction pipeline (such as shape templates or semantic information) prevent a rigorous mathematical definition. Instead, we propose an automatic and robust algorithm based on stable vertex-vertex/edge/face and edge-edge matches.

Adjacencies in the model are discovered by searching for matches between polygon elements. There are five mechanisms that constrain the allowed matches: (1) An auxiliary *global parameter* $r_{max}$ defines the maximal gap size to be closed in the model. (2) *Intrinsic stability* locally avoids self-intersections, flip-overs, edge and diagonal collapses. (3) An extended set of matching candidates allows more degrees of freedom (thus a more connected model) where (2) is too restrictive. (4) *Local pruning* fixes problems mostly introduced by (3), and (5) *global pruning* prevents degeneration of polygons (especially of thin features) by considering global issues

of matches affecting more than two polygons. We first describe the different match types, constrained by (1–3), and then show local and global pruning. Please note that the choice of $r_{max}$ doesn't influence the stability of the pruning algorithms, but only defines the maximal gap size.

5.1.1 *Vertex-Vertex Matching.* We define an adaptive search radius for each vertex of the model. The requirement of intrinsic stability bounds the search radius to half the minimal distance from $\mathbf{p}$ to the polygon's boundary, $d_\partial(\mathbf{p}) := \min_{\mathbf{e}\in\partial P\setminus\mathbf{p}} d(\mathbf{p}, \mathbf{e})$, where $\partial P \setminus \mathbf{p}$ denotes the polygon boundary after removal of $\mathbf{p}$ and its incident edges. Half the distance $d_\partial$ prevents vertices being matched across polygon edges (self-intersections, flip-overs) or vertices (edge or diagonal collapses). To respect the given upper bound, we define $r(\mathbf{p}) := \min(r_{max}, d_\partial(\mathbf{p})/2)$ as the *adaptive search radius* of $\mathbf{p}$.

The candidate set of matches for a vertex $\mathbf{p}$ comprises all vertices of $\mathcal{P} \setminus P$ within search distance $r(\mathbf{p})$. If this candidate set is empty, the closest vertex in $\mathcal{P} \setminus P$ (if not further than $r_{max}$) is included. By doing so, we may violate intrinsic stability intentionally to maintain sufficient degrees of freedom. A subsequent pruning step, described shortly, will restore validity at a later stage, if necessary. We define $r_c(\mathbf{p}) := \max(r(\mathbf{p}), \min(d_c, r_{max}))$ (with $d_c$ being the distance of $\mathbf{p}$ to its closest vertex in $\mathcal{P} \setminus P$) as the *extended search radius* of $\mathbf{p}$.

A priori, two vertices $\mathbf{p}$ and $\mathbf{q}$ are considered matching, if they are mutually included in their respective extended search radii,

$$\|\mathbf{p} - \mathbf{q}\| \leq \min(r_c(\mathbf{p}), r_c(\mathbf{q})).$$

Finally, two matching vertices are supposed to collapse into a corner point at the later optimization stage. Such a corner point is incident to the intersection line $l$ of the two corresponding supporting planes. Consequently, we further require a pair of matching vertices to be in feasible distance to $l$,

$$d(l, \mathbf{p}) \leq r_c(\mathbf{p}) \quad \text{and} \quad d(l, \mathbf{q}) \leq r_c(\mathbf{q}).$$

Please note that the computation of $l$ is numerically stable, as the RANSAC stage gives a priori knowledge about polygons in the same supporting plane.

5.1.2 *Vertex-Edge Matching.* In a similar fashion to vertex-vertex matches, we establish correspondences between vertices and edges. We assign a search radius to each edge $\mathbf{e} = (\mathbf{p}_0, \mathbf{p}_1)$ as the minimal search radius of its endpoints, $r(\mathbf{e}) = \min(r(\mathbf{p}_0), r(\mathbf{p}_1))$. A vertex $\mathbf{p}$ is matched to an edge $\mathbf{e}$ if its orthogonal projection onto the line spanned by $\mathbf{e}$ is in the edge's interior, and (analogous to a vertex-vertex match) the two following expressions hold true:

$$d(\mathbf{p}, \mathbf{e}) \leq \min(r(\mathbf{p}), r(\mathbf{e})),$$

and

$$d(l, \mathbf{p}) \leq r(\mathbf{p}) \quad \text{and} \quad d(l, \mathbf{e}) \leq r(\mathbf{e}),$$

where $l$ is the common intersection line of the corresponding supporting planes.

5.1.3 *Other Matches.* To complete the survey of matches, a vertex $\mathbf{p}$ is paired with a face $\mathbf{f}$ if its orthogonal projection onto the plane spanned by $\mathbf{f}$ is in the face's interior and no further than search radius $r(\mathbf{p})$.

Based on the vertex-vertex and vertex-edge matches, we may further derive edge-edge matches: Two edges are said to match if their endpoints either induce two vertex-vertex matches, a vertex-vertex and a vertex-edge match, or two vertex-edge matches. Edge-edge matches are used only in the matching and global pruning stage and not for optimization, as their contribution to reconstruction is implicitly included through vertex-vertex/edge matches. The same holds in an even stricter sense for edge-face and face-face matches, which are either implied by vertex-vertex/edge/face matches or are not present in the data due to occlusion in the acquisition process.

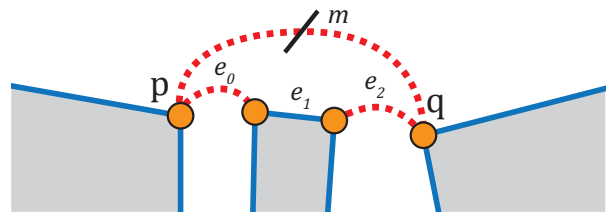


Fig. 4. The false match $m = (\mathbf{p}, \mathbf{q})$, which forces the center polygon's edge $e_1$ to collapse and thus violates the intrinsic stability, is reliably detected by our global pruning strategy.

5.1.4 *Local Pruning.* The vertex-vertex matching yields generally stable results. A few false matches, which result from the inclusion of closest vertices in candidate sets, are corrected in the following pruning step: Consider two or more vertices $\mathbf{q}_i$ of polygon $Q$ being matched to a vertex $\mathbf{p} \in P$. This clearly violates the intrinsic stability requirement for $Q$ and we remove all but the closest matching pair. This and all subsequent pruning steps are implemented on a graph representation $G = (V, E_M)$ of the matches, with all vertices and edges of $\mathcal{P}$ comprising $V$ and the edge set $E_M$ being given by the set of all matches obtained from before (except vertex-face matches). Pruning at this point boils down to investigating all one-ring neighborhoods of $G$.

Similar to vertex-vertex matching, intrinsic stability demands the pruning of those vertex-edge matches where a vertex corresponds to multiple nonadjacent edges of a polygon. This can naturally happen due to overlapping search cylinders (with radii $r(\mathbf{e})$) around edges. Using the graph $G$, we compress this subset of matches to the closest vertex-edge match.

5.1.5 *Global Pruning.* Up to now, the matches have been obtained on a local level only. They disregard any global issues affecting more than two polygons. Consider the situation in Figure 4, with three polygons stringing together and several of the corner points being matched. The vertex-vertex match between $\mathbf{p}$ and $\mathbf{q}$, jumping the center polygon, is not feasible, as it implies a degeneration of the center polygon's edge. Such a degeneration happens when certain polygon elements (vertices/edges) are connected through matches so that they form a *cycle*. The reason is that we have to assume that all polygon elements connected through matches might be joined to the same location during the optimization phase. In this section we therefore present our approach to define and find such cycles. We also show a second approach based on geometric tests for cases where the detection of cycles doesn't necessarily imply a degeneration.

For the example in Figure 4, to detect the contraction of the edge $e_1$, it would be sufficient to extend the edge set $E_M$ in the matching graph $G$ defined earlier by the actual polygon edges, and find the cycle $(m, e_0, e_1, e_2)$ in the resulting graph. However, there are many other cases that would lead to a polygon degeneration, namely, whenever a match would cause two elements of a polygon to contract; see Figure 5 for several examples.

