Efficient Edge-Aware Surface Mesh Reconstruction for Urban Scenes

András Bódis-Szomorú[a], Hayko Riemeschneider[a], Luc Van Gool[a,b]
[a]Computer Vision Laboratory, ETH Zürich, Switzerland
[b]Vision for Industry Communications and Services (VISICS), KU Leuven, Belgium

Highlights

- Meshing approach for both street-side SfM data and large-scale urban height maps.
- Our meshes preserve crease edges and discontinuities without staircasing artifacts.
- 2D base mesh from superpixels or from piecewise-planar depth map partitioning.
- Fast linear vertex depth optimization including a curvature penalty term.
- Excellent trade-off between model compactness and approximation quality.

Appears in: Computer Vision and Image Understanding
Publisher: Elsevier
Submitted: 30 November 2015
Revised: 25 April 2016
Accepted: 9 June 2016

DOI: 10.1016/j.cviu.2016.06.002
Final paper at: http://authors.elsevier.com/sd/article/S1077314216300789

Efficient Edge-Aware Surface Mesh Reconstruction for Urban Scenes

András Bódis-Szomorú[a,*], Hayko Riemenschneider[a], Luc Van Gool[a,b]

[a] *Computer Vision Laboratory, ETH Zürich, Sternwartstrasse 7, CH-8092 Zürich, Switzerland*
[b] *Vision for Industry Communications and Services (VISICS), KU Leuven, Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium*

Abstract

We propose an efficient approach for building compact, edge-preserving, view-centric triangle meshes from either dense or sparse depth data, with a focus on modeling architecture in large-scale urban scenes. Our method constructs a 2D base mesh from a preliminary view partitioning, then lifts the base mesh into 3D in a fast vertex depth optimization. Different view partitioning schemes are proposed for imagery and dense depth maps. They guarantee that mesh edges are aligned with crease edges and discontinuities. In particular, we introduce an effective plane merging procedure with a global error guarantee in order to maximally compact the resulting models. Moreover, different strategies for detecting and handling discontinuities are presented. We demonstrate that our approach provides an excellent trade-off between quality and compactness, and is eligible for fast production of polyhedral building models from large-scale urban height maps, as well as, for direct meshing of sparse street-side Structure-from-Motion (SfM) data.

Keywords: 3D city model, Surface reconstruction, Meshing, Mesh simplification, Superpixel segmentation, Piecewise-planar, Digital Surface Model (DSM), Structure-from-Motion (SfM), Sparse point cloud.

1. Introduction

Automatic 3D modeling of cities is a long-standing challenge, and still receives a lot of attention [1, 2], fueled by applications in urban planning, navigation, entertainment, cultural heritage, real-estate advertising, etc.

The increasing availability of high-density 3D data collected by airborne sensors stimulated recent advances in city modeling at larger scales. Modern Multi-View Stereo (MVS) based on high-resolution airborne imagery has become a solid alternative to airborne LiDAR, not to speak of the additional color and texture information they provide [3]. Companies like Google, Apple and Bentley, have recently invested into automated solutions for generating large-scale textured city models from airborne data. Oblique aerial imagery can capture higher floors of facades, but lower floors and street-level structures are often occluded, especially in metropolitan areas, and strong shadows hamper the extraction of dense correspondences between the images.

Mobile mapping provides an alternative by acquiring high-resolution street-level data. However, the massive amount of imagery and 3D data collected at street-level in large-scale comes with a high toll on the required storage space, as well as the processing and visualization workload. Generic dense MVS is time-consuming when applied to such data, unless restrictive priors (like Manhattan-world[1], e.g. [4]) and GPUs are exploited [5, 6]. This motivates attempts to directly create lighter models from both the imagery and the underlying sparse Structure-from-Motion (SfM) point cloud, without a pixelwise MVS step [7, 8, 9, 10]. Moreover, modern facade analysis algorithms operate at the semantic level of windows, doors and walls [11, 12, 13, 14], which often proves sufficient to obtain realistic visualizations of real facades if the output is enhanced by standard computer graphics (e.g. realistic textures and glossy windows) [15].

In fact, products of both airborne and street-level mobile mapping acquisition, such as point clouds, depth maps, height maps (also known as Digital Surface Models, DSM) or dense meshes usually require further simplification and abstraction to obtain polyhedral models that are easier to handle and visualize.

We conclude that compactness remains a more valuable property than centimeter-accuracy details in large-scale modeling of buildings. Our goal is to cover large areas while capturing architectural aspects like windows being set back, balconies or larger blocks on the facades sticking out, as well as dominant dormers on the roofs and real-world urban roof shapes that are often more complex than the prototypical flat, gable, hip, mansard roofs etc. The Level-of-Detail (LoD) aimed at corresponds to LoD-3 in the CityGML 2.0 standard, where LoD-4 stands for added indoor details, LoD-2 for buildings represented with coarse roofs and flat facades and LoD-1 for models obtained by simple polygon extrusion (flat roofs, flat facades).

This paper focuses on the reconstruction of compact, view-centric surface meshes if either sparse or dense depth information is known. The proposed meshing approach is particularly promising for two complementary tasks in

*Corresponding author
[1] Assumes three mutually orthogonal surface orientations.

large-scale urban modeling (see Figure 1):

1. surface mesh reconstruction directly from an SfM point cloud and street-side imagery in order to overcome the computational bottleneck of pixel-wise 3D reconstruction,
2. compact and accurate mesh approximation of large-scale urban height maps (DSMs), primarily to capture roof and building shapes as polyhedral models.

In Task 1, our method starts from sparse depth data and can be viewed as a depth interpolation, while in Task 2, it starts from dense depth data, and can be considered as a surface approximation or compaction method. Both cases are illustrated in Figure 1.

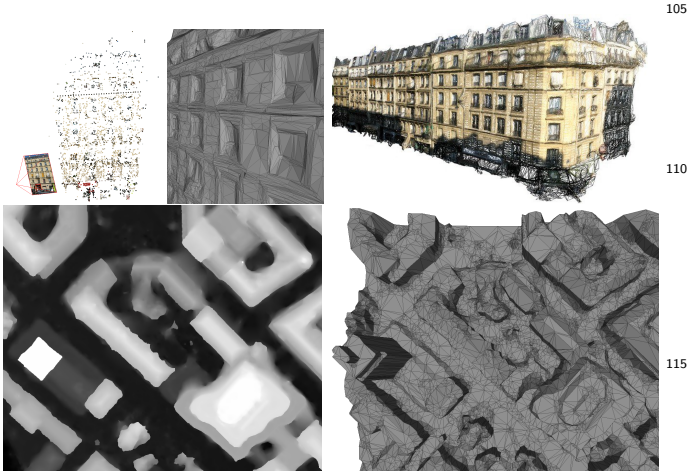

Figure 1: Our method efficiently creates a compact crease- and discontinuity-preserving mesh, given a single view and sparse or dense depth information, as illustrated for sparse street-side SfM (top) and for a dense airborne urban height map (bottom). Our street-side meshes can be combined to form larger models (top-right).

Our method starts by decomposing the input view into partitions. The input view may be a perspective or orthographic 2D observation of a 3D scene, for example, a perspective street-side view or an aerial ortho-view of an urban scene. We discuss partitioning schemes based on images, and propose a compact piecewise-planar decomposition for dense height maps. The partitioning is converted into a 2D base mesh with respect to the view, which is then lifted into 3D in an energy-driven vertex depth reconstruction step. We propose a depth optimization approach that preserves crease edges, and we explicitly detect and handle discontinuities prior to final reconstruction.

Our approach has the following benefits:

- *Watertightness*: our meshes are watertight and free of staircasing artifacts, unlike slanted-plane stereo results, for example.
- *Discontinuities*: these are detected and handled explicitly. We discuss different strategies for different input densities.
- *Edge-alignment*: edges in our final mesh are aligned with crease edges and depth discontinuities. This (1) results in a compact mesh, (2) improves model quality and (3) enables simplified rendering with per-face flat coloring, which may save the complex work of generating a texture atlas.
- *Curvature penalty*: our depth reconstruction penalizes mesh curvature (2nd-order regularization), hence handles textureless areas and missing data in a natural way.
- *Speed and parallellization*: our energy formulation, including the curvature penalty, allows for a sparse linear solver, which renders our depth optimization fast. Given the depth input, our method operates on individual views, hence, it is parallellizable per view. It reconstructs a mesh from SfM input in around 1-2 sec per 1 MPixel image, or can convert a dense 2.6 MPixel height map into a compact 2.5D mesh in around 13 sec on a 3 GHz desktop CPU core.

This paper discusses in more detail the *superpixel mesh* concept that we introduced recently [16], and extends it in several ways:

- *Meshing of dense depth data*: Besides sparse street-side point clouds (Task 1), the method is extended for meshing dense depth data, in particular, large-scale airborne urban height maps (Task 2).
- *Plane-based approach*: Previous image-driven partitioning schemes are complemented with an efficient plane extraction and a novel quality-controlled plane merging that solely rely on a depth or height map.
- *Crease edges*: knowledge about planes is incorporated into the optimization to better preserve crease edges.
- *Discontinuities*: a more precise discontinuity detection and handling approach is proposed, enabled by dense input.
- *Model generation*: our view-driven hole-filling algorithm renders our airborne city models watertight.
- *Experiments*: Additional evaluations compare our method to other meshing methods, in terms of approximation quality and compactness.

The paper is organized as follows. After a literature survey in Section 2, Section 3 gives an overview of the proposed method. In Section 4, we detail our strategy for 2D base mesh extraction, including – but not restricted to – piecewise-planar decomposition and plane merging. Section 5 presents our method for lifting the 2D base mesh into 3D, including data association, depth optimization and discontinuity handling. Experiments are carried out and discussed in Section 6, and the paper is concluded in Section 7.

2. Related work

Since our meshing approach can be viewed as depth reconstruction or interpolation (when applied to sparse 3D points) and as surface approximation or simplification (when applied to dense depth maps), this section attempts to give a literature overview for these aspects.

2.1. Segmentation-based dense stereo

Our reconstruction over an image partitioning is partly inspired by stereo algorithms [17, 18, 19] that make use of an image over-segmentation for aggregating disparity values, and enforce disparity consistency between segments. This strategy effectively propagates depth information from textured to ambiguous, textureless areas [20, 21] and reduces both computational complexity and susceptibility to noise [22, 23]. These methods typically assume fronto-parallel planes [23, 22, 18], slanted planes [17, 19] or other parametric models per superpixel and minimize – but do not eliminate – discontinuity steps between segments, resulting in staircasing artifacts. In turn, our method guarantees continuity on the surface, except for discontinuity edges over which our surface mesh is split explicitly.

2.2. Dense Multi-View Stereo

Multi-View Stereo (MVS) algorithms reconstruct a surface (or volume) from multiple calibrated images, where the calibration is typically provided automatically via SfM.

Volumetric MVS methods partition the scene into cubic voxels [24, 25, 26], tetrahedra [27, 28] or more general convex polyhedral cells [29], and classify these as either inside or outside based on multi-view photoconsistency or line-of-sight constraints, while enforcing regularity. Volumetric methods are known to produce watertight but non-smooth surfaces [30] (unless smoothed through post-processing) and suffer from poor scalability [27]. Yet, thanks to their natural way of fusing observations and because of the advent of cheap memory and GPUs, they increasingly have been applied to large-scale urban scenes. So far this was based on aerial imagery [29, 28, 26]. Meanwhile, it takes hours to reconstruct a street-side scene on a GPU from hundreds of images using CMP-MVS [27], a free, modern volumetric tool that produces high-quality meshes.

Due to its scalability, depth map based (view-centric) MVS [31] has been demonstrated on street-side imagery at city block scale [6, 32]. In [6], real-time performance is achieved via GPU-based plane-sweeps based on the Manhattan assumption and on plane fits to sparse SfM points as priors. A mesh is constructed over depth maps using a quad-tree subdivision algorithm. The resulting mesh edges are not aligned to image gradients. In [32], superpixels are swept in a few dominant directions to improve stability, especially in poorly textured areas. Furukawa *et al.* [4] also strongly regularize MVS via the Manhattan assumption, but only demonstrate this on smaller scenes.

Some recent MVS algorithms developed in Simultaneous Localization and Mapping (SLAM) frameworks can run in real-time on the GPU [33, 34, 35], CPU [36], or on top-grade mobile phones [37]. Many of these capture extremely fine (pixel-wise) detail, but most care little about the compactness of the output.

The latter also applies to patch-based stereo algorithms [38, 39, 40] that directly produce a (semi-)dense 3D point cloud by iterating between 3D patch optimization and multi-view consistency filtering.

Given a coarse mesh (e.g. from volumetric MVS), fine details can be recovered by re-aligning it to the images [41, 28] or jointly evolving the mesh and the camera models [42], while avoiding mesh self-intersections. These approaches enforce multi-view photo-consistency over the mesh faces in an expensive, iterative procedure, and mesh edges are not forced to coincide with actual geometric edges.

2.3. Surface simplification and remeshing

Mesh simplification aims to reduce the number of vertices and faces while staying close to the input mesh.

Vertex clustering methods [43] replace vertex subsets within volumetric partitions by cluster representatives. They are efficient and robust to non-manifold situations, but the quality of the result is not always satisfactory [44].

Incremental mesh decimation [45, 46, 47] is more popular due to a higher level of control over the approximation quality. It iteratively applies elementary topological operations, e.g. edge-collapses, controlled by various quality criteria and rankings. Recently, Salinas *et al.* [48] proposed criteria to prevent important structures – defined by input plane proxies – from collapsing throughout the procedure.

Mesh approximation methods segment the input mesh into partitions and apply parametric models per partition. Starting from random partitions, Variational Shape Approximation (VSA) [49] iterates between mesh partitioning and plane fitting, whereas Lafarge *et al.* [50] insert various parametric 3D primitives into MVS meshes. Gallup et al. [51] segment planar from non-planar regions in multiple views via plane hypotheses from dense MVS depth maps.

Finally, resampling algorithms [52, 44] build a new mesh by connecting samples drawn from the input mesh. They can impose special connectivity, e.g. semi-regularity or quad-meshes. Aliasing artifacts occur where the sampling pattern is not well aligned with the input geometry, however. To prevent this, many approaches rely on an interactive pre-segmentation [44].

2.4. Direct surface extraction from sparse 3D features

In a spirit similar to ours, several methods have been introduced to model the scene directly from imagery and sparse SfM input, but many of these assume piecewise-planarity and do not handle non-planar regions [53, 54, 55, 8, 10]. In our experience, normals and their clustering are not very reliable in sparse SfM point clouds [8, 6], while robust multi-plane fitting usually only captures some of the dominant planes in sparse point clouds.

Another line of work learns depth cues in a training stage to reconstruct the scene from a single image [56, 57, 58]. Saxena *et al.* [59] even incorporates sparse SfM as a cue. These methods rely on strong assumptions about the scene (e.g. Manhattan world) to render the task feasible, and the resulting depth maps are very coarse.

Several methods build a 3D Delaunay Tetrahedralization (3D-DT) spanned by sparse 3D features and use inside/outside volumetric reasoning to find the surface [60, 61, 62, 9]. Hoppe *et al.* [62] found an efficient way to update the tetrahedralization and the optimal surface on-the-fly, while the method of Lhuillier and Yu [9] guarantees manifoldness and shows results on an entire city district.

Others propose to build 2D Delaunay triangulations (2D-DT) over projections of sparse 3D points to find a photoconsistent surface [63] or a surface consistent with lines-of-sights from many views [64, 65]. Geiger *et al.* [66] also exploit a 2D-DT over sparse stereo features to reduce the search space for dense stereo. Our method differs from these in that it builds a 2D-DT aligned with image gradients rather than over SfM point projections, in order to improve model and rendering quality.

Lhuillier [67, 68] also aligns a 2D base mesh to image gradients, but the number of vertices is predefined by a grid independently from the image content, and the alignment procedure is a costly iterative vertex optimization. In contrast, our view partitionings enable direct construction of an edge-aligned 2D mesh, which automatically adapts the number of vertices to the image content for reasons of compactness. In [67], the 3D mesh is obtained from street-side SfM data by iterating between triangle grouping, triangle removal, triangle damping, hole filling and gradient-descent mesh refinement. We propose a different, fast vertex depth optimization and disconintuity handling, and apply them to both SfM data and aerial height maps.

Another way to turn a (dense) point cloud into a mesh is Poisson surface reconstruction [69], but it tends to perform poorly on sparse SfM data, which is inherently non-uniform. Alternatively, Multi-Scale Compactly Supported Radial Basis Functions [70] are applied successfully by Newcome and Davison [33] to fit a coarse base mesh directly to SfM points from each reference view of a hand-held camera. Wang and Yang [71] apply sparse Ground Control Points (GCPs) as hard constraints to interpolate a pixelwise disparity map in a sparse linear formulation, while the pioneering stereo work in [72] solves this with soft constraints. The recent video stablization pipeline *Hyperlapse* [73] fits a proxy mesh to SfM data for image-based rendering, but, unlike in our method, the mesh is not subdivided according to the image content, and its smoothness prior favors fronto-parallel planes.

2.5. Facade modeling from street-side data

Besides MVS methods applied to street-side data, several other works have been proposed for 3D modeling and abstraction of facades. Früh and Zakhor [74] produce textured, dense street-side meshes from LiDAR point clouds. Pylvänäinen *et al.* [75] exploit a 2.5D representation to obtain simplified watertight surfaces from LiDAR points, whereas Gallup *et al.* [76] relax this assumption to an n-layer height map and solve the problem from imagery. Cornelis *et al.* [5] reconstruct simplified canyon-like street models in real-time. The latter works simplify facade models to an extent that important (LoD-3) details are lost. Others use manually ortho-rectified facade images [77, 12], automatic multi-plane fits to SfM points [11, 7], or perform a semantic analysis [78, 13] for facade modeling.

