Regularized 3D Modeling from Noisy Building Reconstructions

Thomas Holzmann   Friedrich Fraundorfer   Horst Bischof
Institute for Computer Graphics and Vision, Graz University of Technology, Austria
{holzmann,fraundorfer,bischof}@icg.tugraz.at

Abstract

In this paper, we present a method for regularizing noisy 3D reconstructions, which is especially well suited for scenes containing planar structures like buildings. At horizontal structures, the input model is divided into slices and for each slice, an inside/outside labeling is computed. With the outlines of each slice labeling, we create an irregularly shaped volumetric cell decomposition of the whole scene. Then, an optimized inside/outside labeling of these cells is computed by solving an energy minimization problem. For the cell labeling optimization we introduce a novel smoothness term, where lines in the images are used to improve the regularization result. We show that our approach can take arbitrary dense meshed point clouds as input and delivers well regularized building models, which can be textured afterwards.

1. Introduction

Using standard 3D reconstruction techniques it is possible nowadays to generate dense point clouds which can be meshed afterwards to create a surface accurately representing the reconstructed scene. For example, one can use open source Structure-from-Motion (SfM) pipelines like Bundler [15] or Colmap [14] to estimate the camera poses and compute a sparse point cloud representing the scene. Afterwards, densification methods like PMVS2 [4] or Sure [13] can be used to densify the point cloud. Finally, meshing techniques like Poisson surface reconstruction [7] or the approach from Labatut et al. [9] produce a detailed mesh representing the reconstructed scene.

However, due to measurement uncertainties in various steps of the reconstruction process, such 3D models are noisy and contain clutter. Therefore, they are not visually appealing even though they contain many details. Additionally, the amount of data used to process, store and transmit such models is quite high, as they can contain millions of points modeling the reconstructed noisy surfaces, which might be problematic for applications using 3D maps but just have limited resources available. Therefore, for many processing and viewing applications, a regularized, more compact representation is desired. This should be a representation excluding the noise and clutter from a dense reconstruction and should be as near as possible to reality if desired, but it should also be possible to generate more

This research was funded by the Austrian Science Fund (FWF) in the project V-MAV (I-1537).

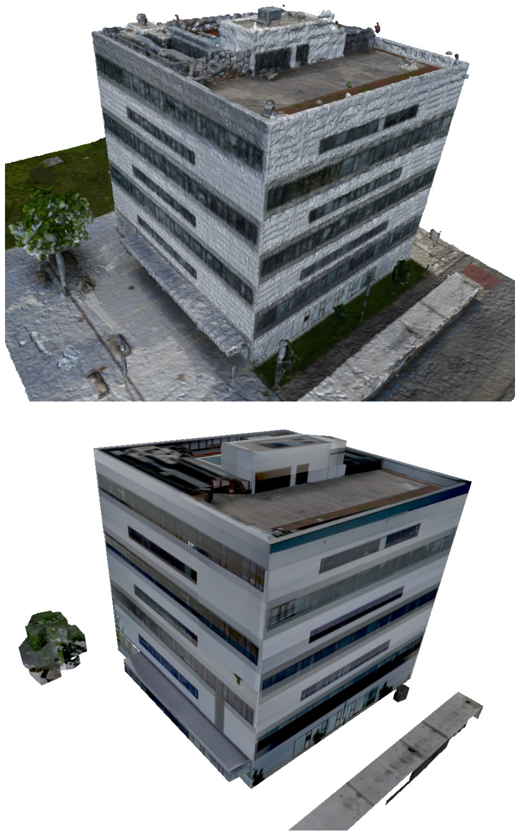


Figure 1. *Meshed dense 3D reconstruction and resulting regularized model. Top:* A 3D building reconstruction created from aerial images taken by an unmanned aerial vehicle (UAV). The camera poses and initial 3D structure were computed with SfM, then a dense model was computed with Sure [13] which was finally meshed using Poisson surface reconstruction [7] (visualized with vertex coloring). Such a mesh can be used as input for our approach. *Bottom:* Resulting regularized model from our pipeline, textured with [17]. We deliver clean and smooth surfaces where the meshed dense point cloud contains a lot of noise. The parameters were set to: $\lambda_{mesh} = 0.4$, $\lambda_{lines} = 0.1$ and $d = 50$.

abstract models, that don't cover details but represent the geometric structure well.

In this paper, we introduce a 3D reconstruction algorithm, which creates regularized 3D models consisting of geometric primitives from image-based 3D reconstructions. Such a regularization should remove small details which are likely to be noise and should describe the input mesh with a small set of vertices and faces. Simultaneously to the data reduction, the regularized model should describe the scene in a proper way, *i.e.* that planar surfaces in the scene are planes in the regularized model and not a noisy surface similar to a plane.

Based on [6], we first divide the dense model into multiple horizontal slices, which are parts of the model bounded by dominant horizontal structures. Then, we compute an inside/outside labeling for each slice using the visibility information of each point in the slice. Using the outlines of the labeling per slice, we compute an irregularly shaped cell decomposition of the whole scene. Finally, we optimize the model by solving a CRF. In this optimization, the level of regularization can be adjusted. In contrast to [6], we introduce a novel smoothness term based on detected line segments in the images, which improves the reconstruction results, especially in areas where the input model contains noisy surfaces. An example result can be seen in Fig. 1.

2. Related Work

There exist several different approaches to regularize 3D models using geometric primitives.