We therefore introduce an *extended matching graph* $G_e$ that prevents all these internal contractions by representing them explicitly through so-called constraint edges. Formally, $G_e = (V, E_e)$ is a graph with $V$ comprising the polygons' elements (vertices and

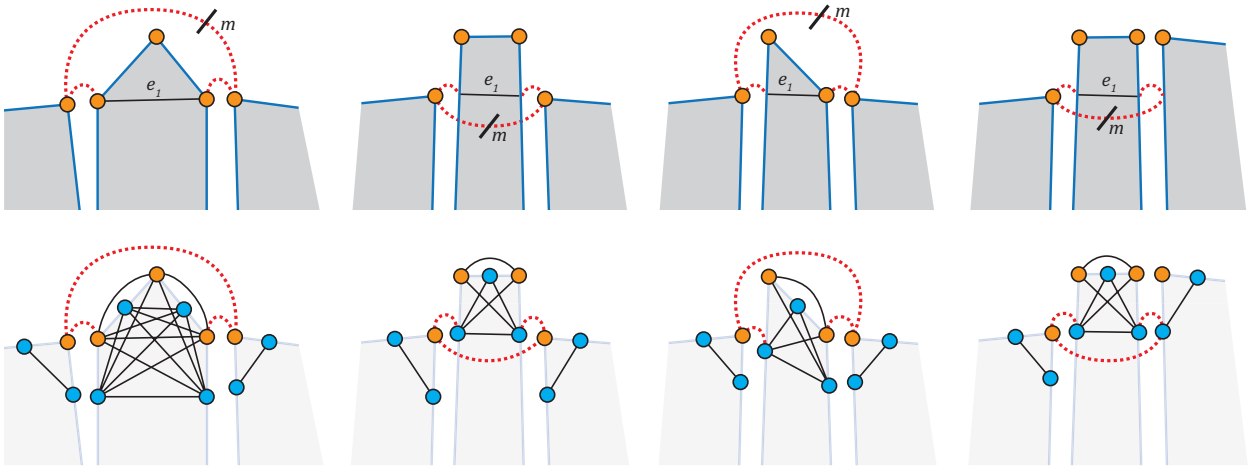


Fig. 5. Top row: Examples of false matches $m$ which would cause the constraint edges (denoted by $e_1$) to contract. Bottom row: We reliably avoid such false matches by searching for cycles in the extended matching graphs (only the matches of the cycles shown here).

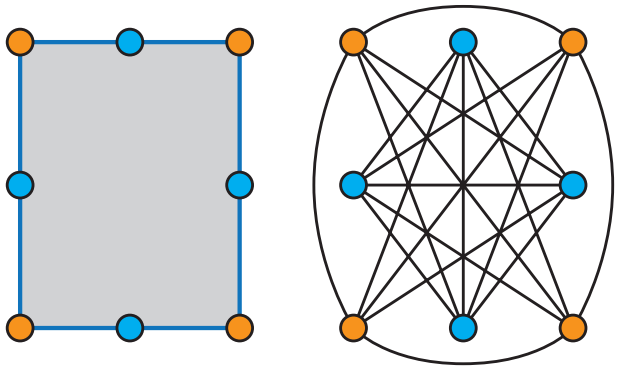

Fig. 6. Construction of the extended matching graph $G_e = (V, E_e)$: Each polygon's vertices (orange) and edges (blue) constitute the node set $V$ (left). The edge set $E_e$ combines the set of all vertex-vertex/edge and edge-edge matches (not shown here), and the constraint set. The latter connects all pairs of elements in $V$ of the same polygon, except polygon vertices with their incident polygon edges (right).

edges, see Figure 6 left). The edge set $E_e = E_M \cup E_C$ combines the set of matches $E_M$, and the *constraint set* $E_C$. The latter connects all pairs of elements in $V$ of the same polygon, except polygon vertices with their incident polygon edges (see Figure 6 right). Figure 5 shows several examples where the detection of an appropriate cycle containing a constraint edge (denoted by $e_1$) in $G_e$ causes a false match to be pruned.

Our strategy is now to determine for every vertex-vertex/edge match $m = (\mathbf{p}, \mathbf{q}) \in E_M$ whether it is part of a "harmful" cycle. We first note that we only look for cycles $c(m)$ where $m$ is directly connected to another match on either side, that is, $c(m) = (m, e_0, \ldots, e_n)$ with $e_0, e_n \in E_M$. The reason is that the degeneration of the constraint edges of $\mathbf{p}$ and $\mathbf{q}$'s polygons is already handled in the local pruning phase. Further, we ignore cycles containing more than one constraint edge, for reasons explained further shortly. Thus, we look for cycles $c(m) = (m, e_0^{k_0}, e_1, e_2^{k_2})$, with $e_1 \in E_C$ and $e_i^{k_i} = (e_{i,1}, \ldots, e_{i,k_i}) \in E_M^{k_i}$ ($e_i^{k_i}$ abbreviated as $e_i$ for $k_i = 1$), $k_i \geq 1$ (see Figure 7 top).

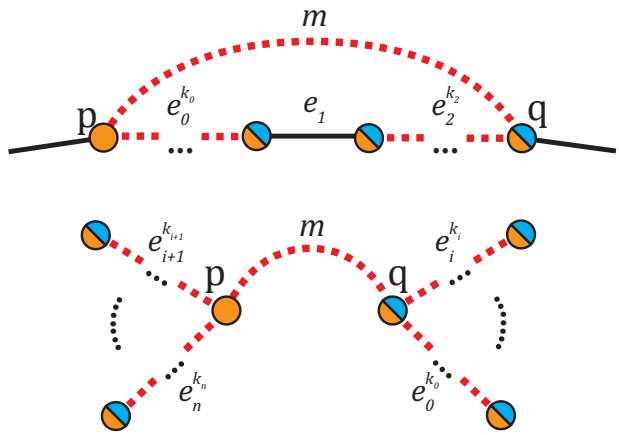


Fig. 7. Illustration of our global pruning idea: We search for each vertex-vertex/edge match $m = (\mathbf{p}, \mathbf{q})$ an edge cycle with only one constraint edge (denoted by $e_1$) in the extended matching graph $G_e$ (top). If such a cycle exists, we prune $m$ (refer to Figures 4 and 5). Otherwise, we search for match-paths $(e_l^{k_l}, m, e_j^{k_j})$ in the actual matching graph $G$ (bottom). Subsequent matches in the paths induce further matches (refer to Figure 8), which are then used to geometrically verify whether $m$ leads to polygon degenerations.

If we find such a cycle, the corresponding match can be pruned directly, because the cycle forces the constraint edge $e_1$ to contract (see Figures 4 and 5). However, in some rare cases (where polygons' elements meet at nonmanifold vertices/edges of the final model) we also observed the pruning of a few "correct" matches. For the convergence of $n$ polygon elements to the same location, $\binom{n}{2}$ connections (matches) are possible, but only $n-1$ involving all those $n$ elements are sufficient. Thus in practice, the pruning of a few "correct" matches doesn't indicate a problem.