2.6. Modeling buildings from aerial data

Airborne LiDAR or MVS data provide detailed information about roof shapes over large areas. Methods extracting low-complexity polyhedral building models from such data may be classified as model-driven or data-driven (see [1, 2] for a more complete taxonomy). As this type of data rarely allows for modeling overhanging structures or facade details, the 2.5D assumption is widely adopted.

Model-driven (top-down) approaches [79, 80] fit instances of 2.5D parametric roof blocks from a shape vocabulary to better cope with noise, or missing data. They output compact models by discarding details for robustness. To capture various building shapes, each block is placed over a 2D partition in a building footprint decomposition. However, such a decomposition is not straightforward to obtain [81, 79] and its quality greatly affects the result.

The more popular data-driven (bottom-up) methods start by fitting primitives, e.g. lines [82], planes [83, 84], cylinders, cones, spheres [82], or surfaces of revolution [85], to then reconstruct a polyhedral model from these. Such methods are usually less compact but more flexible.

Zebedin *et al.* [85] use 3D lines reconstructed from oblique images to split the 2D domain into cells, and then assign (and elevate) each cell optimally to one of the detected primitives. In a similar vein, Lafarge and Mallet [82] restrict label propagation in a planimetric map by 3D boundary segments, and compile a hybrid 2.5D surface of mesh patches and primitives in large-scale urban areas.

Other authors create 2.5D building models by closing vertical gaps between adjacent regions in a piecewise-planar partitioning of a DSM. The partitions may be obtained by region growing in a LiDAR DSM [84] or by superpixelization of an aerial image [86]. These methods result in discontinuity artifacts at crease edges and between regions lying on the same continuous surface.

To overcome this problem, the more involved *2.5D Dual Contouring* method [87] converts a dense LiDAR point cloud to a mesh by joint iterative simplification of an initial mesh and the underlying 2D domain, while keeping the mesh 2.5D and 2-manifold. Topology control has been added later [88] to prevent important structures from collapsing.

Zhou and Neumann in [89] discover and enforce a variety of architectural regularities between planes in an iterative and greedy fashion. Results are only shown on

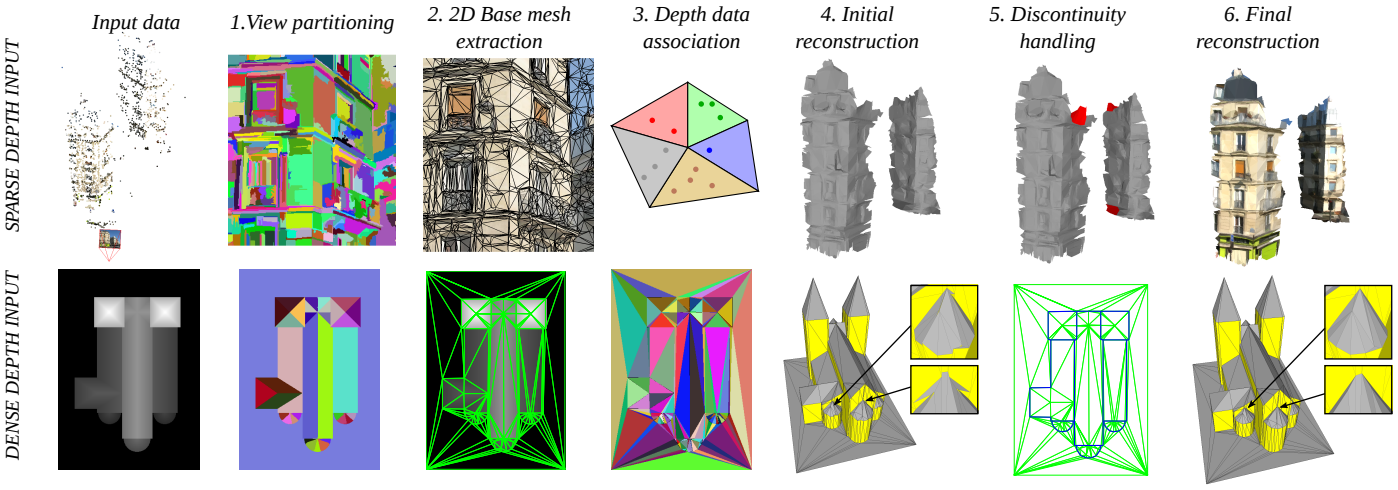

Figure 2: Overview of our method, shown on sparse (top) and dense depth input (bottom). The input view is decomposed into regions, and a 2D base mesh is constructed from these. Next, SfM points (top) or height map cells (bottom) are associated to base mesh triangles. Point/pixel colors encode associated triangles in Step 3. The depth data is used to lift the 2D mesh into 3D. Initial lifting provides evidence for discontinuities (red areas and blue edges in Step 5), which are input to the final reconstruction in Step 6. Best viewed in color.

a few buildings. Verdie *et al.* [90] exploit a regularized plane arrangement to build a watertight model by a fast, approximate volumetric reasoning over a 3D cell complex.

3. An overview of our method

3.1. Input data

In this paper, we demonstrate our approach on two different kinds of input (see Figure 2):

- *Sparse depth*: Street-side images, and the SfM data computed from these, namely, perspective camera models, sparse point cloud and visibility information.
- *Dense depth*: An urban height map (DSM), including geospatial information, derived from airborne nadir imagery using Multi-View Stereo. In this case, no attached ortho-image is needed.

Our approach is not restricted to these inputs though. More generally, our input is sparse or dense depth data given for a fully calibrated perspective or orthographic view of a man-made scene, with an image taken from that view as an additional support in the sparse case. For example, SfM points could be replaced by other Ground Control Points (GCPs)[2] [71], such as reliable sparse matches from stereo vision or sparse LiDAR points with additional visibility information, whereas the DSM could be replaced by depth maps from LiDAR, MVS or structured light (e.g. Kinect data), provided that first gross depth outliers are eliminated. We leave the experimentation with these alternative forms of input for future work.

[2]The term originates from photogrammetry, where GCPs are marked points on the ground with known global positions.

If the input is available for multiple overlapping views, our method treats each view independently. In such a case, multi-view consistency of the resulting meshes is ensured by multi-view principles typically used for pre-computing the depth data. In fact, a significant time is typically spent on SfM, which motivates us to fully exploit this valuable source of information instead of following the typical MVS path by discarding SfM points and densely re-matching the calibrated images. This approach has a natural potential for parallellization per view and allows us to skip pairwise stereo rectification and dense matching.

3.2. Output surface mesh

Our method produces a compact triangle mesh whose edges are aligned with likely crease and discontinuity edges, and whose triangles have their vertices on boundaries of planar areas. In line with challenges of large-scale urban modeling (see Section 1), our focus is compactness besides low approximation error, rather than uniform vertex density or mesh isotropy. In other applications, where the latter properties are more important, triangle-based remeshing algorithms can be used as a post-processing step [44].

The output mesh is an interpolation in case of sparse depth input, and a simplification or summarization in case of dense depth input. Discontinuities are not closed by default, as it is undesirable in many cases. However, an important exception is large-scale 2.5D urban modeling from a nadir ortho-view, where building walls can only be reasonably produced by closing vertical discontinuities. Therefore, we propose an optional discontinuity filling procedure to produce a watertight 2.5D mesh for such use cases.

3.3. Our approach

Our method is illustrated in Figure 2. To achieve the properties discussed in Section 3.2, we partition the im-

Figure 3: Different ways to obtain a 2D base mesh from an image. Rows: (a) Delaunay Triangulation (DT) over image corners, (b) constrained DT over polygonized image edges, (c) constrained DT over polygonized superpixel boundaries. Columns: (1) Harris cornerness (blue for low, red for high) / Canny edge map / superpixel label map, (2) original image with corners and polygonized edges/boundaries in blue, (3) constrained DT over its mean-color rasterization (input SfM points as black crosses), (4-5) 3D reconstruction from sparse SfM.

age or the dense depth input (a height map in our case) into regions whose boundaries are aligned with crease and discontinuity edges, then construct a 2D base mesh with edges aligned to this partitioning in the input view, and finally, we lift the base mesh into 3D via a novel fast vertex depth optimization by using the depth data associated to triangles of the base mesh.

Note that we start from an image-aligned 2D mesh, then reconstruct vertex depths, instead of first obtaining a 3D mesh and then tuning it to the images [41, 28].

Discontinuities require special handling, since discarding them yields distortions and hallucinated surfaces that significantly deteriorate the result. We make use of an initial reconstruction (see Figure 2) and a robust metric to reason about discontinuities, and incorporate the identified discontinuities into the topology of the 2D base mesh prior to final reconstruction.

View partitioning consists of an image over-segmentation into superpixels, or, of a piecewise-planar decomposition of the dense depth input, if available. In the latter case, we efficiently decompose a depth or height map into planes, and propose a plane merging strategy that reduces the number of planes – thus, the number of mesh faces – drastically, while keeping the piecewise-planar approximation error within a prescribed bound. The role of the planar partitioning is two-fold. First, it corresponds to a view decomposition, hence, guides our base mesh extraction step. Second, the plane assignments are re-used in reconstruction, namely, for detecting discontinuities and for improving crease edges in our depth optimization.

4. Base mesh extraction

The first stage of our algorithm constructs a 2D base mesh from either an input image or an input dense depth map or height map. Our local goals here are *(i)* to capture all crease and discontinuity edges *(ii)* and to do so with edges of a maximally compact 2D mesh. These properties enable us to obtain a low polygon-count final 3D model that preserves 3D edge features after the reconstruction stage of Section 5. The first property also improves the approximation quality of the final 3D model, which otherwise gets deteriorated by triangles that cross edge features.

4.1. Base mesh from an image

If an input image is available, we build on the common assumption that crease edges and discontinuities are captured by high image gradients.

We use Delaunay Triangulation (DT) and Constrained Delaunay Triangulation (CDT) [44] to construct a 2D base mesh from an image. CDT forces prescribed line segments into edges of the triangulation.

We have experimented with three methods to obtain a base mesh aligned to image gradients (see Figure 3).

(a) DT of corner-like keypoints detected in the image,
(b) CDT of polygonized binary image edges, and
(c) CDT of polygonized superpixel boundaries.

In case (a) the triangle count is moderated by only retaining the n-best corners. In case (b), binary image edges are turned into polygons by tracing the edge pixels in the image. In case (c), boundaries between superpixels are traced in the segmentation label map, and junctions are identified as points where more than two superpixels meet.

Polygonization is illustrated in Figure 4. We use edges of the polygons to constrain the triangulation. However, we apply Douglas-Peucker polygon simplification [91] in cases (b) and (c) prior to constrained triangulation. This drastically reduces the number of polygon vertices while still preserving image edges. We have found that a low tolerance of 1-3 pixels is a good trade-off between edge-adherence of the triangulation, and low triangle count. In case (c), the polygon simplification is applied separately to each polyline between any two junctions. This preserves the junctions, thus, the coarse structure of the segmentation (right side of Figure 4).

An experiment in Section 6 shows that the partitioning (c) using the graph-based segmentation algorithm of Felzenszwalb and Huttenlocher [92] gives better-quality results than (a), (b) and than some other superpixels.

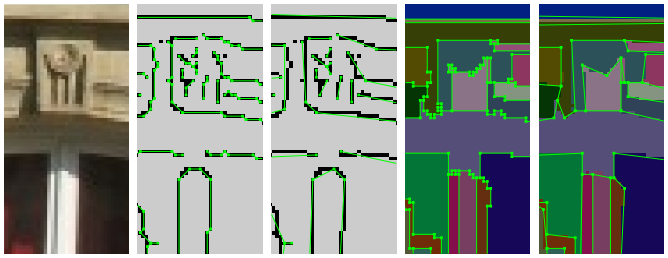

Figure 4: Polygonization of image edge pixels and superpixel boundaries, each with a polygon simplification tolerance of 0 (left) and 2 pixels (right). Polygons are shown in green.

4.2. Base mesh from a dense depth or height map

If a dense depth map or height map (DSM) is given, it provides direct evidence for geometric features, unlike image gradients, but precise localization of creases and discontinuities remain a challenge due to noise and blur. For example, typical oversmoothing in an urban DSM makes roof superstructures appear as bumps on top of the roofs, not to speak about blurred discontinuities at walls.

We extract a 2D base mesh from a dense depth or height map in two steps. First, the depth map is decomposed into locally planar regions, which results in a view partitioning and a plane arrangement approximating the surface (detailed in Sections 4.3 and 4.4). Second, the 2D base mesh is constructed by polygonizing segment boundaries in the label map of the segmentation, and performing a CDT over the polygon edges, that is, by using method (c) described earlier in Section 4.1.

4.3. Piecewise-planar partitioning of dense depth maps

In order to partition a depth (height) map into planar regions, one could adapt Variation Shape Approximation (VSA) [49] to depth maps. VSA computes a piecewise-planar approximation of a mesh, by growing competing planar regions from random seeds, where the number of seeds (planes) is manually given. However, one of our interests is meshing large-scale urban height maps. For these, it is not reasonable for the user to guess the number of planes present in the scene and then to start growing them from random seeds, as under-estimation of the number of planes would easily merge important structures (e.g., walls with ground), and its over-estimation would result in non-compact models. We find it more reasonable to provide error bounds that are lower than the scale of important features and leave it to the algorithm to automatically decide on the number of planes.

Therefore, we use a fast sequential plane growing algorithm for the partitioning (see Algorithm 1). First, we characterize each pixel (x, y) of the depth or height map $D(x, y)$ by its normal $\mathbf{n}(x, y)$ and mean curvature $H(x, y)$ on the surface. We aim at finding a plane arrangement $\Pi = \{\pi_k\}$ and assigning every pixel of the depth map (x, y) to one of the planes. Pixels are considered in increasing order of $|H(x, y)|$ as seeds. A new region $\mathcal{R}_k$ is started from the next best unassigned seed. The seed with its normal determines a plane π_k. Adjacent pixels are added to the region iteratively if both their depths and normals are compatible with the plane π_k. More precisely, a pixel (x, y) is considered to be an inlier for a plane π_k, if the angle between its normal $\mathbf{n}(x, y)$ and the plane normal $\mathbf{n}_k$ is within an angle threshold θ and the 3D point $(\gamma x, \gamma y, D(x, y))$ is within a distance threshold δ from π_k. γ denotes the pixel size in depth (height) units, known as Ground Sampling Distance in photogrammetry. The expansion of $\mathcal{R}_k$ stops when it runs out of adjacent inliers.

It should be noted that the curvature map $H(x, y)$ can be safely calculated over a relatively large support for noise reduction, as it only affects seed ranking. In turn, the normal map $\mathbf{n}(x, y)$ is computed over 3×3 pixel neighborhoods to minimize smoothing across creases and discontinuities. Unfortunately, this results in noisier normals, hence, imprecise initial planes. Therefore, we refit each plane π_k to its inliers on-the-fly every time the support region $\mathcal{R}_k$ increases by a factor of κ, in case the minimum number of inliers $s_{min} = 3$ is reached.

A result of Algorithm 1 is shown at the top of Figure 5.

4.4. Region merging with quality control

If the inlier thresholds δ and θ of Algorithm 1 are tightened to achieve a lower approximation error, the number of planes increases. Fragmentation is particularly noticeable around discontinuities, where normals are noisy. Oversmoothed crease edges are also a typical source of fragmentation. See Figure 5 for an illustration. In practice, a large part of the planes are small fragments. This is undesirable, as the least important details (noise) would induce the most triangles in the base mesh.

Discarding the normal constraint in the growing procedure is not a good solution, as it would e.g. allow non-horizontal planes to expand along a line over horizontal regions. In our experience, both the distance and the normal constraints are important in Algorithm 1. Another obvious attempt to overcome fragmentation would be to relax the inlier thresholds. However, it does not only make the

Algorithm 1: Fast Sequential Plane Growing

input : depth map $D(x,y)$ of N pixels, normals $\mathbf{n}(x,y)$, seed ranking seed : $i \mapsto (x,y)$, $\forall (x,y)$
output: planes $\Pi = \{\pi_k\}$, label map $L : (x,y) \mapsto k$
param : distance threshold δ, angle threshold θ refitting frequency params κ and s_{min}

```
init: Π ← ∅, ∀L(x,y) ← 0, ∀Q(x,y) ← 0, l ← 0
for prio ← 1 to N do
    (x,y) ← seed[prio]
    if L(x,y) ≠ 0 then continue
    queue ← (x,y)
    Q(x,y) ← 1
    l ← l + 1
    inliers ← ∅, support ← 0
    initialize plane π from (x,y) and n(x,y)
    while queue ≠ ∅ do
        (x,y) ← pop next from queue
        L(x,y) ← l
        inliers ← inliers ∪ (x,y)
        if |inliers| ≥ max(κ · support, s_min) then
            π ← FitPlaneLeastSquares(inliers)
            support ← |inliers|
        foreach (u,v) adjacent to (x,y) do
            if (u,v) compatible with π and Q(u,v) = 0 then
                queue ← queue ∪ (u,v)
                Q(u,v) ← 1
    Π ← Π ∪ π
return Π, L(x,y)
```

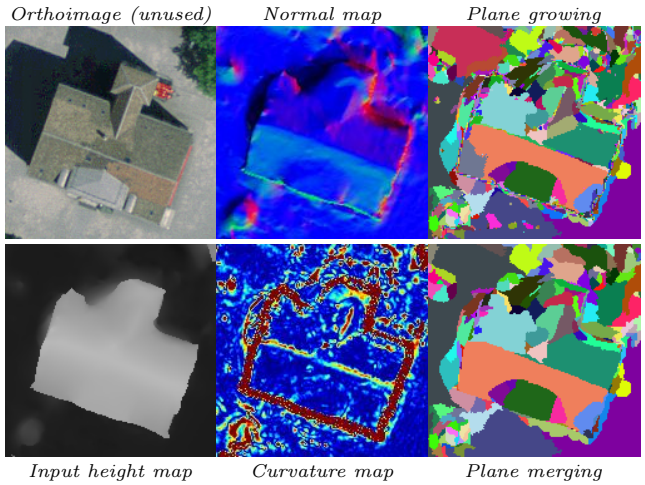

Figure 5: Height map partitioning by plane growing (Algorithm 1) and merging (Algorithm 2). The label maps on the right show that merging effectively removes fragments around creases and discontinuities without changing stable planes, while keeping the approximation error within limits (see Section 6 for a numerical evaluation).

approximation coarser, but also has the disadvantage that noisy crease and discontinuity regions affect the planes extending to such places from more stable regions.