In [19], Zebedin et al. introduce a geometric modeling approach for buildings. Using height data as input, they approximate façades with planes and roofs as surfaces of revolution, which are a natural description of domes and spires and can therefore be used to describe various shapes. With their approach, it is also possible to adjust the level of detail of the reconstruction. Even though their algorithm produces good results with aerial images, they do not model façades, as they are just using height maps as input.

Another approach for abstraction of image-based reconstructions of urban scenes was proposed in [16]. They first classify superfacets into multiple categories like ground, façades or roofs. Then each category is treated differently: *e.g.*, ground is represented by a Delaunay triangulation lifted in 3D and façades and roofs are approximated by a set of planar proxies. With their approach, it is possible to abstract large-scale urban scenes and create models with different levels of detail. Though, their approach is limited to specific pre-defined classes which are used for the superfacet classification.

Xiao and Furukawa [18] proposed a method for indoor modeling using Constructive Solid Geometry (CSG). Using laser scanner data as input, they first separate the model into horizontal slices. Then, they estimate a 2D CSG representation for each slice, which is a set of rectangular primitives, by using the visibility information. Taking these 2D CSG representations as input, they estimate a 3D CSG representation, which is a smooth representation of the whole scene. Finally, they texture the model using ground-level photographs to create visual appealing results. However, their approach is restricted to Manhattan-like scenes and is not designed for noisy image-based reconstructions.

Another approach for indoor scene reconstruction using laser scan data was presented by Oesau et al. in [10] and [11]: Similarly to [18], they partition the scene into horizontal slices at dominant horizontal structures. Then, they also detect dominant vertical structures and use them as splitting planes in order to partition the whole scene into irregularly shaped volumetric cells. These cells are then labeled as free or occupied space by solving an energy minimization problem. In contrast to [18], their approach is also able to reconstruct non-Manhattan-like scenes due to their irregularly shaped cell representation. Though, it is still not well suited for image-based reconstructions, which are potentially sparser and contain more clutter.

In Holzmann et al. [6], a similar approach adapted to image-based reconstructions from buildings was presented. In their approach, the building model is partitioned into horizontal slices separated by dominant horizontal structures. Then, for every slice a floor plan is created by using the visibility information. The floor plans from all slices are then triangulated and used as base faces for the irregularly shaped volumetric cells, which span the whole scene. The whole shape of the building is then estimated by labeling each volumetric cell as inside or outside by solving an energy minimization problem. With the smoothness parameter used in the energy minimization, it is possible to create models with varying degree of regularization. Due to the adaptations especially made for image-based reconstructions, their approach works also well on noisy image-based reconstructions. However, if the error in the input 3D data is too big (*e.g.*, at reflecting windows, due to missing texture), the resulting geometrically abstracted model might follow these errors, as no additional information is used.

In our approach we use horizontal slicing and volumetric cell decomposition similar as in [6]. Additionally, we use image information in order to improve the result especially in erroneous parts of the 3D reconstruction.

3. Geometric Modeling Using Line Cues

In this section, we describe our processing pipeline in detail. A schematic overview of our system can be seen in Fig. 2. First, we separate the meshed input point cloud into horizontal slices and compute an inside/outside labeling for every slice. Using the outlines of the inside labeled regions of all slices, we create an irregularly shaped volumetric cell decomposition of the whole scene. Finally, we generate an

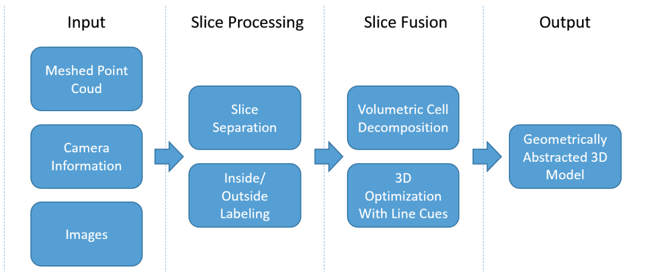


Figure 2. *Overview of the processing pipeline.* We take any meshed point cloud with arbitrary density with the corresponding camera positions and image informations as input. We separate the input model into horizontal slices and compute an inside/outside labeling for each slice. Using visibility information and detected lines in images, we compute an optimized regularized 3D model.

optimized inside/outside labeling of the volumetric cells using visibility information and 2D line segments by formulating the labeling as an energy minimization problem.

3.1. Input Data

As input data, we take a meshed point cloud with arbitrary density and its corresponding camera information (*i.e.* the position of the cameras and the captured images). Our approach works on meshed dense point clouds but also on meshed sparse point clouds (*e.g.*, a result from SfM). An example dense building model can be seen in Fig. 1.

3.2. Horizontal Slicing and Cell Decomposition

This section describes the separation of the input model into horizontal slices, the computation of an inside/outside labeling for each slice and the creation of an irregularly shaped cell decomposition of the whole scene. Except some small changes, this part is already described in [6]. Hence, we refer the reader to [6] for a more detailed description.

As a first step, we detect horizontal structures and separate the model at these structures into horizontal slices. For this, we first need to detect the horizontal direction. As we assume that the ground plane is the dominant plane, we do this by plane fitting: We try to find a plane for which the most points in the scene lie near to it. However, we could also use different methods to estimate the horizontal direction of the scene (*e.g.*, by using inertial measurements). As we usually work with Manhattan-like scenes, we found out that it is beneficial for some processing steps to align the model with the Manhattan world directions. Therefore, we also detect the most dominant plane perpendicular to the ground plane and align the model to these directions. However, one has to mention that a Manhattan world scenery is not necessary for our approach to work.