Most of the degenerations in the model can already be avoided by pruning cycles with one constraint edge. However, there are also some cases involving several constraint edges; see Figure 8 right. Unfortunately, this situation cannot be detected unambiguously by searching for cycles, as can be seen in Figure 8 left. To solve this problem, we present a more general approach that is based on

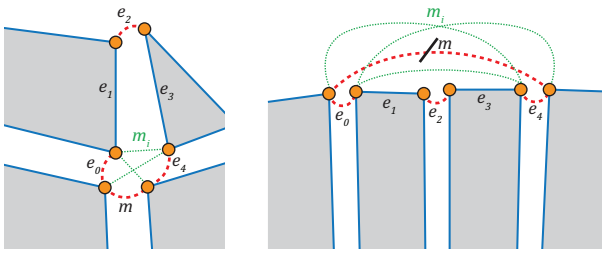


Fig. 8. Cycles (here $(m, e_0, e_1, e_2, e_3, e_4)$) in the extended matching graphs $G_e$ containing more than one constraint edge can be "harmful" (right) or not (left), and thus are not a good indicator for pruning. Instead we geometrically verify whether the induced matches (denoted in green by $m_i$) resulting from subsequent matches $(e_0, m, e_4)$ in the matching graphs $G$ cause polygon degenerations. Please note that in the right image, $m$ is selected in the matching phase due to large search radii of the corresponding vertices (we show only a part of the polygons here).

investigating sequences of matches. Such sequences *induce* further matches between the elements they connect, in the sense that in the optimization phase, these elements will also be joined. We thus need to verify whether the induced matches do not cause polygon degenerations.

Formally, for a match $m$, we search for paths $(e_l^{k_l}, m, e_j^{k_j})$ (with $k_l, k_j \geq 0$) in the actual matching graph $G$ (see Figure 7 bottom). We then check whether all of the induced matches $m_i$ are in $E_M$ as well. For every match that is not in $E_M$, we need to verify geometrically whether it would lead to a polygon degeneration. Here we note that an induced match does not necessarily join the attached elements directly (e.g., two subsequent vertex-edge matches $m_1 = (\mathbf{p}, \mathbf{e})$ and $m_2 = (\mathbf{e}, \mathbf{q})$ do not induce the vertex-vertex match $m_3 = (\mathbf{p}, \mathbf{q})$, but both vertices project to the same edge). To geometrically verify whether an induced match leads to any degeneration, we project the vertices and/or edge endpoints of the match onto the common intersection line $l$ of the polygons' supporting planes. We prune $m$ if one of the thus-modified polygons (with projected vertices/edges) has a flipped normal vector orientation (flip-over) or has self-intersections. Note that the number of paths to investigate is typically low because paths containing only matches stay localized.

## 5.2 Optimization

Based on the discovered adjacencies, we transform the polygons to optimally align with each other, while preserving their planarity and fitting to the input point cloud.

As in Kilian et al. [2008], we introduce a Cartesian coordinate system in the plane of each $P \in \mathcal{P}$, with origin $\mathbf{o}$ and basis vectors $\mathbf{f}_1$ and $\mathbf{f}_2$, and represent a point $\mathbf{p} \in P$ by the coordinates $(p_x, p_y)$, so that $\mathbf{p} = \mathbf{o} + p_x\mathbf{f}_1 + p_y\mathbf{f}_2$. During the optimization, in order to reduce the spatial gaps between adjacent polygons, the coordinates $(p_x, p_y)$ are displaced, while the Cartesian coordinate systems undergo a spatial motion. We linearize the spatial motion of each coordinate system by representing the displacement of each point through the velocity vector field of an instantaneous motion, given by $\mathbf{v}(\mathbf{x}) = \bar{\mathbf{c}} + \mathbf{c} \times \mathbf{x}$. Thus the position of a vertex $\mathbf{p} \in P_i$ during the optimization can be written as

$$\mathbf{p} = \mathbf{o}_i + \bar{\mathbf{c}}_i + \mathbf{c}_i \times \mathbf{o}_i + p_x(\mathbf{f}_{i_1} + \mathbf{c}_i \times \mathbf{f}_{i_1}) + p_y(\mathbf{f}_{i_2} + \mathbf{c}_i \times \mathbf{f}_{i_2}) \quad (4)$$

in the unknown parameters $\mathbf{c}_i, \bar{\mathbf{c}}_i \in \mathbb{R}^3$ of the velocity vector field attached to $P_i$ and in the unknown coordinates $(p_x, p_y)$ (this can be derived by applying the displacement $x' = x + v(x)$ for $x \in \{\mathbf{o}_i, \mathbf{o}_i + \mathbf{f}_{i_1}, \mathbf{o}_i + \mathbf{f}_{i_2}\}$).

5.2.1 *Snapping.* With the adjacency relations discovered by the prior step, we measure the *snapping error* as

$$E_{snap} = \sum_{i,j,k,l} d^2(\mathbf{p}_i, \mathbf{p}_j) + d^2(\mathbf{p}_i, \mathbf{e}_k) + d^2(\mathbf{p}_i, P_l),$$

where $d^2(\mathbf{p}_i, \cdot)$ denotes the distance of vertex $\mathbf{p}_i$ to the vertex $\mathbf{p}_j$, edge $\mathbf{e}_k$, and face $P_l$, respectively.

5.2.2 *Point Cloud Deviation.* For the polygon soup $\mathcal{P}$ not to deviate too much from the input point cloud, we use the *reference term*

$$E_{ref} = \sum_{l=1}^{|\mathcal{P}|} \sum_{i=1}^{|P_l|} d^2(\mathbf{p}_{i(l)}, P_l^{init}). \quad (5)$$

The preceding equation minimizes the sum of squared distances of vertices $\mathbf{p}_{i(l)} \in P_l$ to the initial planes $P_l^{init}$ (see Section 4.1.1). Manually sketched polygons without underlying segments (Section 6.3) are excluded from Eq. (5).

5.2.3 *Orthogonality.* In order to meet orthogonality constraints that naturally exist in urban environments, we include the following two terms

$$E_{\perp_1} = \sum_{i,j} w_{ij}(\mathbf{n}_i \cdot \mathbf{n}_j)^2 \quad (6)$$

and

$$E_{\perp_2} = \sum_{l=1}^{|\mathcal{P}|} \sum_{i=1}^{|P_l|} w_{i(l)}(\mathbf{e}_{i(l)} \cdot \mathbf{e}_{(i+1)(l) \bmod |P_l|})^2, \quad (7)$$

which measure the orthogonality of adjacent polygons and successive polygon edges, respectively. With the unit normal vector of $P$ given by $\mathbf{n} = \mathbf{f}_1 \times \mathbf{f}_2$, Eq. (6) extends over all pairs of polygons, with $w_{ij} = 1$ for adjacent polygons with normals deviating from orthogonality by less than $\frac{\pi}{9}$, and zero otherwise. Optimizing for orthogonality of polygon boundary edges $\mathbf{e}_{i(l)}$ in Eq. (7) (with $w_{i(l)}$ defined similar to $w_{ij}$ for polygons) might result in degenerating edges of vanishing length. This problem is in particular evident in case of missing geometry and is overcome by minimizing the sum of squared distances to current vertex positions $\mathbf{p}_i'$ as follows.

$$E_{cur} = \sum_i d^2(\mathbf{p}_i, \mathbf{p}_i')$$

5.2.4 *Global Energy and Weights.* The previous energy terms are combined into the objective function

$$E = \lambda_{snap}E_{snap} + \lambda_{ref}E_{ref} + \lambda_{\perp}(E_{\perp_1} + E_{\perp_2}) + \lambda_{cur}E_{cur},$$

which is minimized using a Gauss-Newton method. Instead of decoupling the optimization as in Kilian et al. [2008], we solve simultaneously for the parameters of the velocity vector fields attached to the polygons and the 2D coordinates of the vertices, resulting in a nonlinear optimization problem due to the products $p_x\mathbf{c}_i$ and $p_y\mathbf{c}_i$ (Eq. (4)). Transforming the Cartesian coordinate system of $P_i$ corresponding to the pair $(\mathbf{c}_i, \bar{\mathbf{c}}_i)$ would not yield a rigid body motion, but an affine one. Therefore we use the underlying helical motion, which ensures rigidity, as described by Pottmann et al. [2006]. The weights $\lambda$ allow additional control of the optimization. We used $\lambda_{snap} = 1$, $\lambda_{ref} = 0.5$, $\lambda_{\perp} = 0.01$ and $\lambda_{cur} = 0.1$ for all the models shown in the article.