Instead, we use tight thresholds for better approximation quality and better plane precision in planar regions, and propose an additional plane merging strategy to handle small and noisy regions while reducing their effect on the planes of more stable regions.

Our merging algorithm is inspired by edge-collapse mesh decimation methods [44], which typically use a priority queue to rank candidate edge collapses, while having an additional quality control that decides whether a candidate collapse is valid or not. In our algorithm, we rank pairs of candidate regions for merging in the priority queue, and have a binary quality control in terms of an approximation error bound ϵ. ϵ has to be relaxed compared to δ, since the merging procedure always increases the approximation error to obtain a more compact solution.

Given the depth map $D(x,y)$, the planes $\Pi = \{\pi_k\}$ and the label map $L : (x,y) \mapsto k$ from Algorithm 1, the merging procedure returns a new label map $L'(x,y)$ that assigns pixels (x,y) to a subset of the input planes. Therefore, input planes are not altered, only selected.

The approximation error of a plane π over a region $\mathcal{R}_k$ of the label map is defined as

$$e(\mathcal{R}_k, \pi) = d(\mathcal{I}_k, \pi) = \max_{p \in \mathcal{I}_k} d(p, \pi), \tag{1}$$

where $\mathcal{I}_k$ is the set of support pixels of $\mathcal{R}_k$ and $d(p,\pi)$ denotes the Euclidean distance between the plane π and the 3D point p observed at pixel (x,y). We define the global error of a plane arrangement $\Pi = \{\pi_k\}$ as

$$\bar{e} = \frac{1}{|\mathcal{R}|} \sum_{k=1}^{|\mathcal{R}|} e(\mathcal{R}_k, \pi_k), \tag{2}$$

where $\mathcal{R}$ is a partitioning, that is, the set of all regions.

Each region $\mathcal{R}_k = (\pi_k, \mathcal{I}_k, \mathcal{N}_k, e_k)$ is characterized by a plane π_k, the set of inliers $\mathcal{I}_k$, the set of adjacent regions $\mathcal{N}_k$ and the associated approximation error $e_k = d(\mathcal{I}_k, \pi_k)$. A candidate pair for merging is denoted by $(\mathcal{R}_i, \mathcal{R}_j)$, where $\mathcal{R}_i$ is always the larger region, $|\mathcal{I}_i| > |\mathcal{I}_j|$. The merged region $\mathcal{M}$ inherits its properties from $\mathcal{R}_i$ and $\mathcal{R}_j$. Most importantly, it inherits the plane of the larger region, $\mathcal{R}_i$, in order to propagate more stable planes, and to merge in the direction of a lower approximation error.

The algorithm starts by enumerating all adjacent regions as candidate pairs $(\mathcal{R}_i, \mathcal{R}_j)$ for merging. These are ranked according to a certain criterion and pushed into a priority queue. During the procedure, the next best candidate is popped from the queue, and the merge is only accepted if the error $e(\mathcal{M}, \pi_i)$ of the merged region $\mathcal{M}$ is lower than ϵ. In order to avoid a large number of repeated distance evaluations, the error is computed recursively for the merged region, i.e. by reusing the available residuals:

$$e_{ij} = e(\mathcal{M}, \pi_i) = \max\{e_i, e(\mathcal{R}_j, \pi_i)\}. \tag{3}$$

New candidate pairs – formed from the merged region and its neighbors – are pushed into the queue. Since multiple merge candidates are present in the queue that contain the same region $\mathcal{R}_k$, the winner renders other candidates obsolete. If an obsolete candidate is popped from the queue, it is simply discarded. In order to improve memory efficiency, we pre-compute the error of a merge and do not even push candidates into the queue that violate the error

Algorithm 2: Quality-Controlled Plane Merging

input : Depth map $D(x,y)$, Planes $\Pi = \{\pi_k\}_{k=1}^{K}$,
Label map $L : (x,y) \mapsto k$
output: Label map $L' : (x,y) \mapsto k$.
param : error tolerance ϵ

$\forall \mathcal{I}_k \leftarrow \emptyset$, prio_queue $\leftarrow \emptyset$
foreach (x,y) **do** $\mathcal{I}_{L(x,y)} \leftarrow \mathcal{I}_{L(x,y)} \cup (x,y)$

for $k \leftarrow 1$ **to** K **do**
  $\mathcal{N}_k \leftarrow$ all regions adjacent to $\mathcal{R}_k$ in $L(x,y)$
  $e_k \leftarrow$ `MaxDistance`$(\mathcal{I}_k, \pi_k, D)$
  $\mathcal{R}_k \leftarrow (\pi_k, \mathcal{I}_k, \mathcal{N}_k, e_k)$
  $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}_k$;
foreach *adjacent pair* $(\mathcal{R}_i, \mathcal{R}_j)$ **do**
  `PushCandidate` $(\mathcal{R}_i, \mathcal{R}_j)$
while prio_queue $\neq \emptyset$ **do**
  $(\mathcal{R}_i, \mathcal{R}_j, e_{ij}, \text{prio}) \leftarrow$ pop candidate from prio_queue
  if $\mathcal{R}_i$ *or* $\mathcal{R}_j$ *obsolete* **then continue**
  $\mathcal{N} \leftarrow \mathcal{N}_i \cup \mathcal{N}_j \setminus \{\mathcal{R}_i, \mathcal{R}_j\}$
  $\mathcal{M} \leftarrow (\pi_i, \mathcal{I}_i \cup \mathcal{I}_j, \mathcal{N}, e_{ij})$
  $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{M}$
  mark $\mathcal{R}_i$ and $\mathcal{R}_j$ obsolete
  foreach $\mathcal{R}_k \in \mathcal{N}$ **do**
    $\mathcal{N}_k \leftarrow \mathcal{N}_k \cup \mathcal{M} \setminus \{\mathcal{R}_i, \mathcal{R}_j\}$
    `PushCandidate` $(\mathcal{M}, \mathcal{R}_k)$
foreach $\mathcal{R}_k \in \mathcal{R}$ **do**
  foreach $(x,y) \in \mathcal{I}_k$ **do** $L'(x,y) = k$
return $L'(x,y)$ **procedure** `PushCandidate` $(\mathcal{R}_i, \mathcal{R}_i)$
$e_{ij} \leftarrow$ `MaxDistance`$(\mathcal{I}_i \cup \mathcal{I}_j, \pi_i, D)$
if $e_{ij} > \epsilon$ **then**
  prio $\leftarrow$ `ComputePriority`$(\mathcal{R}'_i, \mathcal{R}'_j)$
  prio_queue $\leftarrow (\mathcal{R}'_i, \mathcal{R}'_j, e_{ij}, \text{prio})$

tolerance. The ranking of such pairs can also be skipped. The procedure is detailed in Algorithm 2.

We experimented with the following different criteria for ranking candidate merges $(\mathcal{R}_i, \mathcal{R}_j)$:

- Least increase in the global approximation error.
- Area ratio $a_{ij} = |\mathcal{I}_j|/|\mathcal{I}_i| \leq 1$ that favors the occupation of the smallest regions by the largest neighbors earlier.
- The dihedral angle between plane π_i and π_j favors earlier merging of regions with less normal discrepancy.

We discuss advantages and disadvantages of each ranking scheme in our experimental section (Section 6).

This merging algorithm effectively occupies small fragments by adjacent stable regions, as shown in Figure 5, while leaving the plane of the larger region intact, and by respecting the global error tolerance ϵ. We show in Section 6 that this method reduces the number of planes by an order of magnitude on a large-scale urban DSM, with only a little sacrifice in terms of approximation error.

5. 3D mesh reconstruction

This section considers the problem of optimally lifting a 2D base mesh from a perspective or orthographic view into 3D via known depth values of any density (Steps 3 to 6 in Figure 2). Edges of the 2D base mesh are assumed to be aligned to creases and discontinuities of the surface to be reconstructed or simplified. Section 4 described how to obtain such a mesh.

5.1. Depth data association

During reconstruction, we fit triangles of the base mesh to depth data (Section 5.3). Thus, a preliminary step (see Step 3 in Figure 2) is necessary to associate each GCP or depth map pixel (x,y) to a triangle t_i of the base mesh. The result can be encoded as a mapping $T : (x,y) \mapsto f$ and by its inverse, the set of support points $\mathcal{S}_i$ of each triangle t_i.

Both the number of triangles and the number of pixels may exceed tens of thousands. Instead of an exhaustive search, we first rasterize all triangles of the base mesh into a triangle map $T(x,y)$, then sample the triangle indices from this map at GCP locations. We note that both our depth optimization (Section 5.3) and sparse discontinuity handling (Section 5.5) are robust to sub-pixel errors in the associations that are due to rasterization.

In case of a depth map, $T(x,y)$ directly associates the depth data with the triangles $\{t_i\}$, assuming it is rasterized at the resolution of the depth map. Unfortunately, this direct association typically assigns outlier points to each triangle close to triangle edges, due to the fact that base mesh edges do not follow pixel-wise variations of creases and discontinuities in favor of compactness (see the polygon simplification in Figure 4). While an *a posteriori* search for outlier depths per triangle would do, it is more reasonable to exploit the piecewise-planar segmentation previously used for base mesh extraction (Sections 4.3 and 4.4). To this end, we inspect the triangle map $T(x,y)$ and the plane map $L'(x,y)$ jointly, and associate each triangle to the plane that occupies the majority of pixels inside the triangle. All other pixels are invalidated (set to 0) in $T(x,y)$. In the following, we will denote the derived association of base mesh triangles t_i to planes π_k by $T_\pi : t_i \mapsto \pi_k$. In summary, a robust data association can be achieved by combining the result of piecewise-planar segmentation of Sections 4.3 and 4.4 with the majority vote over base mesh triangles.

5.2. A preliminary reconstruction via planes

The piecewise-planar segmentation of a depth map (Sections 4.3 and 4.4) is not only useful for base mesh extraction (Section 4.2) and robust data association (Section 5.1), but it can also be exploited to obtain a preliminary piecewise-planar reconstruction of the base mesh (see the bottom of Step 4 in Figure 2). We will make use of such a preliminary mesh solely for detecting depth discontinuities (see Section 5.4) prior to final reconstruction.

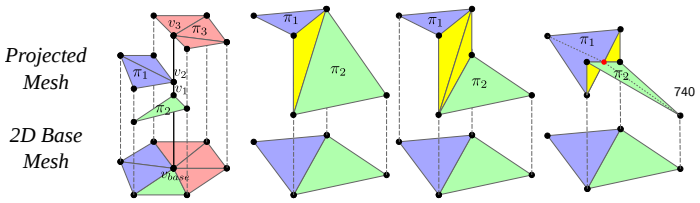

Figure 6: Basic reconstruction by back-projecting each base mesh triangle t_i to its associated plane π_k. Each base vertex v_b is replicated along the viewing direction (left). Three different cases for closing discontinuities (right). Triangle colors denote plane associations, except for the yellow triangles that are inserted optionally.

For this purpose, we project each triangle t_i of the base mesh to its associated plane $T_\pi(t_i)$, where we use the notation introduced in Section 5.1. Here, base mesh triangles are handled separately, i.e. three vertex depth values are obtained per triangle. This replicates each base vertex in the lifted mesh as many times as the number of planes adjacent to the base vertex (see the left of Figure 6).

When the goal is to generate a watertight mesh, e.g. in 2.5D urban modeling, discontinuities in between the lifted triangles can be closed by inserting one or two extra vertical faces per base edge e_{ij} (yellow triangles in Figure 6). Moreover, it is possible to moderate the number of vertices and discontinuity-triangles in the lifted mesh by a vertex clamping procedure prior to closing discontinuities. It consist of a depth clustering in the projected 3D vertex set per base vertex ($\{v_1, v_2, v_3\}$ in Figure 6) using a depth threshold D_{clamp}, and replacing each cluster by its average depth. This improves compactness and can reduce the staircasing effect, but it increases the approximation error as the clamping heuristic does not enforce fidelity to the input (DSM) points.

In our proposed pipeline, we only exploit this preliminary reconstruction of disconnected triangles for reasoning about discontinuities (Section 5.4), which does not require watertightness or vertex clamping. We use the latter steps (with $D_{clamp} = 0$) for the purpose of better visualization of the preliminary models, however. Instead, we propose another depth reconstruction method which uses triangle connectivity constraints and produces a final output mesh free of staircasing artifacts.

5.3. Vertex depth optimization with connectivity

Given the 2D base mesh of Section 4, the sparse or dense depth input and its association to triangles of the 2D base mesh (Section 5.1), we consider the problem of lifting the base mesh to the data in an optimal way, while respecting triangle connectivity.

As mentioned in Section 2.1, superpixel stereo methods apply energy terms that force surfels of adjacent superpixels to be close (except for likely discontinuities), but can not exactly connect the polygonal pieces in 3D, which results in staircasing artifacts everywhere, similarly to the preliminary reconstruction of Section 5.2 (Figure 6). In turn, we follow the opposite strategy: a watertight blanket is assumed everywhere except where discontinuities are explicitly detected. To keep the model watertight, we slightly relax the requirement that each polygonal segment in the view partitioning must observe a single plane or parametric 3D patch. We postpone our discontinuity handling strategy to upcoming Sections 5.4 and 5.5.

5.3.1. Energy formulation

To reconstruct the depths $\{\hat{d}_i\}$ at vertices $\mathcal{V} = \{v_i\}_{i=1}^{V}$, we rely on the GCPs or pixel centers $\mathcal{P} = \{p_i\}_{i=1}^{N}$ with known respective depths $\{d_i\}$. We require the 3D triangles to fit the observed depths and that triangles are smoothly connected. We formulate this as the minimization of a fitting and a smoothness term:

$$E(\hat{\mathbf{d}}) = E_{fit}(\hat{\mathbf{d}}; \mathbf{d}) + \lambda E_{smooth}(\hat{\mathbf{d}}), \tag{4}$$

where $\hat{\mathbf{d}} = (\hat{d}_1, \hat{d}_2, \ldots, \hat{d}_V)$ and $\mathbf{d} = \{d_1, d_2, \cdots, d_N\}$ are the vectors of vertex depths and data depths, respectively, and λ is a scalar balance between our fitting quality and smoothness terms.

We propose to use a least-squares formulation and linear interpolation via barycentric coordinates to obtain a simple quadratic form for Equation (4), which can be minimized very efficiently by a linear solver.

5.3.2. Fitting term

If a data point p_i is associated to a 2D base mesh triangle t_i defined by three vertices $\{v_p, v_q, v_r\}$, then it can be written as $p_i = \alpha_{pi} v_p + \alpha_{qi} v_q + \alpha_{ri} v_r$, where $(\alpha_{pi}, \alpha_{qi}, \alpha_{ri})$ are the barycentric coordinates of data point p_i with respect to triangle t_i. Using these barycentric coordinates, the (known) depth of the data points can be linearly interpolated from the (unknown) depths of the vertices as $\tilde{d}_i = \alpha_{pi}\hat{d}_p + \alpha_{qi}\hat{d}_q + \alpha_{ri}\hat{d}_r$. Collecting the equations for all data points $\mathcal{P} = \{p_i\}$, we obtain the matrix form $\tilde{\mathbf{d}} \triangleq (\tilde{d}_1, \ldots, \tilde{d}_N)^T = \mathbf{A}\hat{\mathbf{d}}$, where $\mathbf{A}$ is an $N \times V$ sparse matrix of all the barycentric coordinates with up to $3N$ non-zeros. The unary term is

$$E_{fit}(\hat{\mathbf{d}}; \mathbf{d}) = (\mathbf{d} - \mathbf{A}\hat{\mathbf{d}})^T \boldsymbol{\Sigma}^{-1} (\mathbf{d} - \mathbf{A}\hat{\mathbf{d}}), \tag{5}$$

where $\boldsymbol{\Sigma}$ is the covariance matrix of the known depths $\mathbf{d}$. Note that the minimizer of Equation (5) is the Maximum Likelihood Estimate (MLE) under the zero-mean Gaussian noise model with covariance matrix $\boldsymbol{\Sigma}$.

5.3.3. Smoothness term

In order to obtain a comparably simple quadratic form for the smoothness term, most methods use a simple squared penalty for the depth differences of adjacent pixels [72, 71] or adjacent vertices of a regular grid mesh [73]. Unfortunately, such a choice favors fronto-parallel planes. Instead, we present a different formulation that rather penalizes curvature as a 2nd-order regularization, yet in a quadratic formulation that allows for a linear solution of Eq. (4).

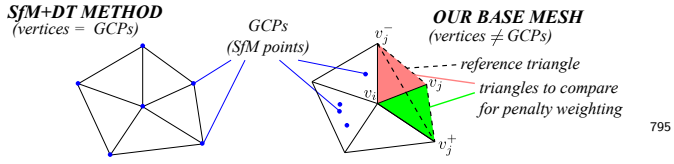

Figure 7: Distinction between Delaunay Triangulation (DT) over GCPs (left) and our base mesh reconstruction (right). 1-ring vertex neighbors are used to calculate our smoothness term penalizing curvature. The depth difference of v_i from the dashed triangle is penalized.