Having estimated the ground plane, we estimate dominant horizontal structures by applying mode estimation using Mean Shift [3]. For this, we just select points with a normal similar to the normal of the ground plane, which

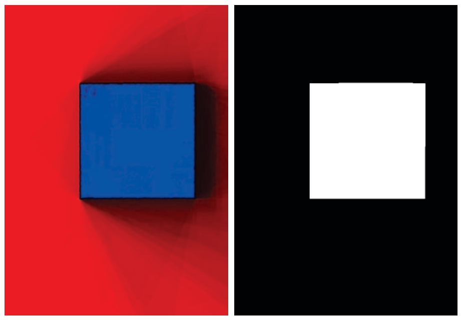


Figure 3. *Slice labeling.* Intermediate processing steps of the processing of a slice in the middle of a simple building. *Left*: An illustration of the free space scores of the slice. The more cameras see the specific area, the more intensive the pixel is red. In case of no visibility: The farther away this area is from a visible part, the more intensive the pixel is blue. *Right:* The resulting inside/outside labeling.

are the points describing horizontal structure, and apply the mode estimation in the horizontal dimension of the scene. The mean shift bandwidth is defined as:

$$bandwidth = \frac{height}{d}, \tag{1}$$

where $height$ is the 3D model height and d is a parameter which has to be set.

Next, we separate the model at the detected horizontal structures into several slices and for each slice, a 2D inside/outside labeling is computed. For this, we need to compute a free space score for each position in the slice. The free space score is positive in areas which could be seen by cameras and negative in areas which could not be seen. To compute a 2D free space score for each position in the slice, we first compute the free space score for each voxel of a voxel grid spanned over the whole scene. Then, we sum up the voxels contained in a slice to get 2D free space scores. Using these free space scores, we compute a binary inside/outside labeling by formulating it as an energy minimization problem and solving it using Graph Cuts [2]. These processing steps are illustrated in Fig. 3.

By extruding the labeling from each slice to 3D, we already get an initial 3D model. However, as each slice is just optimized separately, the vertical surfaces are not smooth. Therefore, we need an additional 3D optimization step, for which we use irregularly shaped volumetric cells, which represent all important structures from the individual slices.

To get a cell decomposition of the whole scene, we project the outlines of the inside labeled parts of all slices onto the ground plane and compute a Constrained Delaunay Triangulation [12] including this lines (see Fig. 4). Finally we extrude the computed triangles between all slice boundaries to get an irregularly shaped cell decomposition of the whole scene including all important scene structures.

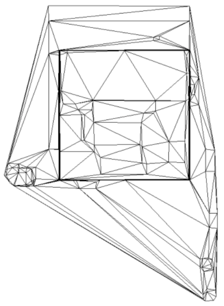


Figure 4. *Constrained Delaunay Triangulation* of outlines of all slices projected to the ground plane. As one can see, near the building walls are many similar lines in the triangulation (thicker lines because of several very near lines) due to noise in the input mesh. All triangles are extruded between all slice boundaries to create a volumetric cell decomposition of the whole scene.

3.3. Volumetric Cell Labeling Using Line Cues

In the final step, we create a regularized labeling of all volumetric cells labeled as inside or outside. As a result, we can create 3D reconstructions consisting of geometric blocks, for which the degree of regularization can be adjusted depending on the smoothness parameters.

We formulate this optimization step as an energy minimization problem. We introduce a novel smoothness term, which uses lines detected in the input images to create a regularized transition from inside label to outside label.

The energy to minimize is defined as:

$$E(L) = \sum_{p \in \mathcal{I}} E_{data}(L(p)) + \sum_{p,q \in \mathcal{N}} E_{smooth}(L(p), L(q)), \tag{2}$$

where $\mathcal{I}$ denotes the set of all volumetric cells, $\mathcal{N}$ is the neighborhood of every cell and L is the (binary) labeling. The neighborhood relation is defined by the volumetric cell complex: all cells that share a common face are neighbors. The data terms, $E_{data}(l_p)$, are defined as the summed up free space scores contained in the volumetric cell normalized by the cell size. The smoothness terms, $E_{smooth}(L(p), L(q))$, are a combination of two smoothness terms and defined as:

$$E_{smooth}(L(p), L(q)) = \lambda_{mesh} E_{s_{mesh}}(L(p), L(q)) + \lambda_{lines} E_{s_{lines}}(L(p), L(q)). \tag{3}$$

$E_{s_{mesh}}(L(p), L(q))$ is the smoothness term which depends on the mesh surface, *i.e.* it is likely that a labeling transition happens if the input mesh surface is near the face between the cells p and q and unlikely otherwise. This smoothness term is described in detail in [6].