# 6. INTERACTIVE 2D MODELING

On top of automatic polygon creation and polygon soup snapping, we propose an interactive editing and modeling system that provides a novel way of user-guided 3D content creation and reconstruction.

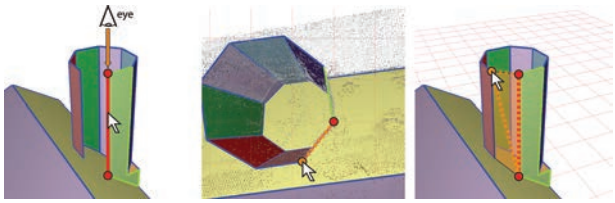


Fig. 9. Sketching of a polygon in an area without an underlying segment: An edge of an adjacent polygon is selected by moving the mouse over it (left). The camera view direction aligns with the edge, and a perpendicular sketching plane is used to sketch a point (middle). The new plane is defined by the edge vertices and the sketched point, and initialized with a rectangular polygon (right).

All user interaction is reduced to sketch-like approximate 2D operations by automatically choosing an appropriate 2D modeling space based on segment planes in the underlying point cloud. Implicitly dropping one dimension drastically reduces interaction complexity and thereby reduces overall modeling effort. At the same time, consistency and accuracy of the reconstructed model increase due to the interactive optimization performed after each modeling step (please see the accompanying video).

## 6.1 Plane Selection

All modeling operations are based on and limited to planes. The active plane is chosen by selecting a polygon or a point cloud segment with a single mouse click. On demand, the camera's view direction aligns to the plane normal, allowing a "2D top-down view" onto it.

## 6.2 Polygon Editing

Polygons, which have either been created automatically or sketched by the user (see the following), can be modified arbitrarily. Once in focus, the editing steps are comparable to a simple 2D vector graphics editing program: By moving the mouse cursor over the corresponding region, a vertex, an edge, or the whole polygon is chosen for manipulation and can be dragged anywhere on the underlying plane. Vertices can be added by right-clicking on an edge, and removed by right-clicking on a vertex.

Individual polygons lying on the same segment plane, which may occur due to holes in the point data, can easily be merged by dragging polygons over each other, resulting in a single polygon consisting of their combined convex hull. We opted for this fast interactive method of solving such cases instead of closing the holes automatically, as there are various situations in which such individual coplanar polygons are intended (e.g., front faces of balconies on a facade).

## 6.3 Polygon Sketching

In sketching mode, the selected plane also acts as a drawing area for new polygons. Manual sketching is applied whenever a segment has no polygons assigned (too few points), or if the existing polygon has been estimated wrongly due to noisy and missing data. In some cases, it can even be faster to replace the polygon by a new one instead of repairing it. Sketching is performed by approximately clicking the new vertex positions on the sketch plane. Existing polygons overlapping the new one are automatically removed.

Since we are dealing with noisy data, sparsely sampled parts of the model will most likely not be found in the RANSAC stage, excluding both automatic polygonalization as well as plane-based sketching in these areas. We therefore implemented an intuitive way to model arbitrary planes adjacent to existing polygons (see Figure 9): The user chooses an edge **e** of a polygon on which the new plane should attach to by hovering over it with the mouse cursor. By entering the sketching mode, the camera view direction aligns with the edge (i.e., **e** is only visible as a point then), and the user selects a point **p** on the plane perpendicular to the selected edge. The new plane is built using **p** and the vertices of **e**, and initialized with a rectangular polygon.

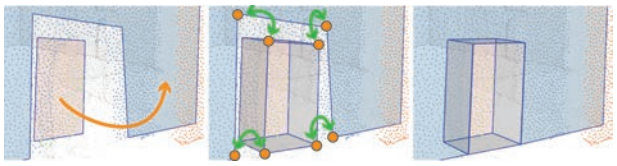

Fig. 10. A hierarchy relation between a dominant facade plane and a door is defined with a single click (left). Side faces are automatically extruded (middle) and contribute to the snapping process: Due to the newly found matching pairs, a continuous surface is in this case generated within a few optimization iterations (right).

## 6.4 Interactive Optimization

It is important to note that both editing and sketching operations only have to be performed very coarsely. As long as the approximate shape of the polygon is given, automatic snapping will align vertices with other parts of the model by favoring right angles while simultaneously refitting the polygon soup to the underlying point cloud. Optimization is therefore interleaved with each individual modeling step, providing the user with immediate feedback. When sketching completely new parts of a model, interactive optimization can also be switched off on-demand.

## 6.5 Hierarchies

The noisier and more sparsely sampled the faces to reconstruct are, the less likely it is that a suitable plane to sketch on can be found in the RANSAC stage. Depending on the chosen acquisition angle and technique, some faces may not even be depicted in the point cloud data at all. Despite the possibility to easily define arbitrary planes as described earlier, modeling the side faces of features like balconies, bays, or windows remains a time-consuming and tedious task. We therefore allow the definition of hierarchy relations between the polygons by "connecting" a child polygon to a parent polygon with a single click: The corresponding side faces are then implicitly generated by extruding the child polygon's edges to its parent plane, reducing the modeling time to a few seconds (the corresponding holes in the parent polygons are created using a simple difference set operation $P_{Parent} \setminus P_{Child}$ ). This approach was used for example in Figure 2 right.

The generated side faces also contribute to the interactive optimization, which proves extremely useful in cases like doors situated at the bottom of a parent facade plane (see Figure 10): the vertices of the parent polygon lying in the front do not have to be manually edited, but are automatically attached to the side faces, generating a continuous surface with a single click.

Our hierarchy definition is extremely useful to model thin features (see Figure 11), as child polygons (belonging to different parent polygons) may snap to each other.

## 6.6 Manual Segment Division

As discussed in Section 4.1.1 and shown in Figure 2 (left), nearby parallel structures may be captured as a single segment. Therefore

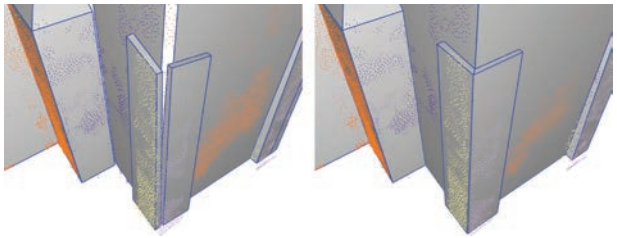

Fig. 11. The snapping is not restricted to the reconstruction of the main structure and to child-parent relations, but it can also handle child-child relations to reconstruct detailed geometry: Two child polygons and their parent polygons after the manual segment division and subsequent polygon extraction (left). The child polygons snap to each other and the corresponding side polygons are projected correctly to the main structure to form the column (right).

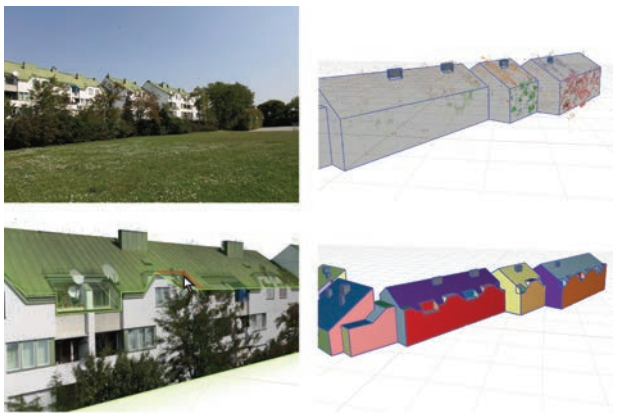

Fig. 12. Reconstruction of a building complex occluded by trees and bushes. Top left: An example photo of the dataset. Top right: The segments extracted from the sparse point cloud (obtained from a photogrammetric approach) and the main structures after approximately three minutes of optimization-aided modeling. Bottom left: The reprojected images help the user to modify the polygon boundaries and to sketch new polygons forming the balconies. Bottom right: The final model after 20 minutes using the additional image information.

we offer the user the opportunity to manually divide (Figure 2 middle right,) a selected segment by applying *k-means clustering* ($k = 2$). For the emerging segments, new polygons are automatically created (see Section 4, Figures 2 middle right, and right). By combining the manual segment division with the hierarchy definition, complete facades including windows, balconies, and doors can be modeled within seconds (see Figures 1, 10, and 11).