We consider, for each vertex v_i, the triangle t_{ij} formed by a neighboring vertex v_j and its previous and next adjacent vertex v_j^- and v_j^+ in the oriented 1-ring neighborhood of v_i (see the right side of Figure 7), and linearly interpolate the (unknown) depth $\hat{d}_{ij}$ of the central vertex v_i from the (unknown) respective depths $\hat{d}_i$, $\hat{d}_j^-$ and $\hat{d}_j^+$ of adjacent vertices v_j, v_j^- and v_j^+ via barycentric coordinates. This can be written as

$$\hat{d}_{ij} = \beta_{ij}\hat{d}_j + \beta_{ij}^+\hat{d}_j^+ + \beta_{ij}^-\hat{d}_j^- . \qquad (6)$$

In case the triangles on the two sides of the edge (v_i, v_j) are in the same plane, the central vertex v_j (in Figure 7) must lie in the plane of $\{v_j, v_j^-, v_j^+\}$, in which case $\hat{d}_{ij} = \hat{d}_i$.

To favor local smoothness, we penalize for the difference between the depth $\hat{d}_i$ of vertex v_i and its interpolated depths $\hat{d}_{ij}$ from the different triangles of adjacent consecutive vertex triplets. Roughly speaking, this drives the depth of any vertex v_i towards a plane approximately passing through its 1-ring neighbors.

An additional weighting term w_{ij} can be used per edge of the base mesh. For example, this allows for a weighting based on the observed color difference of the two triangles meeting in edge $\{v_i, v_j\}$ (red and green triangles in Figure 7), which often correlates with local planarity in man-made scenes [10]. w_{ij} also allows for incorporating plane-based knowledge into the optimization to better preserve crease edges, as will be discussed later in Section 5.3.4.

Using the individual edge weights w_{ij}, the curvature penalty term is defined as

$$E_{ij}(\hat{\mathbf{d}}) = w_{ij}^2(\hat{d}_i - \hat{d}_{ij})^2 = w_{ij}^2(\mathbf{b}_{ij}^T\hat{\mathbf{d}})^2, \qquad (7)$$

where $\mathbf{b}_{ij}$ is a vector of length V containing only the 4 non-zero elements $\{1, -\beta_{ij}, -\beta_{ij}^+, -\beta_{ij}^-\}$.

Collecting all constraints in a matrix form, the pairwise term in (4) becomes

$$E_{smooth}(\hat{\mathbf{d}}) = \hat{\mathbf{d}}^T\mathbf{B}^T\mathbf{W}^2\mathbf{B}\hat{\mathbf{d}}, \qquad (8)$$

where $\mathbf{B}$ is a $V \times V$ sparse matrix formed by vertically stacking the row vectors $\mathbf{b}_{ij}^T$, and $\mathbf{W} = \mathrm{diag}\{w_{ij}\}$ is a diagonal matrix of all adjacency weights w_{ij}. Of the V^2 elements of $\mathbf{B}$, around $18V$ or less are non-zeros, as vertex valence (the number of neighbors) is 6 for interior and 4 for boundary vertices in a regular triangle mesh [44].

5.3.4. Incorporating planes for improved crease edges

The introduction of the edge weights w_{ij} in Equation (8) allows for the incorporation of plane-related knowledge into the optimization, since the smoothness term should be forgiving for high curvatures at crease edges but more strict otherwise. The assignments $T_\pi(t_i)$ of base mesh triangles t_i to planes provide information about crease edges. Therefore, in our experiments with dense input (when planes are available from piecewise-planar partitioning), we set $w_{ij} = 10^{-3}$ for mesh edges where triangles assigned to different planes meet and $w_{ij} = 1$ is set in all other cases.

5.3.5. Optimization

After substitution of Equations (5) and (8) into (4), and differentiation with respect to the depths $\hat{\mathbf{d}}$ of the base mesh vertices, the minimization of (4) boils down to solving the linear system

$$(\mathbf{A}^T\mathbf{\Sigma}^{-1}\mathbf{A} + \lambda\mathbf{B}^T\mathbf{W}^2\mathbf{B})\hat{\mathbf{d}} = \mathbf{A}^T\mathbf{\Sigma}^{-1}\mathbf{d}. \qquad (9)$$

for the vertex depths $\hat{\mathbf{d}}$, given the depths $\mathbf{d}$ at sparse or dense points. This is a sparse linear $V \times V$ system (where V is the number of vertices in the 2D base mesh), which can be solved efficiently by using a linear solver.

The $N \times N$ matrix $\mathbf{\Sigma}$ is the covariance matrix of vector $\mathbf{d}$. For simplicity, we assume equal variance and no correlation between the input depth values ($\mathbf{\Sigma} = \mathbf{I}$).

Figure 8 illustrates the described approach and the effect of the smoothing parameter λ.

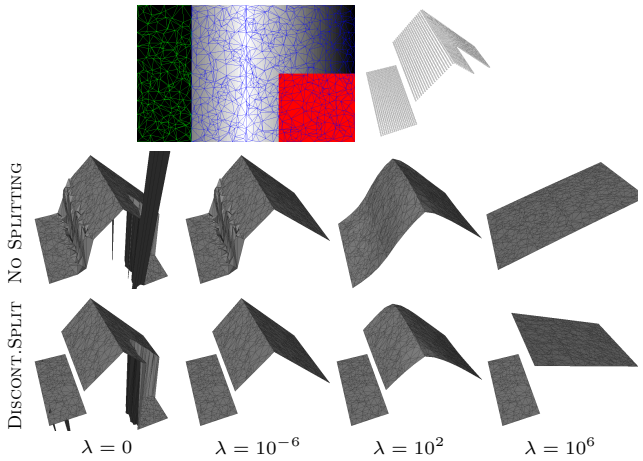

Figure 8: The proposed depth optimization (Section 5.3). Input height map with missing data in red and overlaid 2D base mesh, and its 2.5D point cloud representation (top). Reconstructions with increasing λ without (middle) and with (bottom) discontinuity splitting (Section 5.4). Minor smoothing ($\lambda = 10^{-6}$) already completes missing parts, while crease edges are preserved in a wide range of λ.

The smoothness term of Equation (4) is disabled if $\lambda = 0$. In this case, vertices whose adjacent triangles do not contain valid depth data can not be reconstructed at all, the resulting vertex depths are arbitrary (see results for the red area in Figure 8). This also holds for narrow or

minor triangles that do not contain any pixel centers with known depth (notice the spikes on the left of Figure 8).

A minor non-zero smoothness parameter is already sufficient to stabilize all depth variables and render the system solvable, and a minimum of three GCPs in total are enough to obtain a stable – although planar – reconstruction of the whole base mesh. Further important features, apparent in Figure 8 are the following:

- Missing-data areas (red in the figure) are filled with minimal curvature when $\lambda \neq 0$.
- Since edges of the base mesh are aligned with crease edges of the surface in the example, these are preserved in the reconstruction over a wide range of λ.
- At very high λ value, each component of the base mesh becomes planar.
- Discontinuities are closed, and undesirable minor oscillations occur around them (see the case of no splitting besides $\lambda = 0$ and 10^{-6} in Figure 8).

5.4. Handling discontinuities in dense depth maps

The properties of the 3D mesh produced by our depth optimization depend on the structure and topology of the input 2D base mesh. Crease edges can only be perfectly preserved, if the edges of the 2D base mesh are aligned to these. Moreover, if the 2D base mesh is free of cracks and holes, the reconstructed 3D mesh is also watertight. Fortunately, a crack-free and hole-free base mesh is not a requirement of our depth optimization. This allows us to detect discontinuities and incorporate them into the topology of the base mesh, prior to reconstruction. The bottom of Figure 8 shows that if the base mesh is aligned to discontinuities and is split exactly along these, our depth optimization finds the actual surface properly. Thus, two preliminary tasks need to be solved: discontinuity detection and base mesh splitting.

5.4.1. Discontinuity detection

The task of deciding, for each edge of the 2D base mesh, whether it is a discontinuity or not requires an appropriate measure for the depth separation between adjacent triangles of the base mesh. To this end, we analyze a coarse initial reconstruction obtained by projecting 2D triangles to their planes (see Section 5.2). This initial reconstruction contains two types of artifact, as shown in Figure 9:

1. *staircasing* between adjacent planar regions due to the piecewise-planar approximation per segment,
2. *over- and undershoots* of near-vertical triangles due to blurry discontinuities of the input depth map (typical for an urban DSM) that results in imprecise planes nearly parallel to the viewing direction.

As a result, the naive approach of measuring separation in depth between triangles (e.g. the height of yellow triangles in Figure 9) tends to identify discontinuity edges

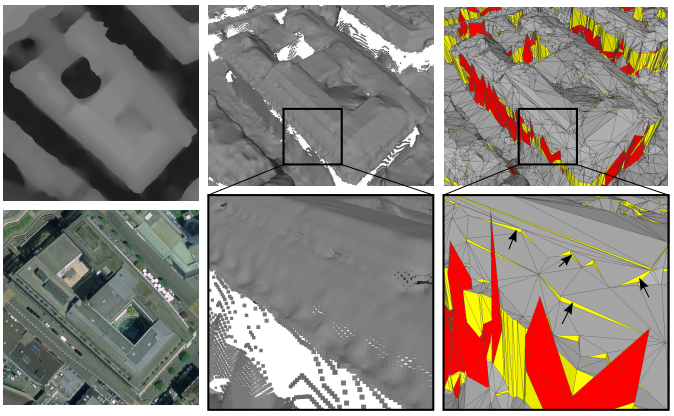

Figure 9: Challenges of discontinuity detection: artifacts in a naive per-triangle preliminary reconstruction (Section 5.2). Ortho-image and DSM of a building (left), 2.5D point cloud corresponding to the DSM (middle), initial reconstruction with staircasing (black arrows) and overshoot artifacts (red triangles).

around the overshoots of steep regions. Splitting the base mesh along these edges would leave steep regions of the mesh (the red triangles in Figure 9) unattached to the rest and the overshoot could appear in the final result even after depth optimization due to a lack of connectivity.

These problems are tackled in two steps. First, we consider spurious triangles of the base mesh as *discontinuity areas*, and eliminate them based on the angle θ_{disc} between their normals and their lines of sight. The eliminated areas are always where large discontinuities are blurred in the input, and since we only apply the depth optimization of Section 5.3 to the remaining triangles (after the base mesh splitting in Section 5.4.2), the input points in blurred discontinuity regions do not contribute to the final result, as expected.

Second, instead of measuring the separation of lifted triangles in depth (along the viewing direction), we measure the minimum of the perpendicular distances to the planes of adjacent triangles, as illustrated in 2D in Figure 10. Using the notation on the right of the figure, our

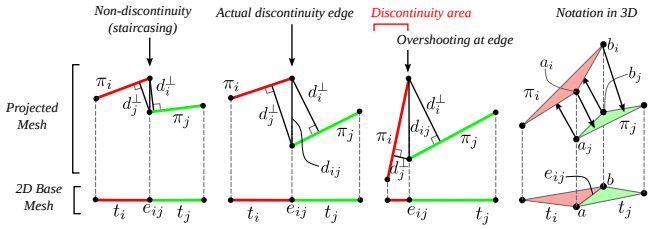

Figure 10: Left: 2D analogy (side views) of cases for discontinuity detection in the lifted mesh of Section 5.2. $\min\{d_i^\perp, d_j^\perp\}$ is a more robust measure of discontinuity than the vertical separation d_{ij}. Right: notation in 3D for Equation (10).

robust measure of the discontinuity along any edge e_{ij} is

$$D_{disc}(e_{ij}) = \max\{\min(d(a_i, \pi_j), d(a_j, \pi_i)), \min(d(b_i, \pi_j), d(b_j, \pi_i))\}, \quad (10)$$

where $d(x, \pi)$ is the distance of point x from plane π. The advantage of this formula is that it gives a low value at the overshoots. An edge e_{ij} of the 2D base mesh is marked a discontinuity edge if $D_{disc}(e_{ij}) < \Delta D_{disc}$. The threshold ΔD_{disc} encodes what height separation can be considered a discontinuity and what an inaccuracy in between planes.

In summary, discontinuities are detected at edge-precision where they are sharp in the input, and as 2D areas (over base mesh triangles) where the input data is too blurry.

5.4.2. Base mesh splitting

The elimination of discontinuity *areas* (triangles) of the 2D base mesh introduces holes in blurred discontinuity regions, and leaves the depth optimization unconstrained across the hole, as desired. Therefore, we only need to consider the edges marked as (sharp) discontinuities, and focus on the problem of splitting the 2D base mesh along these, prior to vertex depth optimization.

The splitting consists of duplicating internal vertices with at least two adjacent discontinuity edges, and boundary vertices with at least one, and of updating the connectivity according to the discontinuity edges. Due to the holes introduced in blurred discontinuity areas, non-manifold vertices occur and need to be handled.

As illustrated in Figure 11, our algorithm visits each vertex of the 2D base mesh, cycles through the list of adjacent edges in its 1-ring neighborhood in the order of consistent orientation, and performs a connected component search on-the-fly. A new component is found whenever two consecutive triangles are not connected, or, when an edge marked as discontinuity edge is crossed. The vertex is replicated for each such 1-ring component.

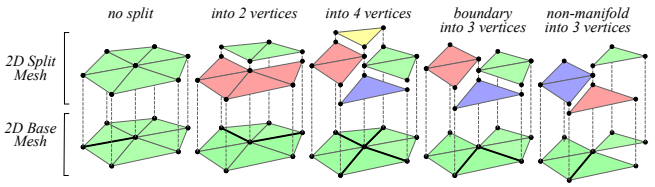

Figure 11: Example cases for splitting (the 1-ring of) a vertex of the 2D base mesh according to its adjacent discontinuity edges (the emphasized black edges). Missing triangles may occur due to their prior removal (Section 5.4.1). Colors distinguish between 1-ring components after the split.

5.4.3. Component analysis

After removing discontinuity triangles around blurry discontinuities and splitting the base mesh along marked edges in sharp discontinuity regions, the mesh may split into several connected components. If a component does not contain at least three points, the reconstruction algorithm can not determine its depth due to lack of connectivity to other components. Therefore, if the set of depth values associated to any triangle of a component is less than three, the component is eliminated prior to reconstruction. After reconstruction, these holes can optionally be filled back in by the hole-filling algorithm proposed shortly.

This stands in contrast to methods that require *each triangle* or superpixel to contain a minimum number of points for plane fitting [19, 10, 68].

5.4.4. Base mesh-driven hole filling

Our discontinuity handling and component analysis eliminate some triangles from the base mesh and split it along discontinuity edges. These changes result in 3D holes after depth optimization. A majority of holes are at discontinuities, nearly parallel to the viewing direction.

Under the assumption of a relief surface without undercut areas, observed from an ortho-view parallel to the base plane of the relief, filling discontinuity holes will reconstruct unobserved surfaces correctly. For example, in 2.5D urban modeling from an aerial ortho-view, walls and other near-vertical surfaces are missing, as they are nearly parallel to the (nadir) viewing direction. In this case, hole filling can reproduce walls and result in a watertight mesh better suited for visualization. In other cases, when either occlusions or true holes occur, hole filling should not be used, as surfaces may get hallucinated.

While it would be possible to use a generic hole filling algorithm [44], we propose a heuristic that complies with our base mesh lifting approach, being driven by the connectivity of the 2D base mesh. Building on the prior knowledge that discontinuity areas contain spurious data points as a result of depth map blur, our hole filling discards the data in these regions. Lhuillier [67, 68] also proposes a 2.5D hole filling, but it requires holes to have coplanar vertices at their boundaries. Instead, we propose a method that fills *any* residual hole, thus resulting in a watertight mesh.

After removing steep areas (Section 5.4.1), splitting discontinuity edges (Section 5.4.2) and connected component filtering (Section 5.4.3) in the 2D base mesh, each 2D base vertex may correspond to zero or more 3D vertices (Figure 11). Denote the known set of 3D vertices obtained from the same 2D base vertex v_i by $\mathcal{U}_i = \{u_{ik}\}$. First, we compute the depths of missing vertices, i.e. the depths of base vertices v_i with $|\mathcal{U}_i| = \emptyset$. Each such vertex obtains a single depth, namely, the average of the depth values of all 3D vertices that are obtained from any of the 1-ring neighbors of v_i in the base mesh. Second, we fill in missing faces, i.e. faces of the 2D base mesh that have been filtered out prior to reconstruction. At this point, each vertex v_i of any missing 2D triangle corresponds to a set of 3D vertices $\mathcal{U}_{ik}$ with $|\mathcal{U}_{ik}| \geq 1$. The task boils down to choose one 3D vertex from the set $\mathcal{U}_{ik}$ for each of the three 2D vertices v_i of the triangle. As the average number of possibilities is rather limited, we exhaustively search through all possible 3D lifting variations of the triangle, and select the minimal-area one.

This reconstructs all vertices and triangles of the base mesh, but holes due to discontinuity edges remain (Figure 11). These holes are not visible from the view, since

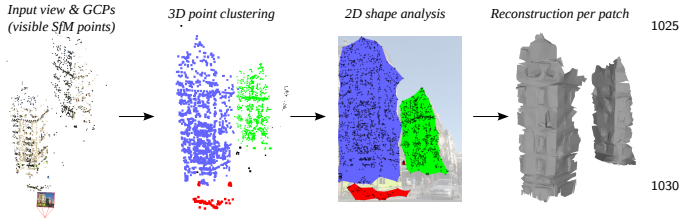

Figure 12: Street-side example: when only sparse depth data is available for view-based meshing, a 3D point clustering and a 2D shape analysis are used to avoid covering major discontinuities and hallucinating surfaces in empty areas (e.g. sky).

they are parallel to the viewing direction. We note that the minimal-area choice for the triangles in the previous step tends to maximize the area of these residual holes, that is, the area of non-slanted walls, in the case of an urban DSM. To fill these, we simply use the filling method discussed in Section 5.2 (see yellow triangles in Figure 6).

5.5. Handling discontinuities in case of sparse GCPs

If only sparse depth information is available, it is difficult if not impossible to reliably and accurately locate discontinuities from a single view, and any mistake leads to unpleasant artifacts in the 3D result. Therefore, we follow a more conservative strategy: we delineate (probably concave) regions of the view that are covered by GCPs in satisfactory density, and reconstruct each patch of the 2D base mesh per delineated region, separately (Figure 12), by the depth optimization of Section 5.3.