The second smoothness term, $E_{s_{lines}}(L(p), L(q))$, is our novel term based on lines detected in the input images. Assuming an initial estimate of the model, all faces connecting two adjacent cells in the volumetric cell decomposition which are visible get backprojected into the images to compute a line-based smoothness score:

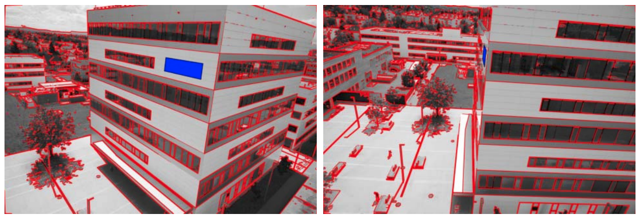


Figure 5. *Camera view with detected lines and backprojected face. Left:* Backprojected face (blue) is not near detected lines, even though it is on the surface of the building. Therefore, this view is not used for the line criterion, as the camera center is not near to coplanar with the face. *Right:* Detected face (blue, hardly visible) and camera are nearly coplanar and the face is nearer to detected lines, as it should be for a face being on the surface of the building. Therefore, this view is used for face score computation.

$$E_{s_{lines}}(L(p), L(q)) = \begin{cases} 0 & \text{if } L(p) == L(q) \\ faceScore(p,q) & \text{else if visible} \\ 1 & \text{else} \end{cases}, \tag{4}$$

where $faceScore(p, q)$ computes a line-based score for the face connecting cell p and q and visible means that at least one vertex of the face is visible by a camera. Therefore, a score gets also computed for faces which are currently not visible but have a connection to the model surface.

To compute the face score based on lines, first line segments are detected using the Line Segment Detector (LSD) [5]. Then, for every face, cameras are selected for which the camera centers are nearly coplanar to the face: The maximum allowed angle between face and camera center is set to 25 deg. This improves the influence of the smoothness term for faces, which belong actually to the model surface but are in the middle of a façade in the images. Fig. 5 illustrates this problem.

In the selected cameras for a face, the distance from the detected line segments to each face edge is computed. Therefore, we search for the nearest line detection with a similar direction than the face edge: The maximum allowed angle between a detected line and face edge is set to 10 deg. With this restriction, just lines are used that are similar to the structure of the face.

For all faces with lines with similar direction, we search for the line which has the smallest normal distance to the face edge. If this distance is above a threshold, it is truncated. The threshold for this maximum line distance is adjusted by the parameter $maxLineDist$ (which we set to 70 $pixels$) and is computed as $maxLineDist$ normalized by the camera distance to the current face and the model size:

$$truncVal = \frac{maxLineDist}{normalizedCamDist}. \tag{5}$$

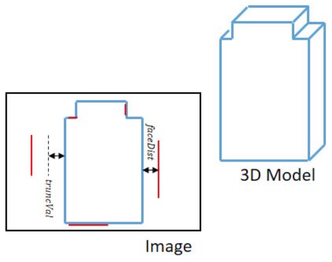


Figure 6. *Line distance computation.* All visible faces are projected into the suitable images. If, as illustrated, the projected surface of the model of the current iteration (blue) is near a detected line (red), this surface will likely stay the same also in the next iteration. Contrary, if the current surface has a high distance to a detected line, as can be seen at the sides of the projection, the surface will probably get shifted towards the detected line in the next iteration. If the distance is too high (left side), it gets truncated.

$normalizedCamDist$ is defined as:

$$normalizedCamDist = \frac{camDist}{avgModelSize}, \tag{6}$$

where $avgModelSize$ is the mean of the maximum x- and y-extension of the input model and $camDist$ is the distance of the camera to the face in arbitrary scale retrieved by SfM. This is necessary, as we don't have a fixed (or metric) scale in our reconstructions and want to enforce a similarly scaled camera distance for all models.

The normalization by $normalizedCamDist$ depicted in Eq. 5 is beneficial, as the normalized truncation value $truncVal$ now corresponds to distances in the model and not to pixel distances. This face-to-line distance truncation significantly improves the results, as it lets the optimization focus on relevant, nearby lines and ignore far away lines.

Consequently, the truncated distance of a face to a line (illustrated in Fig. 6) is computed by

$$truncFaceDist = min(faceDist, truncVal). \tag{7}$$

Finally, the computed face-to-line distance gets multiplied by $normalizedCamDist$ to transform the distance from pixel to an absolute scale: If a camera is farther away, one pixel is a bigger distance than in a near camera. To get the total face-to-line distance for the face backprojected in one camera, we compute the average of all face edges.

The final face score, which is calculated using all cameras which fulfill the above mentioned requirements, is defined as:

$$faceScore = \begin{cases} \frac{avgFaceDist}{maxLineDist} & \text{if } \#validCams > 0 \\ 1 & \text{else} \end{cases}, \tag{8}$$

where

$$avgFaceDist = \frac{\sum_{validCams} truncFaceDist}{\#validCams} \tag{9}$$

and $maxLineDist$ is the parameter also used in Eq. 5.

Therefore, the $faceScore$ defined in Eq. 8 is normalized between 0 and 1, where 0 means the face is near to structures (lines) in the images, and 1 means that the face is far from structures.

As we just compute face scores for faces which have a connection to the currently estimated visible space, we need to do several iterations of the energy minimization step defined in Eq. 2. In the first iteration, the line smoothness term is disabled (λ_{lines} is set to 0), and an initial model is estimated. Starting from the second iteration, also the line smoothness term is used. Usually, 4 iterations are enough to let the model converge to a good solution. Finally, we get an optimized inside/outside labeling containing all the irregularly shaped volumetric cells. From this labeling, models consisting of geometric blocks can be created, which contain well regularized vertical surfaces (*e.g.*, façades) even when using erroneous input data. In the next section, we will show and discuss some results with different input data.