## 7. RESULTS

We have tested our reconstruction and modeling pipeline on a variety of datasets, including six point clouds obtained from photogrammetric methods (Figures 12 and 18), a laser scan (Figure 15), and a synthetic model (Figure 16). Figure 18 demonstrates the individual steps of our framework. Figures 12, 14, and 13 illustrate several applications. Figure 15 compares a model created using our system to the results of mesh reconstruction and decimation algorithms. The synthetic model, Figure 16, is used to validate the accuracy of our algorithm. Interactive modeling sessions for the town hall and church models can be observed in the accompanying video.

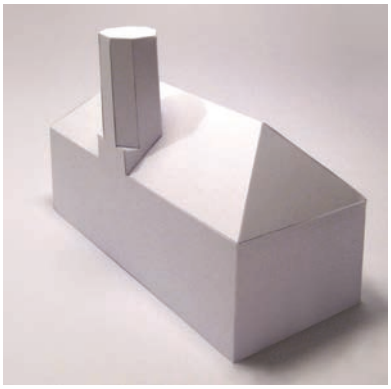

Fig. 13. A paper model of *Town Hall*.

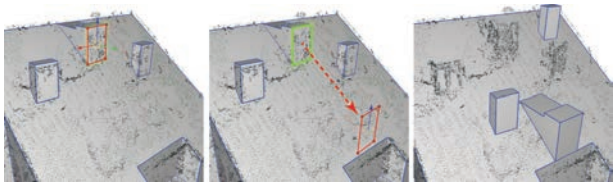

Fig. 14. By design, the reconstructed models offer the generation of shape variations by exploiting the underlying adjacency graph.

### 7.1 Performance and Scalability

In all our test scenes, interactivity could easily be maintained during the modeling sessions on a standard PC workstation (Intel i7 920 CPU with 2.67 GHz, 4GB RAM): The computationally most expensive step after a modeling operation, matching and pruning, is computed within an average time of 0.2 seconds, while the optimization and the update of the rendering scene graph only take a few milliseconds to perform. We solve the sparse systems of linear equations at each Gauss-Newton iteration by a sparse QR factorization [Davis 2011].

Note that in our current implementation, the adjacency (matching) graph is completely rebuilt from scratch after each modeling step. In our test scenes, we experienced an adjacency graph rebuild time of one second as the worst case. By building the graph only once and updating it locally after each modification, the computational effort for the graph update could be decoupled from the geometric complexity, removing this potential bottleneck and keeping the modeling process interactive in larger-scale scenes.

### 7.2 Convergence

Solving the problem presented in Section 5 is a challenging task since we deal with an optimization problem that is:

(1) nonsmooth: the set of computed adjacencies is a discrete variable,

(2) nonlinear: due to the simultaneous optimization of the parameters (refer to Section 5.2.4), and

(3) constrained: the polygons shall remain planar.

To account for (1), we decouple the computation of the adjacencies from the rest of the optimization: at each iteration, we first fix the position of the polygons' vertices and search for adjacencies. Then, we fix the adjacencies and optimize vertex positions.

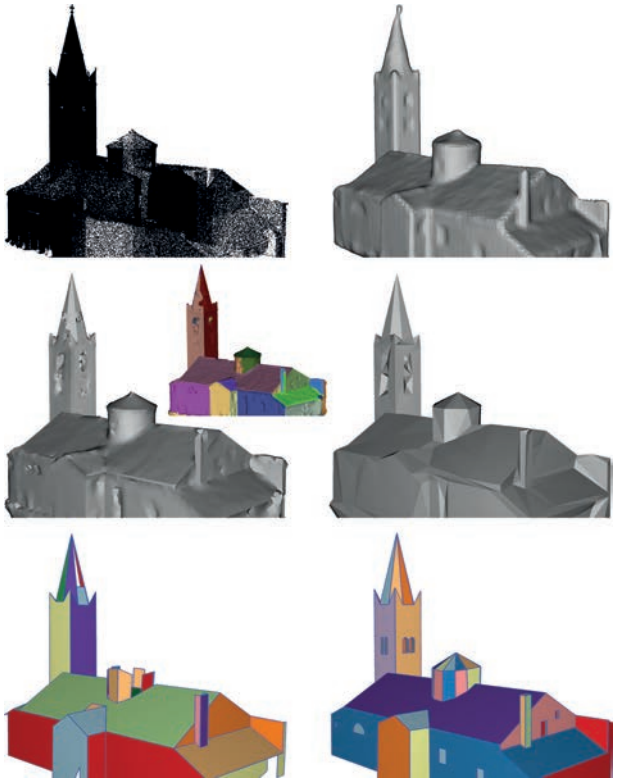

Fig. 15. Top left: Input point cloud (provided courtesy of INPG by the AIM@SHAPE Shape Repository) downsampled to 10% of the original dataset. Top right: Meshed Poisson implicit function [Kazhdan et al. 2006] (41k triangles). Middle left: Feature-preserving mesh [Salman et al. 2010] provided courtesy of the authors (18k triangles). Middle left, small: Segmentation of the latter mesh [Cohen-Steiner et al. 2004]. Middle right: Mesh obtained by applying quadric-based simplification to the segments [Garland and Heckbert 1997] (1k triangles). Bottom row: Initial automatic reconstruction of our method (43 polygons, 289 triangles used for rendering) and final refined model after additional 15 minutes of optimization-aided modeling (174 polygons, 109 of them for the windows, 799 triangles used for rendering).

To account for (2), we choose a Gauss-Newton method (which approximates the distance function) for the smooth optimization and provide a good initialization of the problem (Section 4), which is known to be necessary in the solution of nonlinear optimization problems (see, e.g., Kelley [1999]).

To account for (3), we attach a Cartesian coordinate frame to each polygon and linearize its motion (which is again an approximation).

Due to the underlying characteristics of the given optimization problem, there is no guarantee that a global minimum can be found in an acceptable time. However, the results shown in this section indicate that we manage to find an aesthetically pleasing and accurate solution in a few iterations.

## 7.3 Further Applications

7.3.1 *Photo-Guided Modeling.* During our tests with a wide range of different datasets, we have made the observation that in some cases parts of the model to reconstruct are not only sparsely sampled, but are not depicted in the point cloud data at all. This may be caused by occluders (e.g., lots of trees and bushes), highly

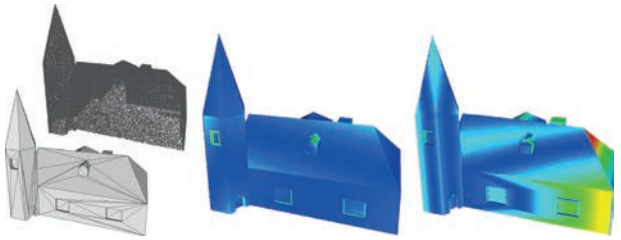

Fig. 16. Point cloud with artificial noise (top left) sampled from a synthetic model (bottom left). Results from O-Snap (middle) and Pointools (right), colored according to approximation error, with blue meaning zero Hausdorff distance, and red high distance, respectively. Note that with O-Snap, the overall building structure is recreated very accurately. Errors mainly appear at sparsely sampled child elements (e.g., windows), especially at their side faces, which currently do not consider the point cloud at all.

reflective materials (e.g., glassy or metallic facades, which lead to problems for both laser scanners and photogrammetric approaches) or the viewing angle from which the building has been captured. Our modeling tools have still proven capable of reconstructing such areas, if the user is provided with photos of the object; in case of photogrammetric data, these can even be reprojected onto the existing geometry. The example in Figure 12 shows a complex of connected buildings that are highly occluded by trees, resulting in a noisy and sparse point cloud in which details like the balconies are not present. The image information reprojected on the basic shapes helps the user to modify the boundaries accordingly, and lets him or her accurately add any missing polygons as explained in Section 6.3. Please note that no reprojection of images was applied during the modeling process of the objects displayed in Figure 18.