In a second step, we segment and eliminate likely residual discontinuities in this initial reconstruction and repeat the reconstruction without these. Next, we discuss details of these two steps.

5.5.1. Point clustering and α-shapes

We start by clustering the subset of 3D points visible in the current view (see Figure 12). Using only the single-view subset highly increases the chance that point clusters on surfaces separated in depth end up in different groups. A region growing in the 3D k-nearest neighbor (k-NN) adjacency graph of the point cloud – with a preference of seeds in denser neighborhoods – proved to be sufficient and fast (we empirically set $k = 30$ and a stop distance of 10 times the median of k-NN distances). This procedure identifies outlier GCPs as unassigned points, which we eliminate before proceeding. We note that other 3D point cloud clustering and outlier filtering strategies could also be used here. For the latter, removing points with less than V_{min} observing views, or if the outlier ratio is roughly known, then ranking the points based on the average distance to their k-NNs and eliminating a low-density fraction of the points, work also well in practice.

Second, we extract the observed 2D shape of each point cluster as the α-shape [93] of the points (see right side of Figure 12). This returns a subset of triangles of the 2D Delaunay-triangulation (2D-DT) over the GCPs. It should be emphasized that this triangulation differs from our 2D base mesh, the latter being defined over image edges or partitions rather than over input points (see Figure 7 for a distinction). For α-shapes, the lowest α value resulting in only manifold vertices is found automatically by trying out increasing values $\{\alpha_i\}$. This guarantees that the outline of each α-shape component can be unambiguously extracted as the longest polygonal boundary. The largest (worst-case) α_i value is set to ∞, which corresponds to the convex hull of the 2D point cluster. As a result, each region sufficiently populated by depth observations is circumscribed by a polygon (Figure 12). These polygons are used to cut out patches of our edge-aligned 2D base mesh, which are then subjected to reconstruction, individually.

5.5.2. Discontinuity-segmentation

The methods in Section 5.5.1 already handle major discontinuities in case of sparse depth input, but surfaces with self-occlusions may survive the procedure in a single cluster. This results in such discontinuities being closed in a consecutive reconstruction step obtained via a watertight 2D base mesh (Section 5.3).

Similar in spirit to the approach described for dense depth maps in Section 5.4.1, we aim to locate discontinuities in this initial surface. However, the sparsity of the data, in this case, does not allow for a robust discontinuity localization at the precision of base mesh edges. Thus, we rather identify discontinuities as groups of faces that are observed under sharp angles (see Figure 13 for an example). This is, again, similar to our filtering of triangles in blurred discontinuity regions in case of dense data (Section 5.4.1).

To obtain a spatial smooth solution, we formulate this problem as a binary segmentation over all F faces $\{f_i\}$ of the initial 3D mesh by minimizing

$$E(\mathcal{L}) = \sum_{i=1}^{F} E_i^{disc}(l_i) + \lambda_{disc} \cdot \sum_{\{f_i,f_j\}} e_{ij}\mathbb{I}[l_i \neq l_j] \qquad (11)$$

over possible labellings $\mathcal{L} = \{l_1, \ldots, l_F\}$, where $l_i \in \{0, 1\}$ is a binary label for face f_i, indicating whether it is a discontinuity. $\varepsilon_{ij} \cdot \mathbb{I}[l_i \neq l_j]$ is a weighted Potts penalty [94], where ε_{ij} is defined as the length of the edge between face f_i and f_j, divided by the circumference of the larger face. The unary term measures the observed angle of a face as $E_i^{disc}(0) = \phi_i$ and $E_i^{disc}(1) = 1 - \phi_i$ with

$$\phi_i = \exp\{-\tfrac{1}{2\sigma^2}(\mathbf{v}_i^T \mathbf{n}_i)^2)\} \qquad (12)$$

where $\mathbf{n}_i$ is the unit-normal of face f_i, and $\mathbf{v}_i$ is the viewing direction of the centroid $\mathbf{c}_i$ of f_i.

We solve the binary problem in Eq. (11) via graph-cuts [95], besides $\lambda_{disc} = 5$, $\sigma = 0.3$ in our experiments.

The segmented discontinuity triangles are removed from the base mesh, and the depths at the remaining vertices are re-optimized using the method of Section 5.3 for each α-shape, separately, to obtain the final mesh (Figure 13).

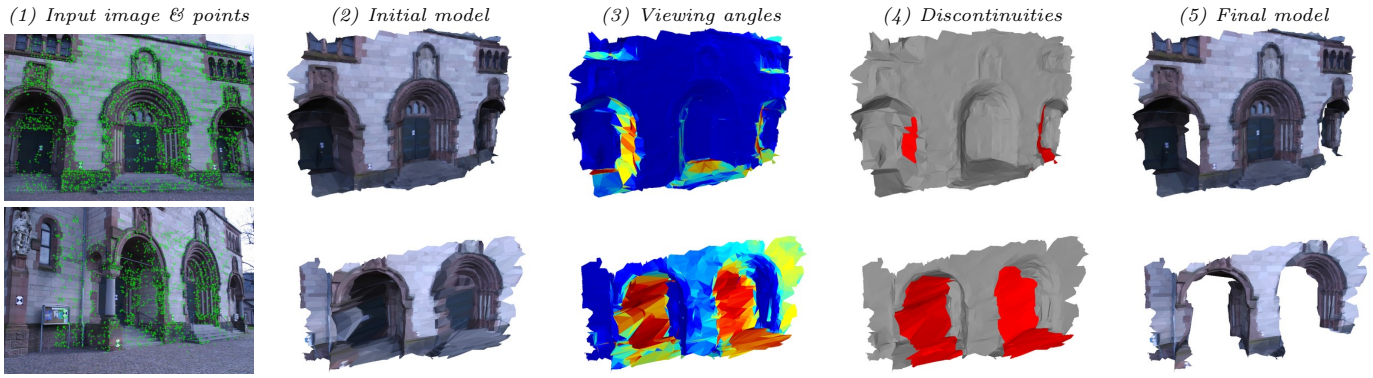

Figure 13: Two examples for discontinuity handling in case of sparse depth information. (1) Input image and GCPs, (2) our initial watertight reconstruction, (3) color-coded viewing angles of mesh faces (red and yellow marks faces observed under sharp angles), (4) graph-cut segmentation of discontinuity regions, (5) final model after refitting. Models are not textured, only colored per face.

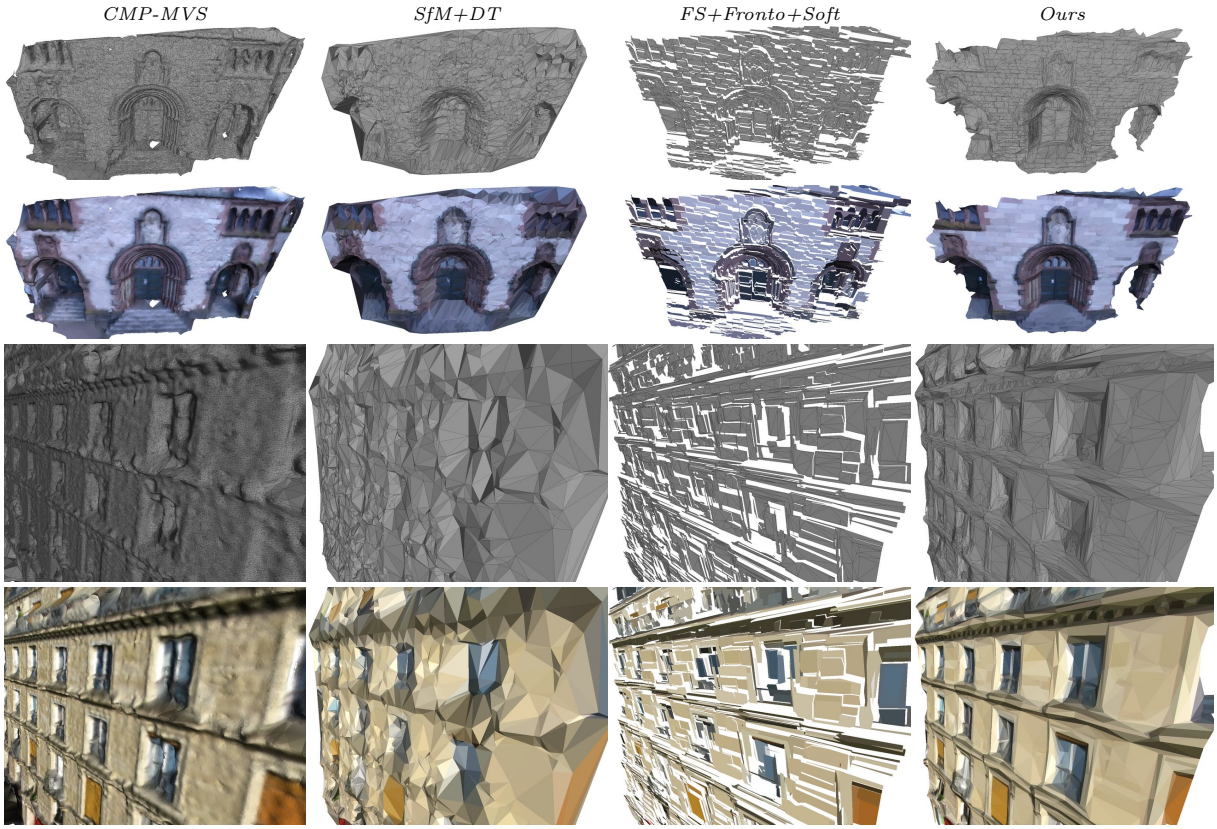

Figure 14: Results for different methods on the Herz-Jesu (top) and Mirbel datasets (bottom). Columns: (1) CMP-MVS mesh, (2) SfM+DT: 2D Delaunay triangulation over SfM points, (3) FS+Fronto+Soft: depths of FS superpixels with fronto-parallel assumption and SfM points as soft-constraints, (4) our method over FS superpixels. Rows: bare geometry and untextured mesh with per-face mean coloring, no texture applied. Notice the level of detail our method can capture from only the sparse SfM data (SfM points are at DT vertices in the 2nd column).

6. Experiments

6.1. Experiments using sparse depth input

We tested our method on landmark scenes (Merton-VI [53], Herz-Jesu [96] and Leuven Castle [31]), our medium-scale street scenes Street Z, Street P and Street M ranging to over 600 images, as well as on relief sequences (Medusa [31], Fountain [96] and our dataset, Dionysos), see Table 1.

The input SfM point cloud has been produced by VisualSfM [97, 98] and SiftGPU [99]. SfM points visible in a chosen perspective view are used as our sparse Ground Control Points (GCPs) with known depth.

6.1.1. Qualitative evaluation

Figure 14 compares our single-view method to the state-of-the-art multi-view stereo tool CMP-MVS [27], and to two baseline methods that estimate a depth map given sparse depths at GCPs in a single view.

The first method (SfM+DT) uses a Delaunay triangu-

			SfM	Ours	PMVS2	CMP-MVS	Input		8×	GPU	1×	
Dataset	Images(±)	Resolution	#pts	#tri/im	#pts	#tri	t_{match}	t_{sfm}	t_{pmvs2}	t_{cmp}	t_{ours}	t_{par}
Street Z	630 (630)	800×1200	238.9k	15.4k	1 620k	3 927k	16051s	207s	1624*	24061s^{+}	1171s	1.9s
Street P	428 (10)	800×1067	365k	13.4k	1 630k	4 697k	710s	811s	1290s*	16143s^{+}	843s	2.0s
Street M	26 (26)	800×1067	19.5k	14.2k	93.3k	1 253k	45s	4s	49s*	1083s^{+}	50.9s	2.0s
Leuven Castle	28 (28)	800×600	16.2k	9.1k	29.6k	586.0k	69s	3s	28s*	825s^{+}	32.3s	1.2s
Medusa	15 (15)	800×640	16.3k	9.4k	30.8k	536.1k	19s	1s	16s*	470s^{+}	18.0s	1.2s
Fountain	11 (11)	1024×683	13.5k	10.1k	44.5k	707.9k	8s	1s	20s*	468s^{+}	16.3s	1.5s
Herz-Jesu	8 (8)	1024×683	8.3k	10.4k	35.2k	492.5k	7s	1s	16s*	334s^{+}	10.6s	1.3s
Dionysos	8 (8)	800×600	4.1k	7.3k	17.4k	243.3k	7s	1s	16s*	229s^{+}	10.0s	1.3s
Merton-VI	6 (6)	1024×768	2.1k	13.3k	10.8k	180.5k	2s	1s	9s*	123s^{+}	9.9s	1.7s

Table 1: Datasets and timings. *8-core parallelism, $^{+}$GPU usage. t_{ours} is given for a single CPU core. ± indicates the number of previous and next images considered for matching. t_{par} is the runtime of our method for a single view (views fully parallellizable after SfM). These results were produced with FS superpixels [92] and a polygon-simplification tolerance of 3 pixels.

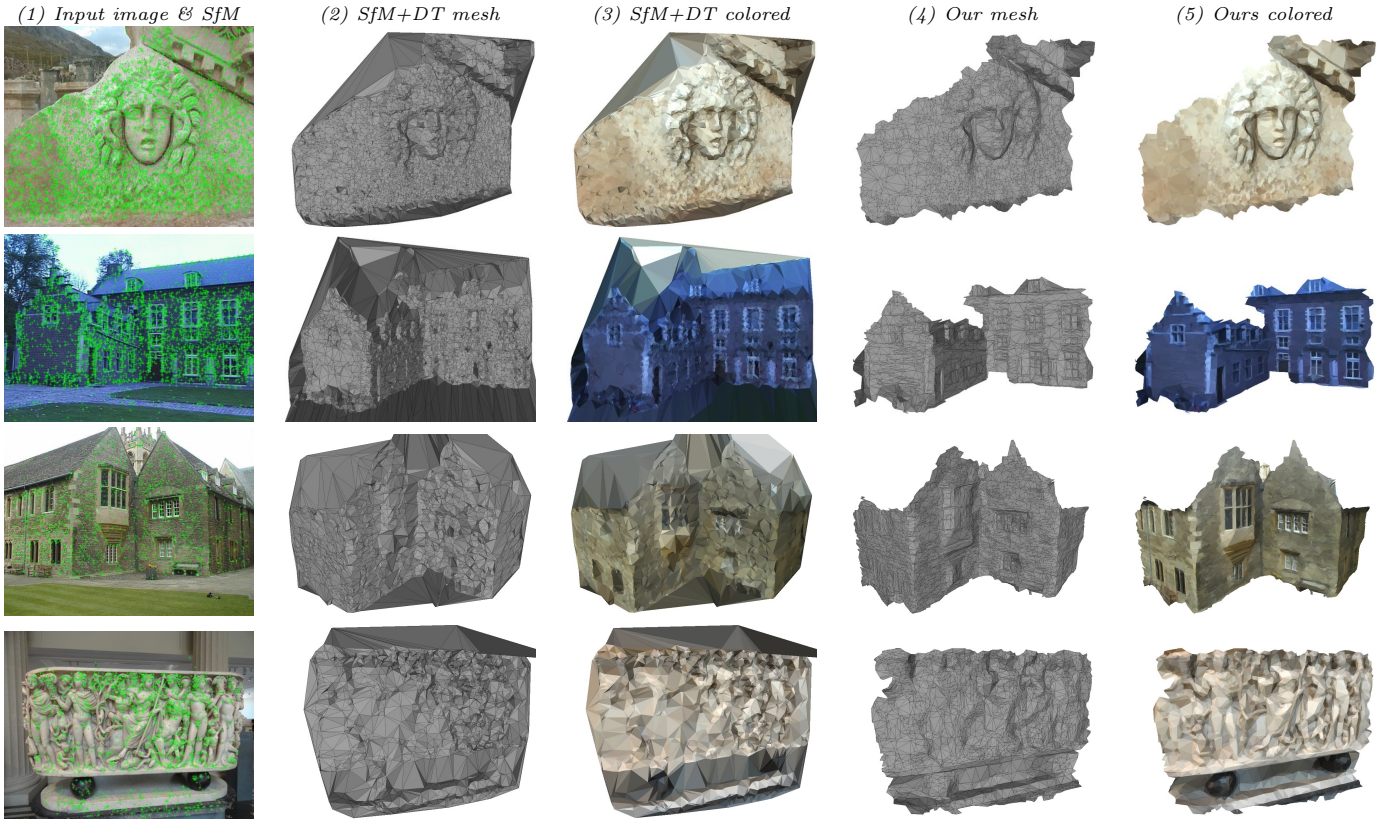

Figure 15: Results for different small and medium-size datasets compared to the SfM+DT baseline. Datasets from top to bottom: Medusa, Leuven Castle, Merton-VI, Dionysos. Columns: (1) input image with visible SfM points overlaid, (2-3) baseline SfM+DT mesh reconstructed from a single view, (4-5) our results using FS superpixels. Models are colored per face, not textured.

lation over 2D observations of SfM points, which is lifted into 3D via the known SfM depths to obtain a 3D mesh (see left side of Figure 7 for a sketch). This is exploited, for example, in [66] for stereo and in [100] for MVS.

The second baseline method (FS+Fronto+Soft) operates over superpixels, assumes a fronto-parallel plane per superpixel, estimates the depth of each plane by using the known GCP depths as soft-constraints while smoothing by penalizing for color-weighted depth differences between superpixels. FS stands for the segmentation method of Felzenszwalb and Huttenlocher [92]. Different aspects of FS+Fronto+Soft are motivated by [22, 72, 73].