4. Experiments

In our experiments, we show that our pipeline can create geometrically regularized 3D models from buildings which finally consist of smooth and clean surfaces instead of a noisy mesh, as in the input data. It can handle input meshes with different quality and input density, where especially on meshed point clouds including some errors (*e.g.*, a point cloud created from SfM) our novel, line-based smoothness term improves the quality of the resulting 3D model compared to the approach presented in [6] (which will be denoted as mesh-based smoothing method in this section). Finally, we deliver textured output models, which can be variably regularized and contain smooth surfaces where meshed point clouds, in contrast, contain a lot of noise and clutter.

4.1. Input Data

As input data, we use meshed 3D reconstructions from buildings. All input images were captured with an UAV equipped with a Sony Alpha 6000 camera. Then, they were processed with our own SfM pipeline. For some experiments, the resulting sparse point cloud from SfM was meshed with our implementation of [9] and used as input. However, to get more accurate, dense point clouds, we used PMVS2 [4] or Sure [13] and used the resulting point cloud meshed by Poisson surface reconstruction [7] as input.

4.2. Implementation Details

Our implementation uses the Graph Cut implementation from Olga Veksler [8] [2] [1], OpenCV for various image processing steps, the original LSD implementation [5] and is implemented mainly in C++. For texturing, we use the implementation described in [17]. As our regularized face triangles were sometimes too big for texturing, we applied midpoint subdivision on the mesh before texturing to make

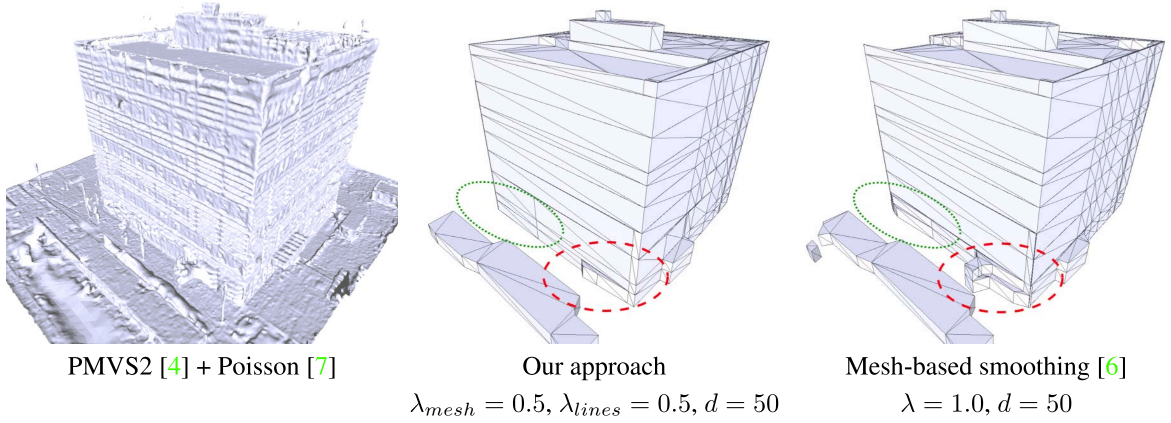

PMVS2 [4] + Poisson [7]
Our approach
$\lambda_{mesh} = 0.5$, $\lambda_{lines} = 0.5$, $d = 50$
Mesh-based smoothing [6]
$\lambda = 1.0$, $d = 50$

Figure 7. *Results for the Block Building with PMVS2. Left:* The input data created with PMVS2 and Poisson surface reconstruction. As one can see, this reconstruction contains more noise and clutter and is incomplete compared to the Sure reconstruction in Fig. 1. *Middle:* The result of our approach, which contains less artifacts (green dotted and red dashed circle) compared to the Mesh-based smoothing (*right*). The parameters for both were chosen to create smooth and clean façades while simultaneously keeping most of the structural details.

the mesh triangles smaller. Our experiments were executed on an Intel Core i7-4820k @ 3.7 GHz with 16 GB RAM.

As our approach, compared to [6], additionally detects line segments and runs multiple iterations of optimization, it has a higher runtime. Though, the runtime highly depends on the structure of the 3D model (*e.g.*, number of slices, ratio free/occupied space voxels, number of images used for line detection) and therefore it can vary significantly between different models. The runtime at the Block Building model with PMVS2 input data was 50.4 minutes.

4.3. Parameter Selection

We changed three parameters during our experiments. First, we set the Mean Shift parameter d depending on the structure of the scene. If the building is not very high or if there are not many horizontal structuring elements, one can set d lower to produce less slices. If there is a lot of horizontal structure in small distances, one should set d higher to produce more slices and cover all horizontal elements. Usually, we set this parameter between 40 and 50.

The two smoothness parameters λ_{mesh} and λ_{lines} define the amount of regularization. The selection of a good value for λ_{mesh} heavily depends on the quality of the input mesh: If we use a nearly error-free input mesh just including a little bit of noise, it is recommended to set λ_{mesh} higher in comparison to λ_{lines}. Contrary, if the input mesh contains big errors we suggest to set λ_{mesh} lower, as otherwise the regularized output may be influenced too much from the erroneous mesh. λ_{lines} should be set according to the amount of errors which should be corrected with this image term. For an erroneous mesh, one should set λ_{lines} to a higher value, for a mesh containing just few errors, one should set λ_{lines} to a lower value. We usually set λ_{mesh} between 0.2 and 0.8 and λ_{lines} between 0.1 and 0.5.