7.3.2 *Manufacturing.* Precise reconstructions with low face count are of interest to applications beyond architecture, in particular to manufacturing. Simple production patterns are valuable, for example, for upfolding planar cut patterns from paper or sheet metal. We shortly outline here how to implement the reverse operation to upfolding in our pipeline to generate production data. Unfolding a polyhedron to a planar, connected shape without any self-intersections by only cutting along edges is a well surveyed research area (refer to Demaine and O'Rourke [2007]). Interestingly, it is still unknown if any convex polyhedron allows such an edge unfolding, whereas it is known that there exist nonconvex polyhedra where this is not possible. Basically, the solution space for a given mesh is given by all spanning trees of the mesh's face dual. By relaxing the constraints and requesting not a single but a small number of connected components, we determine a feasible spanning tree by heuristically searching the solution space. An example of a folded paper version of the *town hall* model is shown in Figure 13.

7.3.3 *Advanced Editing.* Besides the modeling features introduced in Section 6, our models offer (by using the underlying adjacency graph) further editing possibilities to create different looks of reconstructed shapes: The user selects a single face of a chimney and applies an affine transformation to it. The connected component of the adjacency graph (containing the chimney's transformed face) undergoes the same affine transformation and the optimization is applied to reestablish a closed model (see Figure 14). While our main task is still reconstruction from point clouds, our pipeline leads to interesting ways for shape manipulation. Although having different objectives, the idea of optimization coupled editing

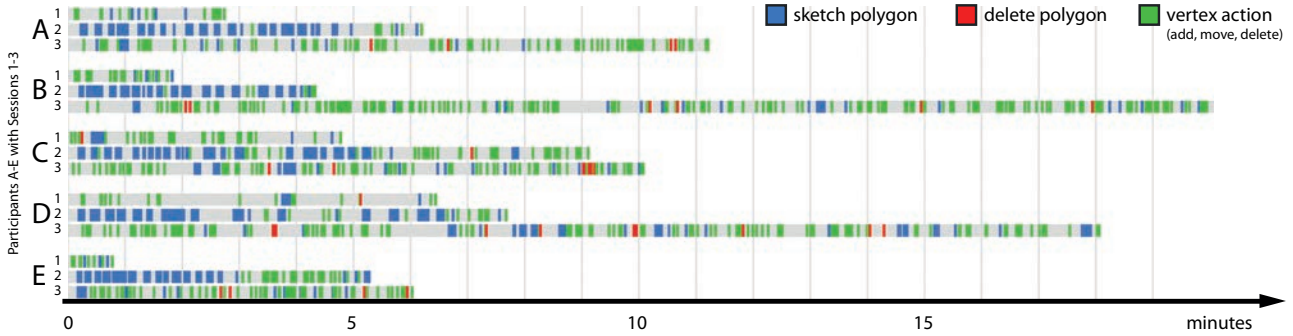


Fig. 17. All modeling sessions of our informal user study. Each of the 15 horizontal bars represents the timeline of a single session. Each user interaction has been logged and time-stamped. Different colors represent different kinds of interaction. Session 1 shows mostly vertex actions (move, add, delete) as results from the automatic pipeline are fine-tuned. Session 2 consists mostly of polygon sketching because no initial model is available. Session 3 is based on a complicated church dataset. This takes more time, but all users are able to create a clean model. Participants: A has basic computer graphics skills. B and E are researchers in real-time rendering. C is a skilled user of commercial modeling software. D has absolutely no computer science and graphics background.

has been extensively studied in the shape manipulation framework introduced by Gal et al. [2009].

## 7.4 Evaluation

To evaluate our method, we compared the visual quality of the models generated using our system and various other mesh reconstruction and decimation algorithms, performed a test to validate the accuracy of our results compared to using an existing interactive tool, and conducted a user study with nonexpert users to show the ease of use of our method.

7.4.1 *Comparison with Meshing Methods.* To evaluate the visual quality of our reconstructions, we applied our method and various other meshing techniques to the laser scan of the *Church of Lans le Villard* (Figure 15 top left). Figure 15 compares the different approaches: MeshLab's [2008] implementation of *Poisson Surface Reconstruction* [Kazhdan et al. 2006] (top right), Salman et al.'s [2010] feature-preserving mesh generation (middle left), the latter method followed by Graphite's [2010] implementation of geometry segmentation [Cohen-Steiner et al. 2004] (middle left, small image), and the same model with quadric-based mesh decimation [Garland and Heckbert 1997] applied to each segment with fixed boundary, for which we used MeshLab [2008] again (middle right). The third row shows the results of our reconstruction and modeling pipeline.

7.4.2 *Accuracy Comparison.* We used our system and a commercial point-based modeling tool, the Pointools [2011] plugin for SketchUp, on a point cloud sampled from a synthetic house model. Noise was added to sample positions in the amount of 0.5% of the bounding box diagonal. Modeling was performed by a skilled artist, who was instructed to create the most accurate model possible in the two tools, both of which he had used before. There were no time constraints. After completion the artist reported to be confident having created perfectly accurate models in both tools, but also that he needed considerably more time and patience for modeling in Pointools. In order to quantify this feedback, we compared Hausdorff distances for each result to the original synthetic model (measured using the Metro tool [Cignoni et al. 1996]), and modeling times (see Table I). The results indicate that our method outperforms the commercial tool in terms of accuracy and modeling time. In addition to the Table I, see also Figure 16 for a visualization of the approximation error.

Table I. Comparison of Modeling Times and Approximation Errors (color coded in Figure 16) Relative to the Model's Bounding Box Diagonal

| Tool | Duration | Min | Mean | Max |
|---|---|---|---|---|
| O-Snap | 6.2 minutes | 0 | 0.000588 | 0.007794 |
| Pointools | 35.7 minutes | 0 | 0.001433 | 0.009382 |

7.4.3 *User Study.* In order to verify our tool's general usability for nonexpert users, we performed an informal user study. All participants had never used the tool before and received the same ten minutes hands-on introduction. Each user had to complete three separate O-Snap sessions. Since not all candidates had a computer graphics or modeling background, we only gave the general directive to create a *good-looking model*. We stopped each session as soon as the user reached a closed model without major deficiencies. A time-stamped log file of all user interactions was generated for each session (see Figure 17).

| Session | Data Set | Initial Reconstruction | Duration *Ø, (min-max)* |
|---|---|---|---|
| 1 | Town Hall | available | 3.7 minutes (1-6) |
| 2 | Town Hall | no | 6.8 minutes (5-9) |
| 3 | Old Church | available | 13.1 minutes (6-20) |

*Session 1* is based on the simple town hall model shown in Figure 18 (top row). Here the initial reconstruction (RANSAC, Sections 4 and 5) is almost perfect, and users only have to add a missing back wall and fine-tune some faces. An average user solves this task in three minutes. *Session 2* is equivalent to 1, but *without* the polygonalization step (Section 4). Users have to manually sketch all the polygons on the underlying segment planes (Section 4.1.1), and are aided by interactive optimization (Section 5), which takes seven minutes on average. We conclude that fine-tuning our automatic pipeline results is about twice as efficient as sketching all the polygons from scratch. On the other hand, the interactive optimization allows a novice user to create a model from scratch in still acceptable time. Finally, *Session 3* challenges users with a complex church model, shown in Figure 18 (second row). The initial model contains misaligned faces and some parts are missing, but all users were able to deal with the high geometric complexity and successfully create a closed model in only 13 minutes on average.