For a fair evaluation, all methods are applied after the point clustering proposed in Section 5.5.1, i.e. after removing SfM outliers. Figure 14 demonstrates that meshes of SfM+DT have triangles crossing actual crease edges as the method is not aware of observed edge locations. Also, it is fully susceptible to noise in the SfM points, as it uses them as hard-constraints. SfM+DT has the advantage over FS+Fronto+Soft that it keeps the mesh naturally watertight. The piecewise-continuous assumption of FS+Fronto+Soft over superpixels does not allow for a watertight mesh. This also holds for slanted-plane stereo methods (e.g. [17, 19]). The fronto-parallel assumption [101, 22] only emphasizes the staircasing. In turn, our method respects geometric edges, produces a connected

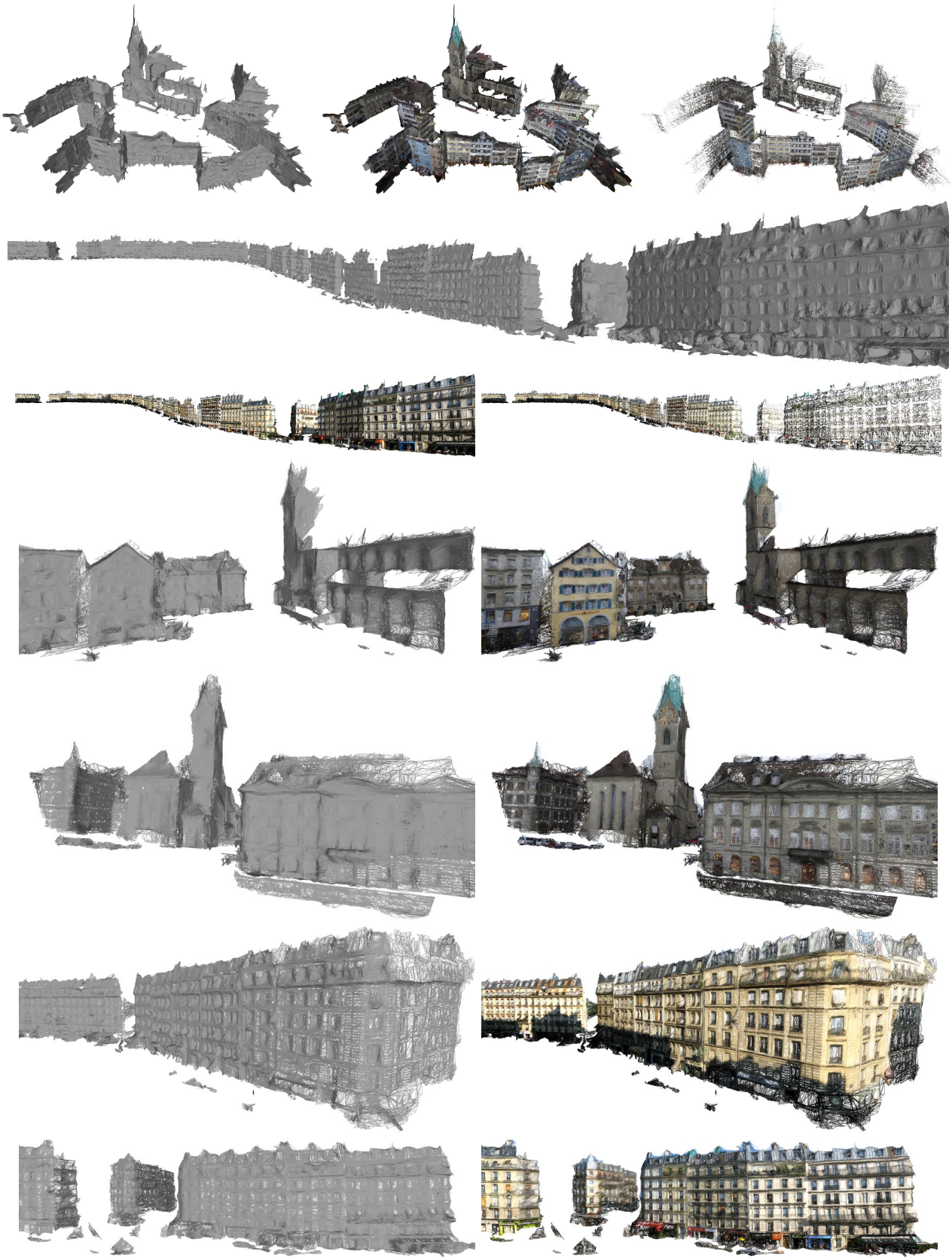

Figure 16: Results from our method applied to every view of larger street-side scenes and sparse SfM data. All single-view meshes are visualized jointly. Rows: (1) Complete Street Z dataset of 630 images and (2-3) complete Street P dataset of 428 images, both shown as a mesh, colored mesh, and cloud of colored vertices. Below are close-up views for Street Z (rows 4-5) and Street P (rows 6-7).

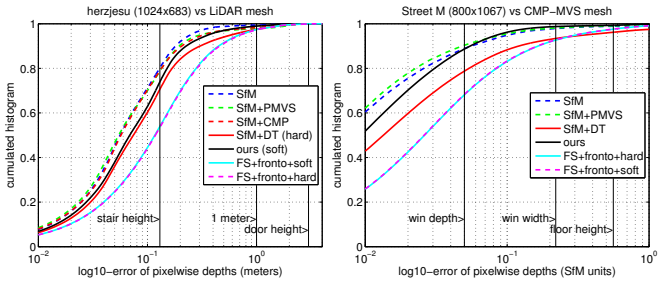

Figure 17: Cumulative histograms of pixelwise depth errors over all images of Herz-Jesu with respect to the LiDAR mesh [96] (left) and Street M with respect to the CMP-MVS [27] mesh (right). Three vertical markers are at the size of a stair, 1 meter and door height for Herz-Jesu, and window depth, window width and floor height for Street M.

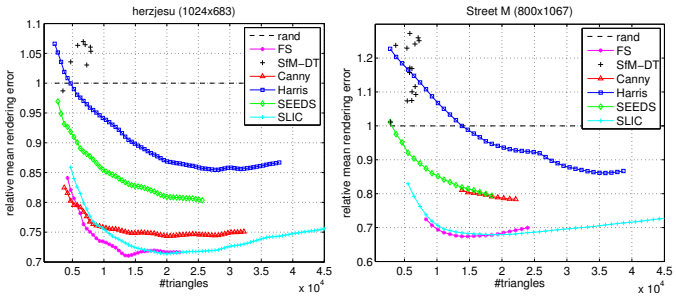

Figure 18: Errors of a per-face flat rendering for different 2D triangulation methods as a function of the triangle count per image, for two different datasets. Errors are relative to those of a triangulation over uniform random points. The curves are averages over all images at each triangle count. For SfM-DT (+), each marker corresponds to the error of a single triangulation per image.

mesh (per α-shape), and the mesh captures the important crease edges of the structure, which are observable as image edges. As a result, triangles cover mostly homogeneous image areas, which enables us to obtain visually pleasing renderings with per-face flat coloring, i.e. without applying a texture.

Further models produced by our method from individual views are shown in Figure 15 in comparison to the SfM+DT baseline, and Figure 16 shows our results for larger-scale street-side scenes. In the latter case, we applied the proposed method on every single view of the sequence (without view selection) and jointly visualized the resulting overlapping meshes.

6.1.2. Evaluation of depth accuracy

We evaluate the accuracy of our meshes with respect to a high-resolution reference mesh $\mathcal{M}^r$. In comparison, we evaluate the methods SfM+DT, FS+Fronto+Soft, CMP-MVS introduced above, and additionally compare to the accuracy of the source SfM point cloud, of the dense point cloud produced by Patch-based Multi-View Stereo (PMVS2) [39], and of the method FS+Fronto+Hard which is similar to FS+Fronto+Soft but applies the known sparse point depths as hard constraints (motivated by [71]).

The discrepancy between a mesh $\mathcal{M}$ and the reference mesh $\mathcal{M}^r$ is measured by comparing the depth maps rendered for the two meshes from the same viewpoint i, knowing the camera matrix $\mathbf{P}_i^r$. In the same vein, the depths of points with respect to view i are compared to the depth maps of $\mathcal{M}^r$ in the views according to the visibility information from SfM/PMVS2. We collect the pixelwise/pointwise absolute depth errors for all views of a dataset, and compute the cumulative density function of the errors.

Figure 17 shows the error curves for the Herz-Jesu and Street M datasets. For Street M, we use the high-resolution CMP-MVS mesh as reference mesh $\mathcal{M}^r$, whereas for Herz-Jesu, the publicly available LiDAR mesh [96] is used as ground-truth. To compare the reconstructed meshes given in the SfM frame to the LiDAR mesh given in a metric frame, a precise registration procedure is carried out between the frames via the known SfM points, ground-truth cameras and depth maps.

In Figure 17, vertical markers are placed at meaningful scales, e.g. stairstep/door/floor height. The higher the curve of a method, the higher the precision. The plots consistently show that state-of-the-art dense-MVS methods (CMP-MVS, PMVS2) provide the highest precision. The proposed single-view surface fitting method (black curve) has a higher accuracy than all tested single-view baseline methods (SfM+DT, FS+Fronto+Soft, FS+Fronto+Hard), and has an accuracy comparable to dense-MVS methods at the scales particularly interesting for city scenes (e.g. starting around the depth of a window or the height of a stair).

6.1.3. Evaluation of rendering quality

In Section 4.1, we present different ways to obtain a 2D base mesh from an image. To evaluate these approaches, we rasterize the 2D mesh with mean image colors per triangle – which is equivalent to rendering the 3D mesh from the particular viewpoint with per-face flat coloring – and compare the rasterized image to the original one. We measure the discrepancy by averaging absolute differences over color channels, pixels and all images of the dataset. This rendering quality implicitly indicates how well the decomposition is aligned with image edges, e.g. for semantic classification or visualization.

Figure 18 compares Delaunay triangulations over Harris corners [102], polygonized Canny edges [103], and polygonized FS [92], SEEDS [104] and SLIC [105] superpixel boundaries, for the Herz-Jesu and Street M datasets. For each method, we vary its core parameter (e.g. Canny threshold) to obtain different triangle counts. As a baseline, we evaluated triangulations on top of n uniform random points over the image (*rand* method), and report all accuracies relative that of *rand*. For comparison, we also show the result of SfM+DT introduced earlier, i.e. a triangulation over projected SfM points (originating from SIFT points in VisualSFM [99]). We conclude from Figure 18, that FS and SLIC superpixels (slightly slower) approximate the image best at any number of triangles, while

Harris corners and the SfM+DT give the lowest accuracy. The fastest methods, SEEDS and Canny reside in between.

6.1.4. Timings

Particularly positive aspects of our method are its speed and potential for parallellization. Table 1 shows timings for different datasets processed on a single 3.4 GHz CPU core in Matlab, where a majority of low-level tasks are implemented as C++ MEX. After SfM, Street Z (the largest dataset of 630 images of around 1 MPixel) was reconstructed in 6.7 hours using CMP-MVS on a GeForce GTX 660 GPU and 28 minutes (14.4× faster) using our approach on a single CPU core. PMVS2 returns a good-quality semi-dense point cloud in only 27 minutes but using 8-core parallellization, and it outputs a point cloud that requires further meshing to obtain a surface model.

Although the SfM pipeline provides the input for our method, it is interesting to take the time for keypoint matching and SfM also into consideration (see Table 1). Street Z was reconstructed in 11.2 hours in total using CMP-MVS, and 4.8 hours using our method (2.3× faster), while Street P in 4.9 hours using CMP-MVS, and 39 minutes using our method (7.5× faster). The time spent on matching depends heavily on the matching scheme applied, however. To demonstrate this, all images are matched against all for Street Z, and each image to ±10 images in sequence for Street P. We note that vocabulary trees, other SfM pipelines, or real-time SLAM pipelines could also be used to produce our input faster.

Our algorithm spends around 0.8-3 seconds per 1 MPixel image. The majority is spent on 2D base mesh extraction, and it depends on the method and, for example, the superpixel implementation chosen. The actual vertex depth reconstruction (with discontinuity segmentation) solves in less than 0.3 seconds per view in Matlab for around 15-20k unknown depth values. Moreover, our method is fully parallellizable per image (see the last column of Table 1).

6.2. Experiments using dense height maps

Besides testing our approach on sparse street-side SfM datasets, we evaluate the method on dense depth data. Since our focus is city modeling, we have chosen to use urban Digital Surface Models (DSM) for this purpose, obtained by Multi-View Stereo (MVS) techniques from airborne imagery, and our goal here is to compute geometrically accurate but considerably compact 3D mesh model of a large-scale urban area.

6.2.1. Input data

The RGB images used for generating the input DSM have 15 cm ground resolution, and were taken in nadir direction from a platform flying at an altitude of around 3 km. A dense DSM and an ortho-image were obtained by using RealityCapture[3] over a 9 km^2 area of the city of Zürich in Switzerland, over a raster grid of 25 cm resolution. The DSM is split into 56 non-overlapping tiles, each of a resolution of 1600×1600 pixels (covering an area of 400×400 m^2, each).

6.2.2. Evaluation method

We applied our pipeline to each of the 56 DSM tiles, which includes piecewise-planar partitioning, base mesh extraction, discontinuity handling and depth optimization (we refer to the bottom of Figure 2 for an overview). In addition, we apply our hole filling algorithm described in Section 5.4.4 to produce vertical building walls, and a watertight mesh. An overview of the default parameters of our method for dense depth data is given in Table 2. Parameters were fixed empirically, which proved to be stable enough to use them throughout our experiments.

Parameter	Section	Symbol	Default
Plane growing distance tolerance	4.3	δ	0.2 m
Plane growing angle tolerance	4.3	θ	20°
Plane refit size factor	4.3	κ	1.5
Plane merging error tolerance	4.4	ϵ	1.0 m
Polygon simplification tolerance	4.2	dp	2 px
Discontinuity triangle threshold	5.4.1	θ_{disc}	75°
Discontinuity edge threshold	5.4.1	ΔD_{disc}	1.0 m

Table 2: Core parameters of our pipeline for dense depth input, and the section where each of these are introduced.

For each tile, we evaluate the compactness and the accuracy of the mesh resulting from our pipeline. We define the following measures:

- *Compactness*: the ratio between the number of input points and the number of output mesh vertices.
- *3D approximation error*: we measure the 3D distances from the input points corresponding to DSM pixels to the nearest point on the output mesh (via the AABB-tree implementation of CGAL [106]), and report the mean distance. For efficiency, we only evaluate over a random sample of 10^5 input points.
- *Bad area ratio*: the output mesh is rendered as a height map that is compared to the input height map by taking pixel-wise absolute differences (measuring errors along the gravity vector), and the ratio of pixels with an error larger than a threshold (25 cm) is reported.

Since the DSM is generated from top views, walls are captured as blurry discontinuities, i.e. spurious surfaces with incorrect input. For a fair evaluation, we detect these regions by computing normals in 3×3 pixel neighborhoods over the DSM, and only evaluate 3D approximation error and bad area ratio over points with normals that do not deviate more than 70° from the vertical direction (this excludes around 5% of the input points).

We also measure the accuracy of our intermediate plane arrangements after plane growing and merging (Algorithms 1 and 2). More precisely, the mean of the 3D distances from

[3] https://www.capturingreality.com/

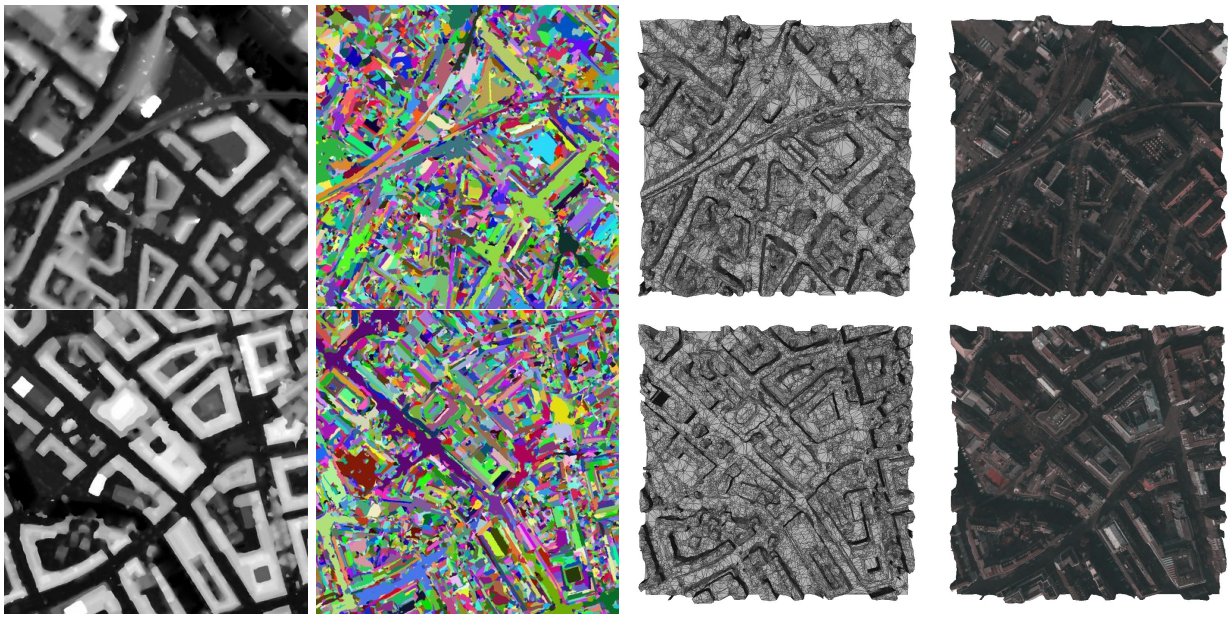

Figure 19: Our qualitative results over two out of the five test DSM tiles reported in Table 3, namely, Zurich 001 (top) and 022 (bottom). From left to right: (1) input DSM tile of 1600×1600 pixels, (2) view decomposition after plane growing and merging, (3) our resulting mesh, (4) our mesh textured. The parameter values in Table 2 were used to produce these results. See also Figure 20 for a closer look.

	planes				2D	3D	3D	filled	overall		
Dataset	#grown	error	#final	error	#ver	#ver	#tri	%tri	comp	error	runtime
Zurich 001	76.5k	0.056	6.2k	0.151	25.8k	28.4k	56.1k	14.1%	90.1	0.092	11.3s
Zurich 004	97.1k	0.054	8.0k	0.155	30.3k	34.4k	67.9k	19.2%	74.4	0.091	18.5s
Zurich 021	98.6k	0.054	8.2k	0.153	30.5k	34.7k	68.6k	19.4%	73.8	0.093	9.7s
Zurich 022	92.7k	0.055	7.3k	0.158	28.0k	32.0k	63.1k	18.8%	80.0	0.092	10.8s
Zurich 036	93.9k	0.055	7.4k	0.163	28.2k	32.1k	63.4k	17.5%	90.8	0.095	11.5s
Zurich 56 tiles	5.10M	0.055	423k	0.154	1.61M	1.82M	3.58M	16.9%	78.9	0.092	11.7m

Table 3: Results of our algorithm on five 2.56 MPixel urban DSMs, each covering an area of 400×400 m^2 (see Figure 19), and on all 56 tiles covering a 9 km^2 area. The parameter values in Table 2 were used. Mean 3D approximation errors of the input 2.5D point cloud by the plane arrangements and by the final watertight mesh are given in meters. The number of final vertices divided by the number of input points is reported as compression factor (comp). #grown and #final are the number of planes after plane growing (Alg. 1) and merging (Alg. 2).

each input point is evaluated to its associated plane (see association procedure in Section 5.1). Points with no valid plane assignments are skipped.