4.4. Block Building

This simple building consists out of one block with some additional installations on the roof and some small roofs on the side. In Fig. 8, example input images are printed. For this dataset, we used meshed SfM, PMVS2 [4] and Sure [13] data as input.

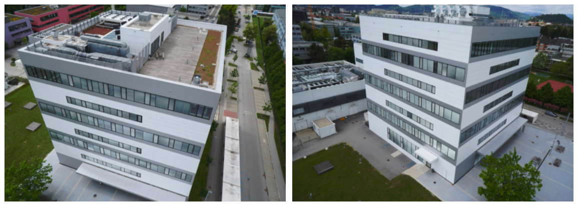
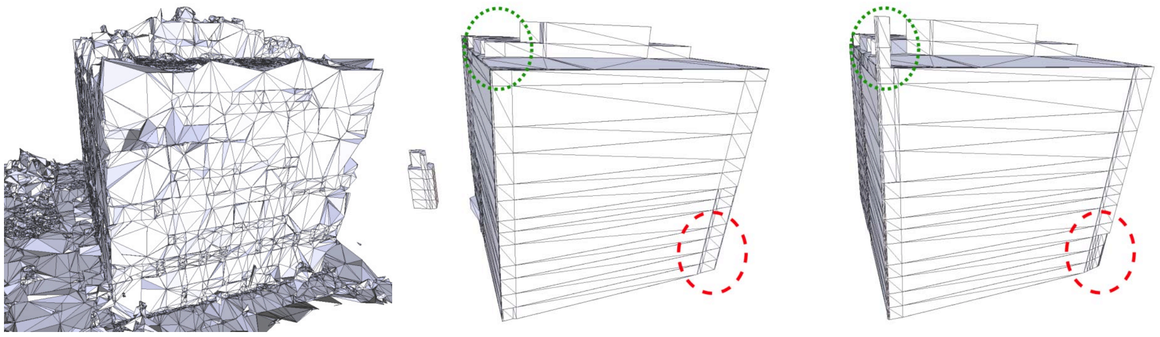


Figure 8. *Input images of the Block Building.* As one can see, this simple building consists mainly of planar façades, including several windows and poorly textured surfaces.

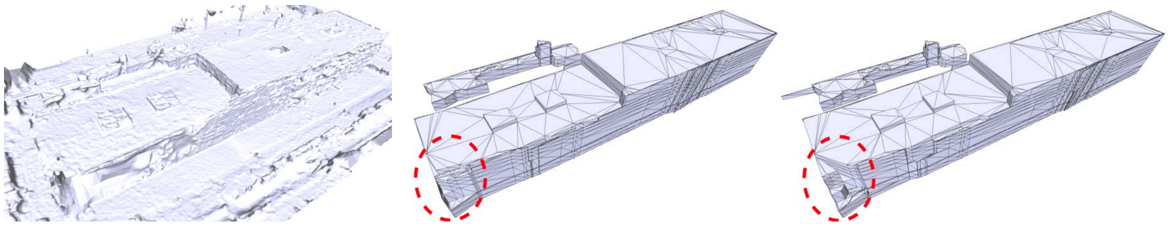
As Sure delivers the most accurate point cloud, also the results from our pipeline using Sure data as input are the most convincing ones. You can see the textured result and the corresponding input data in Fig. 1. Using dense Sure input data, only a little smoothing was necessary to get smooth surfaces and the new line smoothness parameter does not significantly improve the result. Therefore we set especially the line smoothness parameter comparably low, as setting the mesh smoothness term already results in a well regularized model. The Mean Shift parameter d was set so that all important horizontal structures were detected (*e.g.*, small roof above entrance, installations on roof).

The results from our pipeline using meshed PMVS2 data as input can be seen in Fig. 7. Our approach (middle) has a smoother surface compared to the mesh-based smoothing approach (right) especially in the area in the red dashed and green dotted circle. In the red dashed circle, a tree was fused with the building in the input model. Therefore, using mesh-based smoothing, the optimization found the optimal solution by including the tree within the output model. Instead, when using our approach, a line could still be detected in the images at the boundary of the building and therefore the optimization pulled the result towards the cor-



SfM + Labatut et al. [9] meshing | Our approach $\lambda_{mesh} = 0.2, \lambda_{lines} = 0.5, d = 50$ | Mesh-based smoothing [6] $\lambda = 3.5, d = 50$

Figure 9. *Results for the Block Building with pure SfM. Left:* The input data, which is the SfM point cloud meshed by Labatut et al. [9]. This reconstruction is very noisy and also contains quite big wrong parts. *Middle:* Nevertheless, our approach estimates still smooth surfaces. Though, some details are missing compared to Fig. 7. *Right:* Even though the smoothing is already quite strong, which leads to smoothing artifacts (*e.g.*, in the green dotted circle), the mesh-based smoothing still follows the erroneous mesh strictly (*e.g.*, in the red dashed circle).