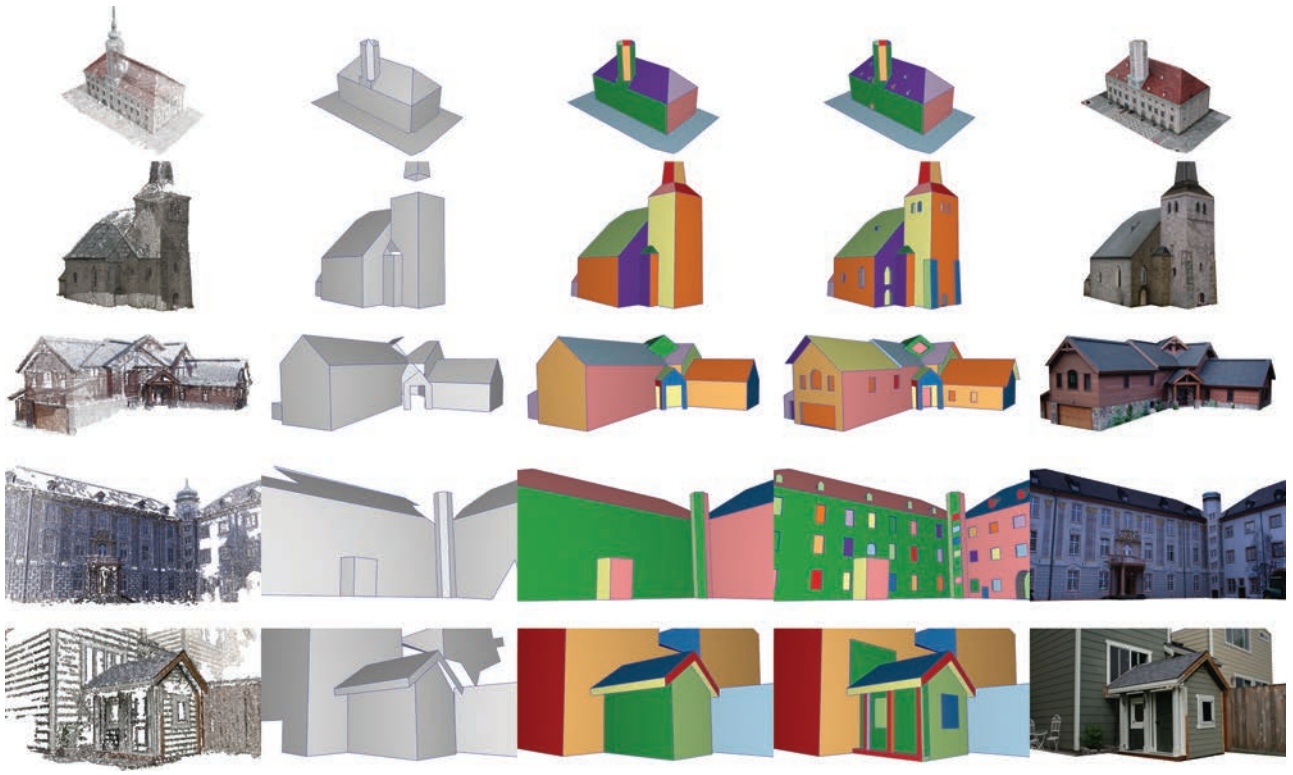

Fig. 18. Results from five point cloud data sets (generated from photos) shown from top to bottom: town hall, old church (Photos courtesy of Rainer Brechtken), mountain house (Photos courtesy of Sinha et al. [2008]), castle-P19 (Photos courtesy of Strecha et al. [2008]) and playhouse (Photos courtesy of Sinha et al. [2008]). Left to right: input point cloud, initial automatic reconstruction, refined model, final model with advanced details added, final textured model (with the photos simply back-projected onto the model, more sophisticated methods for seamless texturing exist, e.g., Sinha et al.'s graph-cut optimization [2008]).

*General observations.* Users who spend more than average time consistently try to model ever smaller details or extrapolate building parts not contained in the original data. All users are able to quickly recover from erroneous modeling actions using *undo* or by deleting incorrect shapes. No user was ever genuinely lost or stuck. An unexpected but useful observation is that users who manage to accidentally sketch a polygon inside a wrong plane consistently try to flip this polygon over to the correct plane. Currently such an operation is not supported, but it seems to be expected intuitively, which is an inspiration for future work.

Our main conclusion is that nonexpert users are perfectly capable of understanding and applying O-Snap's modeling tools after only ten minutes of basic introduction. Even participants without any prior CG and modeling experience are able to create shapes of buildings, aligned with the underlying point cloud data.

*Comparison with Pointools.* We compared the effectiveness of our system to an existing commercial tool by instructing the user study participant with the most experience in 3D modeling to create models of *town hall* and *old church* using Pointools (please see the accompanying video). A time limit of 30 minutes per scene was stipulated. The candidate was already familiar with the tool. While he succeeded in modeling the town hall in about 12 minutes (as compared to 5 minutes using O-Snap, see Figure 17, C), he had severe problems handling the more complex church model. Choosing appropriate points in the point cloud to construct the building's outline as well as to draw faces on top of the extruded outline consumed much of the available time. At the end of the 30 minutes time limit, he ended up with an incomplete model. Using O-Snap he was able to successfully complete the same task in about 10 minutes.

We conclude that a Pointools-like approach is very well suited for quickly creating simple axis-aligned models, but becomes tedious for more complex or incomplete (real-world) datasets. The recommended approach of ground plane-based extrusion of building outlines results in additional effort and inaccuracies, since many architectural elements cannot be captured by simple extrusion and have to be fixed manually.

In contrast, O-Snap's concept of sketching in 2D (on planes automatically fitted to the underlying point cloud) is more in line with an artist's workflow. It also strongly supports the comprehension of complex datasets by removing the effort required to extract geometric meaning from raw point data.

## 8. CONCLUSION

We presented an interactive 3D modeling system that leverages techniques from mathematical optimization to provide a novel way of user-guided reconstruction and modeling of architectural scenes. The system first proposes an initial automatic reconstruction from an unstructured point cloud, by extracting candidate planes and estimating coarse polygons on these planes. Local feasible

adjacency relations between polygons are automatically computed and enforced to align (snap) different parts of the model, while maintaining a fit to the input data. Besides these local relations, the system also favors orthogonality. In an interactive modeling phase, the model can be refined using coarse strokes on 2D planes. After each step, snapping reestablishes a watertight model where feasible.

## 8.1 Limitations and Future Work

Since many man-made objects, and especially buildings, consist of planar surfaces, we solely used planes in our reconstruction and modeling pipeline. In practice, this already allows handling a wide range of architectural styles by approximating curved surfaces in the scene by planes (see the cylindrical and conical parts of the model approximated by a number of planes in Figure 15 bottom right). However, our method is not limited to planes and can support other shapes by extending our matching definition and defining appropriate modeling spaces, which is an inspiration for future work.

In order to keep our matching and pruning definition simple, we allowed a few restrictions: (1) skew edges aren't matched, and (2) vertex-face matches aren't pruned. The edge-edge matching definition and the extended matching graph can be revised to overcome these restrictions, but we didn't observe any cases in our datasets where this would be necessary.

The robustness of the boundary extraction algorithm can suffer from nonuniformly sampled segments. Even though we have not observed this known limitation of alpha shapes in our datasets, one could use *conformal alpha shapes* [Cazals et al. 2005], which employ a local scale parameter (instead of the global $\alpha$) to reconstruct nonuniformly sampled surfaces.

Currently, our system does not investigate repetitive structures (e.g., balconies, windows on a facade), but these structures can efficiently be modeled through our optimization-aided hierarchy operation. However, we plan to extend our system to analyze regular structures as proposed by Pauly et al. [2008].

Our novel modeling system differs significantly from other 3D modeling techniques through its interactive coupling with the optimization routines. Currently, we are performing more extensive user studies to learn more about expectations of O-Snap's users.