6.2.3. Results

A bird's eye view of our resulting meshes for two tiles are shown in Figure 19, while close-up views of some individual buildings are shown in Figure 20. It can be seen from the figures that crease edges of the DSM are preserved in our meshes, and that the size of the triangles adapt to the planar regions (large triangles in large flat areas and dense sampling in non-flat regions). One can also observe sharp discontinuities around walls.

The close-up views of Figure 20 show how compact our resulting mesh is compared to the dense 2.5D input point cloud, while it captures important structures.

Our numerical results for the same data are summarized in Table 3. Our method achieves a compression rate of around 80, and around 10 cm mean accuracy over all tested tiles in 13 seconds per tile on average, when using the parameter settings of Table 2. All 56 tiles covering an area of 9 km^2 at 25-cm resolution (143M input points) are processed in 11.7 minutes.

Most of the processing time (around 50%) is spent on plane merging, the rest is spent on computing normals and curvatures (9%), plane growing (15%), base mesh extraction (5%), and reconstruction (14%).

Table 3 also shows that the price paid for the compaction in the plane merging step is that the initial error of the grown plane arrangement increases from 5 cm to around 15 cm on average (besides the error tolerance parameter of $\epsilon = 1$ m). An alternative way to reconstruct a plane arrangement as a watertight model in 3D is to use a polyhedral cell complex (as in [29] or [90]). In such an approach, the accuracy of the plane arrangement would roughly reflect the final accuracy. In contrast, our relaxation of the strict planarity criterion to a triangulation allows us to reduce the error again (to around 10 cm in the example runs). In our experience, the planarity of each identified flat region is violated very slightly in our meshes, just as much as is necessary for watertightness.

The three different prioritization strategies used in the

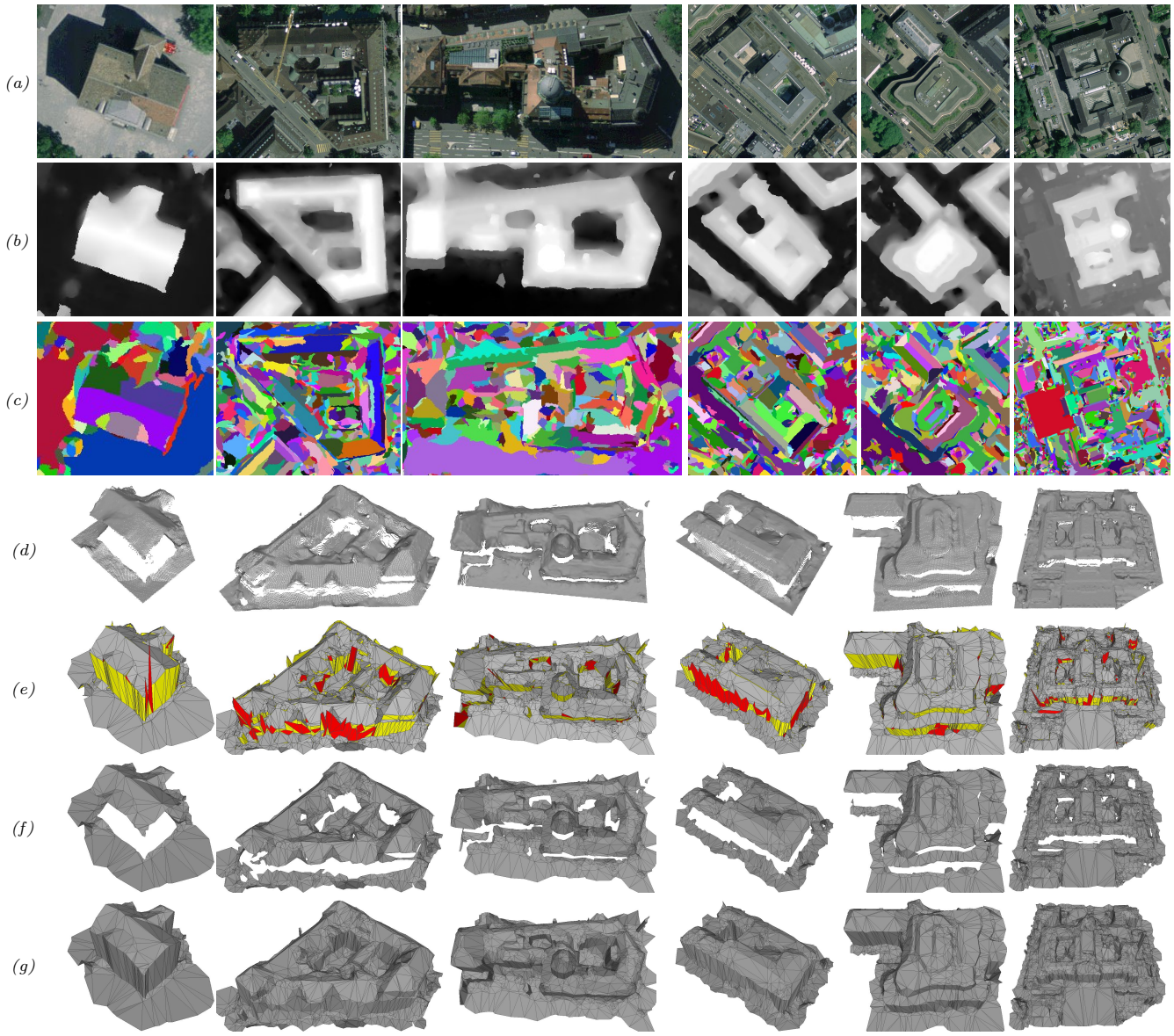

Figure 20: Close-up views of some buildings in our results. Rows: (a) an ortho-image for illustration (not used as input), (b) input DSM, (c) view partitioning after plane growing and merging, (d) the input DSM visualized as a 2.5D point cloud (notice the density and redundancy of it), (e) initial mesh obtained by projecting each base mesh triangle to its associated plane (inserted vertical closures are yellow, and detected overshoots are in red), (f) result of depth optimization, (g) our final watertight mesh after applying the proposed 2.5D hole filling.

	plane merging				2D	3D model	
Dataset	prio type	#planes	error	time	#ver	#ver	error
Zurich 001	Dihedral	6246	0.151	6.6s	**25.8k**	**28.4k**	**0.0913**
	MinError	**6105**	0.148	6.4s	26.4k	29.1k	0.0935
	AreaRatio	6284	**0.138**	**3.8s**	28.2k	31.3k	0.0920
Zurich 022	Dihedral	7328	0.158	6.3s	**28.0k**	**32.0k**	**0.0917**
	MinError	**7243**	0.153	7.1s	28.6k	32.8k	0.0945
	AreaRatio	7572	**0.142**	**4.1s**	30.8k	35.5k	0.0922

Table 4: A comparison of different prioritization strategies in plane merging (see Algorithm 2 in Section 4.4). The best value is highlighted separately for each dataset.

plane merging algorithm (Section 4.4) are compared in Table 4 on two tiles, using the parameters of Table 2. The ranking based on the ratio of areas (AreaRatio) gives the least error of the plane arrangement. Prioritization by the least error increase (MinError) gives the least amount of planes. However, the dihedral angle criterion slightly outperforms the other two, both in terms of approximation quality and compactness of the final 3D model. This can be explained by the more compact regions (consisting of less triangles), whose boundaries capture 3D edge features better. Thus, we used the dihedral angle criterion to produce all our results (Table 3).

6.2.4. Comparison to other methods

To evaluate the range of compactness and accuracy that can be achieved by our method, we analyzed the effect of parameter variations to these characteristics. We

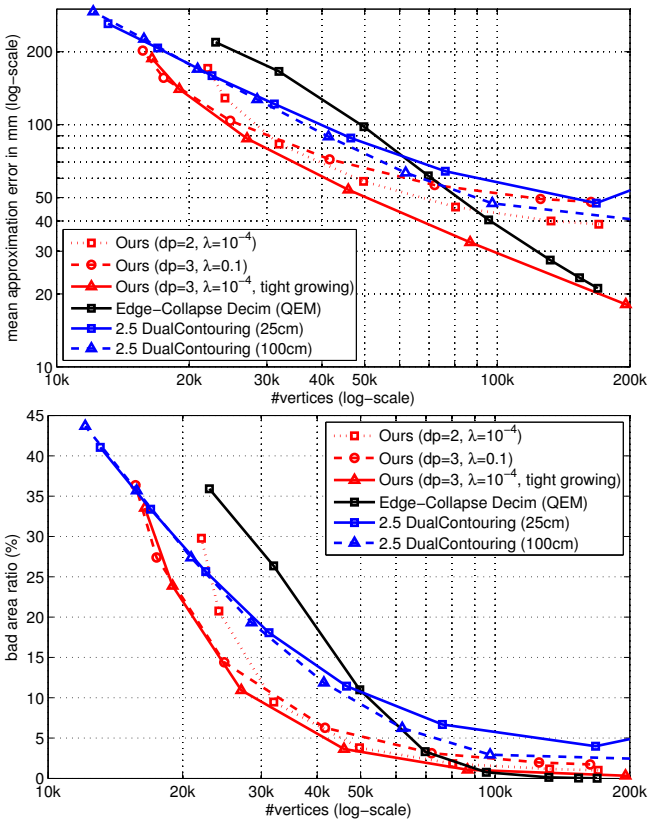

Figure 21: Evaluation of the ranges of model compactness vs. accuracy by varying parameters of our method, and comparing the results to other methods: QEM Mesh Decimation, and 2.5D Dual Contouring [87]. Accuracy is measured as 3D approximation error (top) and bad area ratio with an error threshold of 25 cm (bottom), both defined in Section 6.2.2.

vary the merging tolerance ϵ (it takes the values 0, 0.1, 0.25, 0.5, 1.0 and 2.0 meters), while keeping other parameters fixed to their default values in Table 2, unless stated otherwise. In addition, we compared the resulting characteristics to those obtained from 2.5D Dual Contouring [87] and from edge-collapse mesh decimation via a Quadratic Error Measure (QEM) [46] by using the implementation in the VCG library distributed with Meshlab [107].

We performed the evaluations on the DSM tile Zurich 022 shown in Figure 19 and listed in Table 3. We generated a dense pixel-wise mesh from the DSM by connecting adjacent vertices, fed this to the baseline QEM Mesh Decimation algorithm, and set it to obtain roughly similar compression as our method. In the case of 2.5D Dual Contouring we obtained different curves by varying its grid size parameter (ranging from 5 cm to 3 m) at fixed values of its clustering threshold parameter (ranging from 5 cm to 1 m). For clarity, we only show two of the Dual Contouring curves which are the most competitive in comparison to our method.

Results are shown in Figure 21. The lower the curve the better. Although QEM Mesh Decimation (black curve) is very accurate when the vertex counts are higher (over 100k vertices), our method outperforms both QEM Mesh Decimation and 2.5D Dual Contouring in the vertex count range 15k-100k. This range corresponds to more extreme simplifications (compression rates from 25.6 to 170), which is particularly interesting to obtain compact large-scale models. If less vertices than 15k are used, the approximation becomes poor with all methods, resulting in a bad area ratio of over 35%.

Our method outperforms both baselines over the full range of interest in terms of 3D approximation error, when the smoothness parameter value is low ($\lambda = 10^{-4}$) and the initial plane growing parameters are tighter (the default values of δ and θ in Table 2 were reduced from 20 cm and 20° to 5 cm and 5°, respectively). This comes at a slightly higher toll on the processing time however, which increases to around 30 seconds from the average of 13 sec per tile shown in Table 3. In comparison, mesh decimation runs between 60-70 seconds, and Dual Contouring can be also as fast as 30 seconds. All our tests were performed on a 3.4 GHz desktop CPU core with a Matlab implementation with a majority of low-level tasks implemented in C++.

7. Conclusions

7.1. Discussion and limitations

Our view partitioning strategies are based either on a single image or on a single dense depth map. A combination of these two is left for future work. In our experience, the SfM point cloud is not sufficiently dense to extract roof shapes at the desired LoD in our aerial nadir scenario. In turn, a dense height map is enough in itself to accomplish the task, without the need to rely on an additional ortho-image. Unfortunately, the nadir source images often contain strong shadows and little or no evidence along walls, which are then easy to miss by an image partitioning scheme. In turn, volumetric MVS surface reconstruction smoothly closes roofs to ground in these regions, producing direct 3D evidence (a variation in height and normal). This finally allows our piecewise-planar height map decomposition to capture walls as region boundaries.

Our image decomposition approach assumes that the surface of interest can be captured by image partitions, and is sampled densely enough to interpolate the surface at the desired level of detail. Narrow occluding objects and areas with very low point density are out of scope and are filtered out in our point cloud clustering and outlier filtering step.

If the input urban height map contains the typical artifact of slanted walls due to blur, our approach approximates but does not *repair* these areas, as high-level semantic information may be needed to do so. Also, preliminary outlier filtering and hole filling (e.g. multi-view depth map fusion) might be necessary for very noisy or semi-dense depth maps. Otherwise an excessive amount of missing pixels may prevent locally planar areas from becoming 4-pixel-adjacent during the plane extraction procedure.

The runtime of the recursive merging procedure depends on content: it becomes more expensive when many

regions need to be merged. This occurs when large and relatively flat areas are oversegmented by a too conservative region growing. For example, one of the 56 tiles of Zurich is processed in 40 seconds instead of the average 13 seconds, using the settings in Table 2, due to a huge flat area covered by rails behind the central station. A solution is to prevent over-segmentation by relaxing the plane growing criteria.

The proposed discontinuity handling copes with discontinuities efficiently. Yet, very peaky towers may be confused with non-vertical walls of comparable slopes due to input blur, and are eliminated or shortened in our output (see our angle threshold θ_{disc}), which could be prevented by an explicit detection of such structures or by using better-quality input with less blur in wall regions.

Our method *fits* surface meshes to the input data. Unlike MVS, it is not suitable to extend the reconstructed surface beyond the area sampled by input SfM points. For example, the ground is not recovered from street-side imagery if SfM does not provide input points there.

In our street-side experiments, the proposed method has been applied to every image. This is redundant, since the same input SfM point is observed by 3.9 (Street Z) and 4.5 (Street P) views on average. Additional speed-up could be obtained by a preliminary view selection [108]. We found that our meshes are highly consistent, and a view selection with mesh trimming may be sufficient for some applications. In the case of an aerial height map, the resulting mesh tiles are non-overlapping, and have minor discrepancy at tile boundaries, due to the dense input.

Finally, this paper did not address the problem of mesh fusion. Existing volumetric fusion methods remesh the input and, hence, could ruin the edge-alignment and compactness properties of our meshes. If these properties can be sacrificed for the sake of obtaining a single fused mesh, we found that Poisson surface reconstruction [69] works well over a dense oriented point cloud sampled from our meshes (while it produces artifacts or an oversmoothed result on our inhomogeneous input SfM point cloud).

7.2. Summary

In this work, we propose an efficient approach for building compact, edge-preserving, view-centric triangle meshes if either dense or sparse depth data is available. The presented approach is data-driven (generic) which allows it to capture complex roof and facade structures in an urban scenario. The method consists of view partitioning, base mesh extraction, occlusion handling, vertex depth optimization (mesh fitting), and an optional hole filling. Different view partitioning schemes are presented for images and dense depth maps. These ensure that edges of the 2D base mesh are aligned with crease edges and discontinuities, with triangles covering low-variation or flat areas. Discontinuities are handled prior to final reconstruction by segmenting discontinuity areas where the data is sparse or blurred, and by splitting the 2D base mesh along discontinuity edges where the data is dense and sufficiently sharp. The proposed depth optimization respects these discontinuities, exploits a minimum-curvature smoothness prior in a formulation that allows for a fast linear solver, handles weakly textured areas, and preserves crease edges.

The method is evaluated in two largely different scenarios of 3D urban modeling, namely, on street-side images with an underlying sparse SfM point cloud and on large-scale dense urban height maps. However, the approach is not restricted to these, and oblique aerial or street-side LiDAR/MVS depth-maps, or indoor scenes would also be interesting to try in the future. Combining data from both aerial and street-side acquisition in order to get more complete 3D city models is an exciting future direction, as well. The proposed overarching methodology is in line with this objective, as it works well on both kinds of input.

Our experimental results show an excellent trade-off between quality and compactness. Runtime on a single CPU core is 1-2 seconds per image in case of SfM data and 1.3 minutes per km^2 for urban height maps. For the latter, a data compression factor of around 80 could be achieved, which results in a 3D city model of 9 km^2 in a mere 65 megabytes.