PMVS2 [4] + Poisson [7] | Our approach $\lambda_{mesh} = 0.8, \lambda_{lines} = 0.5, d = 40$ | Mesh-based smoothing [6] $\lambda = 3.5, d = 40$

Figure 10. *Results for the Long Building.* As one can see, the input data (*left*) has errors in the mesh. This occurred due to insufficient image data of this part of the building. Because of this errors, the mesh-based smoothing (*right*) cannot regularize this part well (red dashed circle), where our approach (*middle*) makes a better approximation of the real façade.

rect solution. In the green dotted circle, the mesh-based smoothing also follows the noisy mesh surface, where our approach delivers a cleaner surface. We selected the parameters to keep important parts of the building (*e.g.*, when setting λ for mesh-based smoothing higher, the small low building in front of the Block Building would vanish due to over-smoothing).

In order to show that our approach can also handle very noisy meshes created from sparse point clouds, we meshed the resulting point cloud from SfM with Labatut et al. [9] and used this data as input for our pipeline. In Fig. 9, the input mesh, the result from our approach and the result from mesh-based smoothing [6] is shown. Our approach (middle) delivers smooth surfaces and still contains some small details. In contrast, the mesh-based smoothing (right) still follows the erroneous mesh too strictly at some parts (*e.g.*, in the red dashed circle), even though the smoothing is already at its limits, which leads to some smoothing artefacts (*e.g.*, in the green dotted circle). If we increase the smoothing (*i.e.* increase the value for λ), more and more artifacts arise while the error in the red dashed circle will stay.

4.5. Long Building

This building consists of two parts with two different heights. Input images can be seen in Fig. 11. For this dataset, we just used meshed PMVS2 [4] data as input.

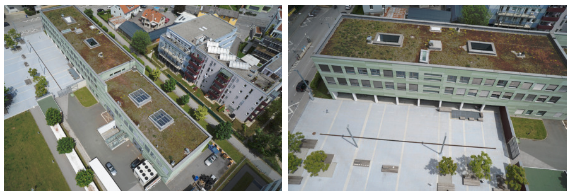


Figure 11. *Input images of the Long Building.* The building mainly consists out of 2 block parts with different height.

In Fig. 10, results from our approach and from the meshed-based smoothing approach [6] are printed. As not enough images were captured for the seen side side of the building, there are big errors in the reconstruction (left). The mesh-based smoothing approach (right) cannot correct these errors properly (*e.g.*, at the part in the red dashed circle). When enforcing more smoothing (*i.e.* using a higher value for λ), the result gets even worse as this approach follows the erroneous mesh. Contrary, our approach (middle) approximates a planar surface (which is the geometric structure of the façade) better, as it uses additional information from the original input images.

In Fig. 12, another part of the building of the results presented in Fig. 10 is visible. The mesh-based smoothing approach (right) has irregularities on the surface (*e.g.*,

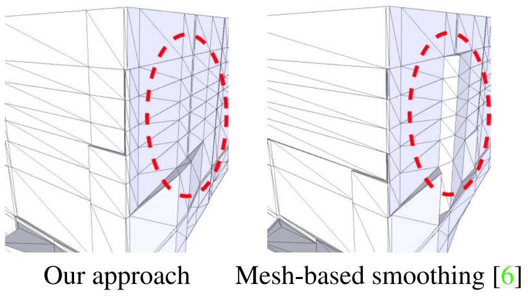


Figure 12. *Back side of the Long Building* of the same results as illustrated in Fig. 10. As one can see, our approach (*left*) delivers a smoother façade compared to mesh-based smoothing (*right*).

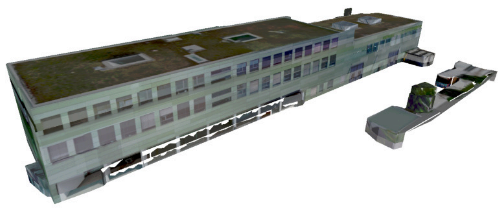


Figure 13. *Textured Long Building.* Same result from our approach as in Fig. 10 with viewing direction from the opposite side. Textured afterwards with [17].

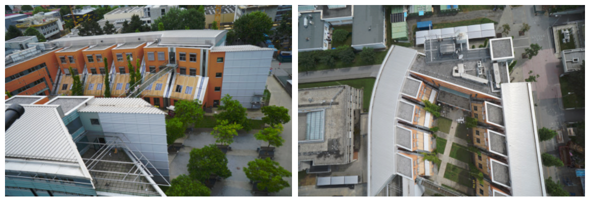


Figure 14. *Input images of the Non-Manhattan Building.* This building contains non-Manhattan-like façades and sloped surfaces.

in the red dashed circle), as it follows the surface of the noisy mesh. Contrary, our approach (left) delivers a much smoother, planar surface similar to the original façade.

The same result from our approach textured afterwards with [17] can be seen in Fig. 13. Due to a wrong geometry in the lower left part, several faces could not be textured and are therefore white in this reconstruction. This happens due to the noisy PMVS2 data, with which we could not estimate a correct geometry for this part. However the other parts look good which indicates that the estimated geometry is correct.

4.6. Non-Manhattan Building

This dataset contains a more complex building. In difference to the other buildings shown in our experiments, it does not have a Manhattan-like outline and contains sloped surfaces. In Fig. 14, one can see example input images.