## ACKNOWLEDGMENTS

We wish to express our thanks to the reviewers for their insightful comments, and to the user study participants for their efforts and valuable feedback. We also thank Mariette Yvinec for providing us with the output mesh of their algorithm for the Church of Lans le Villard, and Evelyn Sutterlüti for the upfolding of the paper model of town hall. For providing photogrammetric datasets (from which we generated the datasets used in the article), we thank Sudipta Sinha (mountain house, playhouse), Rainer Brechtken (old church) and Strecha et al. [2008] (castle-P19). We also acknowledge the Aim@Shape Shape Repository for the laser scan of the Church of Lans le Villard.

## REFERENCES

ALLIEZ, P., COHEN-STEINER, D., TONG, Y., AND DESBRUN, M. 2007. Voronoi-Based variational reconstruction of unoriented point sets. In *Proceedings of the 5th Eurographics Symposium on Geometry Processing*. Eurographics Association, 39–48.

ATKINSON, A. C. AND RIANI, M. 2000. *Robust Diagnostic Regression Analysis*. Springer.

AVRON, H., SHARF, A., GREIF, C., AND COHEN-OR, D. 2010. $\ell_1$-sparse reconstruction of sharp point set surfaces. *ACM Trans. Graph. 29*, 135:1–135:12.

BOISSONNAT, J.-D. AND OUDOT, S. 2005. Provably good sampling and meshing of surfaces. *Graph. Models 67*, 405–451.

BOTSCH, M., PAULY, M., GROSS, M., AND KOBBELT, L. 2006. Primo: coupled prisms for intuitive surface modeling. In *Proceedings of the 4th Eurographics Symposium on Geometry Processing*. 11–20.

CAZALS, F., GIESEN, J., PAULY, M., AND ZOMORODIAN, A. 2005. Conformal alpha shapes. In *Proceedings of the Eurographics/IEEE VGTC Symposium on Point-Based Graphics 0*, 55–61.

CHEN, J. AND CHEN, B. 2008. Architectural modeling from sparsely scanned range data. *Int. J, Comput. Vis. 78,* 2–3, 223–236.

CIGNONI, P., ROCCHINI, C., AND SCOPIGNO, R. 1996. Metro: Measuring error on simplified surfaces. Tech. rep., Paris, France.

COHEN-STEINER, D., ALLIEZ, P., AND DESBRUN, M. 2004. Variational shape approximation. *ACM Trans. Graph. 23*, 905–914.

DAVIS, T. A. 2011. Algorithm 915, SuiteSparseQR: Multifrontal multithreaded rank-revealing sparse QR factorization. *ACM Trans. Math. Softw. 38,* 1.

DEBEVEC, P. E., TAYLOR, C. J., AND MALIK, J. 1996. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *Proceedings of SIGGRAPH 96*. 11–20.

DEMAINE, E. AND O'ROURKE, J. 2007. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press.

EDELSBRUNNER, H. AND MÜCKE, E. P. 1994. Three-Dimensional alpha shapes. *ACM Trans. Graph. 13*, 43–72.

FLEISHMAN, S., COHEN-OR, D., AND SILVA, C. T. 2005. Robust moving least-squares fitting with sharp features. *ACM Trans. Graph. 24*, 544–552.

FURUKAWA, Y., CURLESS, B., SEITZ, S. M., AND SZELISKI, R. 2009. Manhattan-world stereo. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1422–1429.

GAL, R., SORKINE, O., MITRA, N. J., AND COHEN-OR, D. 2009. iWIRES: An analyze-and-edit approach to shape manipulation. *ACM Trans. Graph. 28,* 3, #33, 1–10.

GARLAND, M. AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. 209–216.

GRAPHITE. 2010. http://alice.loria.fr/index.php/software.html.

JOLLIFFE, I. T. 2002. *Principal Component Analysis* 2nd Ed. Springer, New York.

KAZHDAN, M., BOLITHO, M., AND HOPPE, H. 2006. Poisson surface reconstruction. In *Proceedings of the 4th Eurographics Symposium on Geometry Processing*. Eurographics Association, 61–70.

KELLEY, C. T. 1999. *Iterative Methods for Optimization*. SIAM, Philadephia, PA.

KILIAN, M., FLÖRY, S., CHEN, Z., MITRA, N. J., SHEFFER, A., AND POTTMANN, H. 2008. Curved folding. *ACM Trans. Graph. 27,* 3, #75, 1–9.

KOBBELT, L. P., BOTSCH, M., SCHWANECKE, U., AND SEIDEL, H.-P. 2001. Feature sensitive surface extraction from volume data. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. ACM, New York, 57–66.

LI, Y., WU, X., CHRYSANTHOU, Y., SHARF, A., COHEN-OR, D., AND MITRA, N. J. 2011. GlobFit: Consistently fitting primitives by discovering global relations. *ACM Trans. Graph. 30,* 4, Article 52.

MESHLAB. 2008. MeshLab:/An open=source 3D mesh processing System. http://meshlab.sourceforge.net.

MUSIALSKI, P., WONKA, P., ALIAGA, D. G., WIMMER, M., VAN GOOL, L., AND PURGATHOFER, W. 2012. A survey of urban reconstruction.

In *EUROGRAPHICS 2012 State of the Art Reports*, Eurographics Association, 1–28.

NAN, L., SHARF, A., ZHANG, H., COHEN-OR, D., AND CHEN, B. 2010. SmartBoxes for interactive urban reconstruction. *ACM Trans. Graph. 29*, 93:1–93:10.

PAULY, M., MITRA, N. J., WALLNER, J., POTTMANN, H., AND GUIBAS, L. 2008. Discovering structural regularity in 3D geometry. *ACM Trans. Graph. 27,* 3, #43, 1–11.

POINTOOLS LTD. 2011. Pointools plugin for sketchup. http://www.pointools.com/pointools-plug-in-for-sketchup.php.

POTTMANN, H., HUANG, Q.-X., YANG, Y.-L., AND HU, S.-M. 2006. Geometry and convergence analysis of algorithms for registration of 3D shapes. *Int. J. Comput. Vis. 67*, 277–296.

ROUSSEEUW, P. J. AND LEROY, A. M. 1987. *Robust Regression and Outlier Detection*. John Wiley & Sons, New York.

SALMAN, N., YVINEC, M., AND MERIGOT, Q. 2010. Feature preserving mesh generation from 3D point clouds. *Comput. Graph. Forum 29,* 5, 1623–1632.

SCHINDLER, K. AND BAUER, J. 2003. A model-based method for building reconstruction. In *Proceedings of the 1st IEEE International Workshop on Higher-Level Knowledge in 3D Modeling and Motion Analysis*. 74–82.

SCHNABEL, R., DEGENER, P., AND KLEIN, R. 2009. Completion and reconstruction with primitive shapes. *Comput. Graph. Forum 28,* 2, 503–512.

SCHNABEL, R., WAHL, R., AND KLEIN, R. 2007. Efficient RANSAC for point-cloud shape detection. *Comput. Graph. Forum 26,* 2, 214–226.

SINHA, S. N., STEEDLY, D., SZELISKI, R., AGRAWALA, M., AND POLLEFEYS, M. 2008. Interactive 3D architectural modeling from unordered photo collections. *ACM Trans. Graph. 27*, 159:1–159:10.

STRECHA, C., VON HANSEN, W., GOOL, L. J. V., FUA, P., AND THOENNESSEN, U. 2008. On benchmarking camera calibration and multi-view stereo for high resolution imagery. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*.

VAN DEN HENGEL, A., DICK, A., THORMÄHLEN, T., WARD, B., AND TORR, P. H. S. 2007. VideoTrace: Rapid interactive scene modelling from video. *ACM Trans. Graph. 26*.

VANEGAS, C. A., ALIAGA, D. G., AND BENES, B. 2010. Building reconstruction using Manhattan-world grammars. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*. 358–365.

WERNER, T. AND ZISSERMAN, A. 2002. New techniques for automated architectural reconstruction from photographs. In *Proceedings of European Conference on Computer Vision (ECCV)*. 541–555.

Received November 2011; revised April 2012; accepted May 2012