Acknowledgments. This work was supported by the European Research Council (ERC) project VarCity (#273940). We thank Prof. Konrad Schindler and the Photogrammetry and Remote Sensing Group at ETH Zurich, Switzerland, for providing the Zürich aerial dataset.

References

[1] N. Haala, M. Kada, An update on automatic 3D building reconstruction, ISPRS Journal of Photogrammetry and Remote Sensing 65 (6) (2010) 570–580.

[2] P. Musialski, P. Wonka, D. G. Aliaga, M. Wimmer, L. Van Gool, W. Purgathofer, A survey of urban reconstruction, in: EUROGRAPHICS, 2012.

[3] F. Leberl, A. Irschara, T. Pock, P. Meixner, M. Gruber, S. Scholz, A. Wiechert, Point Clouds: Lidar versus 3D Vision, Photogrammetric Engineering and Remote Sensing 76 (10) (2010) 1123–1134.

[4] Y. Furukawa, B. Curless, S. Seitz, R. Szeliski, Manhattan-world stereo, in: CVPR, 2009.

[5] N. Cornelis, B. Leibe, K. Cornelis, L. Van Gool, 3D urban scene modeling integrating recognition and reconstruction, in: IJCV, Vol. 78, 2008, pp. 121–141.

[6] M. Pollefeys, D. Nistér, J. M. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S. J. Kim, P. Merrell, C. Salmi, S. Sinha, B. Talton, L. Wang, Q. Yang, H. Stewénius, R. Yang, G. Welch, H. Towles, Detailed real-time urban 3D reconstruction from video, IJCV 78 (2-3) (2008) 143–167.

[7] J. Xiao, T. Fang, P. Tan, P. Zhao, E. Ofek, L. Quan, Image-based facade modeling, in: SIGGRAPH Asia, 2008.

[8] S. Sinha, D. Steedly., R. Szeliski, Piecewise Planar Stereo for Image-based Rendering, in: ICCV, 2009.

[9] M. Lhuillier, S. Yu, Manifold surface reconstruction of an environment from sparse structure-from-motion data, CVIU 117 (11) (2013) 1628 – 1644.

[10] A. Bódis-Szomorú, H. Riemenschneider, L. Van Gool, Fast, Approximate Piecewise-Planar Modeling Based on Sparse Structure-from-Motion and Superpixels, in: CVPR, 2014.

[11] H. Riemenschneider, U. Krispel, W. Thaller, M. Donoser, S. Havemann, D. Fellner, H. Bischof, Irregular lattices for complex shape grammar facade parsing, in: CVPR, 2012.

[12] A. Martinovic, M. Mathias, J. Weissenberg, L. Van Gool, A Three-Layered Approach to Facade Parsing , in: ECCV, 2012

[13] A. Martinović, J. Knopp, H. Riemenschneider, L. Van Gool, 3D all the way: Semantic segmentation of urban scenes from start to end in 3D, in: CVPR, 2015.

[14] M. Koziński, R. Gadde, S. Zagoruyko, G. Obozinski, R. Marlet, A MRF shape prior for facade parsing with occlusions, in CVPR, 2015.

[15] P. Müller, G. Zeng, P. Wonka, L. V. Gool, Image-based procedural modeling of facades, ACM Graphics 26 (3) (2007) 85.

[16] A. Bódis-Szomorú, H. Riemenschneider, L. V. Gool, Superpixel meshes for fast edge-preserving surface reconstruction, in: CVPR, 2015.

[17] Z.-F. Wang, Z.-G. Zheng, A region based stereo matching algorithm using cooperative optimization, in: CVPR, 2008, pp. 1–8.

[18] Y. Taguchi, B. Wilburn, C. Zitnick, Stereo reconstruction with mixed pixels using adaptive over-segmentation, in: CVPR, 2008.

[19] Q. Yang, L. Wang, R. Yang, H. Stewenius, D. Nister, Stereo matching with color-weighted correlation, hierarchical belief propagation, and occlusion handling, PAMI 31 (3) (2009) 492–504.

[20] D. Scharstein, R. Szeliski, A taxonomy and evaluation of dense two-frame stereo correspondence algorithms, IJCV 47 (1/2/3) (2002) 7–42.

[21] M. Brown, D. Burschka, G. Hager, Advances in computational stereo, PAMI 25 (8) (2003) 993–1008.

[22] C. Zitnick, S. Kang, Stereo for image-based rendering using image over-segmentation, IJCV 75 (1) (2007) 49–65.

[23] H. Tao, H. Sawhney, R. Kumar, A global matching framework for stereo computation, in: ICCV, 2001.

[24] S. Seitz, B. Curless, J. Diebel, D. Scharstein, R. Szeliski, A comparison and evaluation of multi-view stereo reconstruction algorithms, in: CVPR, 2006.

[25] G. Vogiatzis, C. Hernandez, P. Torr, R. Cipolla, Multi-view stereo via volumetric graph-cuts and occlusion robust photo-consistency, PAMI 29 (12) (2007) 2241–2246.

[26] I. Garcia-Dorado, I. Demir, D. G. Aliaga, Automatic urban modeling using volumetric reconstruction with surface graph cuts, Computers & Graphics 37 (7) (2013) 896–910.

[27] M. Jancosek, T. Pajdla, Multi-view reconstruction preserving weakly-supported surfaces, in: CVPR, 2011.

[28] H.-H. Vu, P. Labatut, J. Pons, R. Keriven, High Accuracy and Visibility-Consistent Dense Multi-view Stereo, PAMI 34 (5) (2012) 889–901.

[29] A. Chauve, P. Labatut, J. Pons, Robust piecewise-planar 3D reconstruction and completion from large-scale unstructured point data, in: CVPR, 2010.

[30] P. Labatut, J. Pons, R. Keriven, Efficient multi-view reconstruction of large-scale scenes using interest points, Delaunay triangulation and graph cuts, in: ICCV, 2007.

[31] M. Pollefeys, L. Van Gool, M. Vergauwen, F. Verbiest, K. Cornelis, J. Tops, R. Koch, Visual modeling with a hand-held camera, IJCV 59 (3) (2004) 207–232.

[32] B. Micusik, J. Kosecka, Multi-view superpixel stereo in urban environments, IJCV 89 (1) (2010) 106–119.

[33] R. A. Newcombe, A. J. Davison, Live dense reconstruction with a single moving camera, in: CVPR, 2010.

[34] S. J. L. Richard A. Newcombe, A. J. Davison, DTAM: Dense Tracking And Mapping in real-time, in: ICCV, 2011.

[35] M. Pizzoli, C. Forster, D. Scaramuzza, REMODE: Probabilistic, monocular dense reconstruction in real time, in: ICRA, 2014.

[36] A. Geiger, J. Ziegler, C. Stiller, Stereoscan: Dense 3D reconstruction in real-time, in: Intelligent Vehicles Symposium (IV), 2011.

[37] P. Tanskanen, K. Kolev, L. Meier, F. Camposeco, O. Saurer, M. Pollefeys, Live metric 3D reconstruction on mobile phones, in: ICCV, 2013, pp. 65–72.

[38] M. Habbecke, L. Kobbelt, A surface-growing approach to multi-view stereo reconstruction, in: CVPR, 2007.

[39] Y. Furukawa, J. Ponce, Accurate, Dense, and Robust Multi-View Stereopsis, PAMI 32 (8) (2010) 1362–1376.

[40] E. Tola, C. Strecha, P. Fua, Efficient large scale multi-view stereo for ultra high resolution image sets, Machine Vision and Applications 23 (5) (2012) 903–920.

[41] G. Vogiatzis, P. Torr, R. Cipolla, Bayesian stochastic mesh optimisation for 3D reconstruction, in: BMVC, 2003.

[42] A. Delaunoy, M. Pollefeys, Photometric bundle adjustment for dense multi-view 3D modeling, in: CVPR, 2014.

[43] J. Rossignac, P. Borrel, Multi-resolution 3D approximations for rendering complex scenes, in: Modeling in Computer Graphics, Springer Berlin Heidelberg, 1993, pp. 455–465.

[44] M. Botsch, P. Alliez, L. Kobbelt, M. Pauly, B. Lévy, Polygon mesh processing, A K Peters, 2010.

[45] H. Hoppe, Progressive meshes, in: SIGGRAPH, 1996, pp. 99–108.

[46] M. Garland, P. S. Heckbert, Surface simplification using quadric error metrics, in: SIGGRAPH, 1997, pp. 209–216.

[47] P. Alliez, N. Laurent, H. Sanson, F. Schmitt, Mesh approximation using a volume-based metric, in: Seventh Pacific Conference on Computer Graphics and Applications, 1999, pp. 292–301.

[48] D. Salinas, F. Lafarge, P. Alliez, Structure-aware mesh decimation 34 (6) (2015) 211–227.

[49] D. Cohen-Steiner, P. Alliez, M. Desbrun, Variational shape approximation, in: SIGGRAPH, 2004, pp. 905–914.

[50] F. Lafarge, R. Keriven, M. Bredif, Insertion of 3D-primitives in mesh-based representations: Towards compact models preserving the details, IEEE Trans. on Image Processing 19 (7) (2010) 1683–1694.

[51] D. Gallup, J. Frahm, M. Pollefeys, Piecewise planar and non-planar stereo for urban scene reconstruction, in: CVPR, 2010.

[52] P. Alliez, G. Ucelli, C. Gotsman, M. Attene, Shape Analysis and Structuring, Springer Berlin Heidelberg, 2008, Ch. Recent Advances in Remeshing of Surfaces, pp. 53–82.

[53] T. Werner, A. Zisserman, New techniques for automated architecture reconstruction from photographs, in: ECCV, 2002.

[54] N. Snavely, S. M. Seitz, R. Szeliski, Photo tourism: Exploring photo collections in 3D, in: SIGGRAPH, 2006.

[55] F. Fraundorfer, K. Schindler, H. Bischof, Piecewise planar scene reconstruction from sparse correspondences, IJCV 24 (4) (2006) 395–406.

[56] D. Hoiem, A. A. Efros, M. Hebert, Recovering surface layout from an image, IJCV 75 (1) (2007) 151–172.

[57] A. Saxena, M. Sun, A. Ng, Make3D: Learning 3-D scene structure from a single still image, PAMI 31 (5) (2009) 824–840.

[58] L. Ladicky, J. Shi, M. Pollefeys, Pulling things out of perspective, in: CVPR, 2014.

[59] A. Saxena, M. Sun, A. Ng, 3-D reconstruction from sparse views using monocular vision, in: ICCV, 2007.

[60] O. Faugeras, E. Bras-Mehlman, J. Boissonnat, Representing stereo data with the Delaunay triangulation, Artificial Intelligence 44 (1) (1990) 41 – 87.

[61] Q. Pan, G. Reitmayr, T. Drummond, ProFORMA: Probabilistic Feature-based On-line Rapid Model Acquisition, in: BMVC, 2009.

[62] C. Hoppe, M. Klopschitz, M. Donoser, H. Bischof, Incremental surface extraction from sparse structure-from-motion point clouds, in: BMVC, 2013.

[63] D. D. Morris, T. Kanade, Image-consistent surface triangulation, in: CVPR, 2000.

[64] C. Taylor, Surface reconstruction from feature based stereo, in: ICCV, 2003, pp. 184–190 vol.1.

[65] A. Hilton, Scene modelling from sparse 3D data, Image and Vision Computing 23 (10) (2005) 900 – 920.

[66] A. Geiger, M. Roser, R. Urtasun, Efficient large-scale stereo matching, in: ACCV, 2010.

[67] M. Lhuillier, Toward automatic 3D modeling of scenes using a generic camera model, in: CVPR, 2008.

[68] M. Lhuillier, A generic error model and its application to au-

tomatic 3D modeling of scenes using a catadioptric camera, IJCV 91 (2) (2011) 175–199.
[69] M. Kazhdan, M. Bolitho, H. Hoppe, Poisson surface reconstruction, in: Eurographics, 2006, pp. 61–70.
[70] Y. Ohtake, A. Belyaev, H.-P. Seidel, A multi-scale approach to 3D scattered data interpolation with compactly supported basis functions, in: Shape Modeling International, 2003, pp. 153–161.
[71] L. Wang, R. Yang, Global stereo matching leveraged by sparse ground control points, in: CVPR, 2011.
[72] P. Fua, A parallel stereo algorithm that produces dense depth maps and preserves image features, Machine Vision and Applications 6 (1) (1993) 35–49.
[73] J. Kopf, M. F. Cohen, R. Szeliski, First-person hyper-lapse videos,, in: SIGGRAPH, 2014.
[74] C. Frueh, S. Jain, A. Zakhor, Data processing algorithms for generating textured 3D building facade meshes from laser scans and camera images, in: IJCV, 2005.
[75] T. Pylvanainen, J. Berclaz, V. Hedau, T. Korah, M. Aanjaneya, R. Grzeszczuk, 3D city modeling from street-level data for augmented reality applications, in: 3DPVT, 2012.
[76] D. Gallup, M. Pollefeys, J.-M. Frahm, 3D reconstruction using an n-layer heightmap, in: DAGM, 2010.
[77] O. Teboul, L. Simon, P. Koutsourakis, N. Paragios, Procedural modeling and image-based 3D reconstruction of complex architectures through random walks, in: IJCV, 2010.
[78] H. Riemenschneider, A. Bodis-Szomoru, J. Weissenberg, L. Van Gool, Learning where to classify in multi-view semantic segmentation, in: ECCV, 2014.
[79] M. Kada, L. McKinley, 3D building reconstruction from lidar based on a cell decomposition approach, in: ISPRS Workshop CMRT09, 2009, pp. 47–52.
[80] F. Lafarge, X. Descombes, J. Zerubia, M. Pierrot-Deseilligny, Structural approach for building reconstruction from a single DSM, PAMI 32 (1) (2010) 135–147.
[81] M. Ortner, X. Descombes, J. Zerubia, Building outline extraction from digital elevation models using marked point processes, IJCV 72 (2) (2007) 107–132.
[82] F. Lafarge, C. Mallet, Creating large-scale city models from 3D-point clouds: A robust approach with hybrid representation, IJCV 99 (1) (2012) 69–85.
[83] J. Chen, B. Chen, Architectural modeling from sparsely scanned range data, IJCV 78 (2-3) (2008) 223–236.
[84] C. Poullis, S. You, Automatic reconstruction of cities from remote sensor data, in: CVPR, 2009.
[85] L. Zebedin, J. Bauer, K. Karner, H. Bischof, Fusion of feature- and area-based information for urban buildings modeling from aerial imagery, in: ECCV, 2008.
[86] S. Kluckner, H. Bischof, Image-based building classification and 3D modeling with super-pixels, in: PCV, 2010.
[87] Q.-Y. Zhou, U. Neumann, 2.5D dual contouring: A robust approach to creating building models from aerial lidar point clouds, in: ECCV, 2010.
[88] Q.-Y. Zhou, U. Neumann, 2.5D building modeling with topology control, in: CVPR, 2011.
[89] Q.-Y. Zhou, U. Neumann, 2.5D building modeling by discovering global regularities, in: CVPR, 2012.
[90] Y. Verdie, F. Lafarge, P. Alliez, LOD generation for urban scenes, ACM Graphics 34 (3) (2015) 30:1–30:14.
[91] D. H. Douglas, T. K. Peucker, Algorithms for the reduction of the number of points required to represent a digitized line or its caricature, Cartographica 10 (2) (1973) 112–122.
[92] P. F. Felzenszwalb, D. P. Huttenlocher, Efficient graph-based image segmentation, IJCV 59 (2) (2004) 167–181.
[93] H. Edelsbrunner, D. G. Kirkpatrick, R. Seidel, On the shape of a set of points in the plane, IEEE Transactions on Information Theory 29 (4) (1983) 551559.
[94] V. Kolmogorov, R.Zabih, What energy functions can be minimized via graph cuts?, PAMI 26 (2) (2004) 147–159.
[95] Y. Boykov, O. Veksler, R. Zabih, Fast approximate energy minimization via graph cuts, PAMI 23 (11) (2001) 1222–1239.
[96] C. Strecha, W. von Hansen, L. Van Gool, P. Fua, U. Thoennessen, On benchmarking camera calibration and multi-view stereo for high resolution imagery, in: CVPR, 2008.
[97] C. Wu, VisualSFM: A Visual Structure from Motion System, `http://ccwu.me/vsfm/` (Accessed 28 Jun 2013, version 0.5.22).
[98] C. Wu, S. Agarwal, B. Curless, S. M. Seitz, Multicore bundle adjustment, in: CVPR, 2011.
[99] C. Wu, SiftGPU: A GPU implementation of Scale Invariant Feature Transform (SIFT), `http://cs.unc.edu/~ccwu/siftgpu` (Accessed 14 Nov 2013, version 0.5.400).
[100] N. Salman, M. Yvinec, Surface reconstruction from multi-view stereo of large-scale outdoor scenes, International Journal of Virtual Reality 9 (1).
[101] Y. Wei, L. Quan, Region-based progressive stereo matching, in: CVPR, 2004.
[102] C. Harris, M. Stephens, A combined corner and edge detector, in: Alvey Vision Conference (AVC), 1988.
[103] J. Canny, A computational approach to edge detection, PAMI 8 (6) (1986) 679–698.
[104] M. Bergh, X. Boix, G. Roig, B. Capitani, L. Van Gool, SEEDS: Superpixels extracted via energy-driven sampling, in: ECCV, 2012.
[105] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, S. Süsstrunk, SLIC superpixels compared to state-of-the-art superpixel methods, PAMI 34 (11) (2012) 2274–2282.
[106] CGAL, The Computational Geometry Algorithms Library, `http://www.cgal.org` (Accessed 2 Oct 2015, version 4.6.3).
[107] MeshLab, `http://meshlab.sourceforge.net/` (2 Apr 2014, version 1.3.3).
[108] M. Mauro, H. Riemenschneider, L. Van Gool, A. Signoroni, R. Leonardi, An Integer Linear Programming model for View Selection on Overlapping Camera Clusters, in: 3DV, 2014.