The regularized result and the dense input reconstruction computed with Sure [13] can be seen in Fig. 15. Non-rectangular outlines of the building are not a problem for the algorithm, as any outline gets approximated by a polygonal line. Also sloped structures are approximated by stairway-

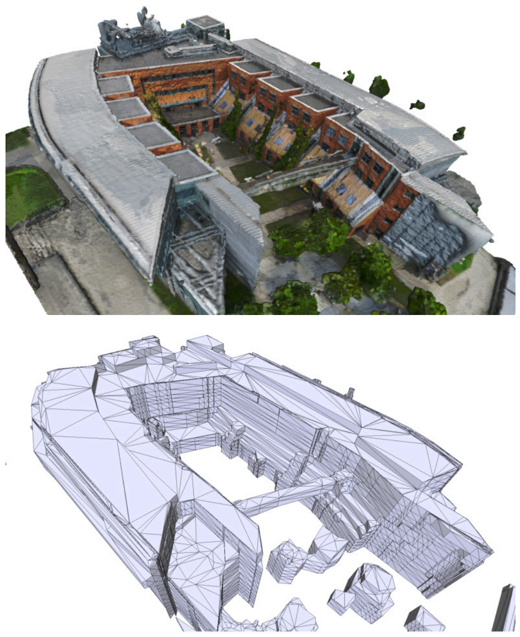


Figure 15. *Result from the Non-Manhattan Building Top:* Input mesh created with Sure [13] and Poisson meshing [7] (vertex coloring). *Bottom:* Resulting regularized model. Sloped surfaces are approximated with stairway-like structures, non-Manhattan-like outline is approximated by a polygonal line. The parameters were set to: $\lambda_{mesh} = 0.4$, $\lambda_{lines} = 0.2$ and $d = 50$.

like structures. To get a more detailed approximation one could lower the the Mean Shift bandwidth (*i.e.* higher the value for d) in order to get more slices and therefore a finer stairway approximation. However, one has to mention that this is a difficult building, where also vegetation is very near or fused with the building , which makes it difficult at some parts to get a correct labeling without additional semantic information. Therefore, some details are missing in the regularized model.

5. Conclusion and Future Work

We have presented an approach for regularizing noisy building reconstructions, which improves the regularization by using detected lines in the input images. We have shown that our approach improves the results when using erroneous input data, where the additional image information is very beneficial. We deliver well regularized 3D models, which can be used for visualization, where noisy 3D models are not desired, but are also useful for further processing (*e.g.*, on devices with limited resources), as the amount of data is massively reduced.

Future work will include the usage of semantic scene information to separate buildings from other 3D objects (*e.g.*, vegetation). We will also investigate in the modeling of sloped roofs, which are currently just approximated by stairway-like structures.

References

[1] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2004. 5

[2] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1222–1239, 2001. 3, 5

[3] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002. 3

[4] Y. Furukawa and J. Ponce. Accurate, dense, and robust multi-view stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010. 1, 5, 6, 7

[5] R. Grompone, J. Jakubowicz, J. M. Morel, and G. Randall. Lsd: A fast line segment detector with a false detection control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(4):722–732, April 2010. 4, 5

[6] T. Holzmann, C. Hoppe, S. Kluckner, and H. Bischof. Geometric abstraction from noisy image-based 3d reconstructions. In *Proceedings of The 38th Annual Workshop of the Austrian Association for Pattern Recognition (ÖAGM)*, 2014. 2, 3, 4, 5, 6, 7, 8

[7] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Eurographics Symposium on Geometry Processing*, 2006. 1, 5, 6, 7, 8

[8] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? In *Proceedings European Conference on Computer Vision*, 2002. 5

[9] P. Labatut, J.-P. Pons, and R. Keriven. Efficient multi-view reconstruction of large-scale scenes using interest points, delaunay triangulation and graph cuts. In *Proceedings International Conference on Computer Vision*, 2007. 1, 5, 7

[10] S. Oesau, F. Lafarge, and P. Alliez. Indoor scene reconstruction using primitive-driven space partitioning and graph-cut. In *Eurographics Symposium on Geometry Processing*, 2013. 2

[11] S. Oesau, F. Lafarge, and P. Alliez. Indoor scene reconstruction using feature sensitive primitive extraction and graph-cut. In *Proceedings International Society for Photogrammetry and Remote Sensing*, 2014. 2

[12] L. Paul Chew. Constrained delaunay triangulations. *Algorithmica*, 4(1-4):97–108, 1989. 3

[13] M. Rothermel, K. Wenzel, D. Fritsch, and N. Haala. Sure: Photogrammetric surface reconstruction from imagery. In *Proceedings LC3D Workshop, Berlin*, 2012. 1, 5, 6, 8

[14] J. L. Schönberger and J.-M. Frahm. Structure-from-motion revisited. In *Proceedings IEEE Conference Computer Vision and Pattern Recognition*, 2016. 1

[15] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: Exploring image collections in 3d. In *ACM Trans. on Graphics (SIGGRAPH)*, 2006. 1

[16] Y. Verdie, F. Lafarge, and P. Alliez. Lod generation for urban scenes. In *ACM Transactions on Graphics*, 2015. 2

[17] M. Waechter, N. Moehrle, and M. Goessele. Let there be color! - large-scale texturing of 3d reconstructions. In *Proceedings European Conference on Computer Vision*, 2014. 1, 5, 8

[18] J. Xiao and Y. Furukawa. Reconstructing the world's museums. In *Proceedings European Conference on Computer Vision*, 2012. 2

[19] L. Zebedin, J. Bauer, K. Karner, and H. Bischof. Fusion of feature- and area-based information for urban buildings modeling from aerial imagery. In *Proceedings European Conference on Computer Vision*, 2008. 2