Rudolf Prettenthaler, BSc

Scalable Surface Reconstruction from Point Clouds

Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Software Engineering and Business Management

submitted to

Graz University of Technology

Supervisors

Dipl.-Ing. Dr.techn. Christian Mostegel
Ass.Prof. Dipl.-Ing. Dr.techn. Friedrich Fraundorfer
Univ.-Prof. Dipl-Ing. Dr.techn. Horst Bischof

Institute for Computer Graphics and Vision
Head: Univ.-Prof. Dipl-Ing. Dr.techn. Horst Bischof

Graz, September 2018

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Date	Signature

Abstract

In this work we present a scalable surface reconstruction method for multi-scale multi-view (stereo) point clouds. The process pipeline consists of an octree data partitioning with unlimited depth and local Delaunay tetrahedralization, followed by the surface extraction: At first overlapping local surface hypotheses are generated by graph cut energy minimizations. These overlapping hypotheses are then merged by using a two step technique where an own formulated graph cut minimizes the hole boundary lengths. In our experiments we challenge our approach with multiple freely available public datasets where we compete with two state-of-the-art scalable surface reconstruction methods. The results prove that our method is on par in terms of quality, completeness and outlier resistance. On a purposely created showcase dataset with 2 billion points and a scale variety of 4 orders of magnitude our approach demonstrates its unprecendented potential for scalability. Furthermore, in all experiments the main memory consumption never exceeded 9 gigabyte per process. These promising results encouraged us to submit this work as a paper to IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017 where it got accepted [36].

Kurzfassung

In dieser Arbeit stellen wir eine skalierende Methode zur Oberflächenrekonstruktion für Punktwolken vor. Unser Ansatz beinhaltet die Aufteilung der Punkte mittels einer Octree-Datenstruktur, welche unbegrenzte Tiefe zulässt, deren Delaunay Triangulierung, gefolgt von der eigentlichen Rekonstruktion: Zu Beginn werden überlappende Oberflächen-Hypothesen durch Anwendung einer Energieminimierung (minimaler Graphen-Schnitt) erzeugt. Die Zusammenführung dieser Hypothesen zu einer vollständigen Oberflächenrekonstruktion erfolgt in zwei Schritten und wird durch eine Minimierung der Länge der Grenzkanten bewerkstelligt. Dabei kommt eine eigens formulierte Energieminimierung mittels Graphen-Schnitt zum Einsatz. In unseren Experimenten vergleichen wir unsere Methode mit zwei aktuellen skalierenden Methoden für Oberflächenrekonstruktion anhand mehrerer frei verfügbarer Datensätze. Die Resultate zeigen, dass unsere Methode in der Qualität, Vollständigkeit der Rekonstruktion und auch in der Beständigkeit gegen Daten-Ausreißer, mit den aktuellen Methoden mithalten kann. Auf einem eigens erstellten Datensatz, welcher aus 2 Milliarden Punkten besteht und eine enorme Messungsdistanz-Varianz (der Größenordnung Vier) beinhaltet, demonstriert unser Ansatz sein Potenzial für skalierbare Oberflächenrekonstruktion. Außerdem hat in allen von uns durchgeführten Experimenten der Arbeitsspeicherverbrauch die 9 Gigabyte Grenze pro Prozess niemals überschritten. Diese vielversprechenden Resultate haben uns ermutigt unsere Methode als Facharbeit für die IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, einzureichen. Unsere Arbeit wurde mit Erfolg angenommen [36].

Contents

List of Figures

1. Introduction

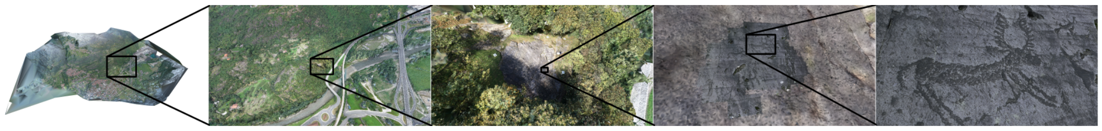


Figure 1.1.: Shows the reconstructed colored mesh from the valley dataset. The rectangles mark the viewpoint of the following outcrop. The first image from the left shows the whole reconstructed area of 6 km². The ground sampling distance drops from left to right from 1 m to 50 µm.

This work focuses on surface reconstruction from multi-view stereo (MVS) generated point clouds with enormous scale variety. In general MVS is a term for a group of techniques that use stereo correspondences between more than two images from a normal still image camera to reconstruct a 3D geometry. Thus we are able to combine images from different acquisition platforms that capture a scene in a large scale variety. For example, we can combine images that were taken by an UAV (unmanned aerial vehicle) to cover a certain region with images from hand held camera that capture images from a short range to preserve more detail. The variety of scale (capture distance) can be enormous. A single UAV can vary its ground sampling distance (distance between pixel centers on the ground) by multiple orders of magnitude by changing its flight altitude. Thus there is no guarantee for a constant point density and 3D uncertainty when combining 3D geometries that were reconstructed from different acquisition platforms. In addition the number of points can be massive: In recent years dense MVS approaches became more and more popular and current state-of-the-art dense MVS approaches can compute 3D points around the total number of acquired pixels per stereo image pair. This scales up for a stereo image pair, taken with a modern camera, up to 10 million points and even more [38]. Capturing images that result in billion of points is neither difficult nor time consuming, it can be done within a few hours of time.

But reconstructing a consistent and highly detailed surface mesh from this huge amount of data is not straightforward. Due to the scale variety and amount of data the scalability or completeness of the reconstructed surface often lacks: While global approaches generally focus on the generation of a fully closed mesh (which usually results in a higher completeness) these methods are not

suitable for being scalable. At one point the data would not fit into the memory and too much data has to be thrown away. Where local approaches can achieve scalability through their local formalization they often trade completeness for scalability. This results in holes in the surface where the point density of the input point cloud changes. Our desired goal is to achieve both, completeness and scalability by dividing the mesh reconstruction into (overlapping) sub problems. In theory this might not be fully achievable: If we consider a point cloud with two clusters of equal metric size but different point density, the sparse point cluster might have no points at all while the dense one might contain more points that cannot fit into memory. Without constraining the magnitude of the point density variety completeness cannot be guaranteed!
Nonetheless our approach can be seen as a hybrid between local and global methods. We make use of the Divide and Conquer principle to encourage scalability. Thus, for being scalable we have to assure that each sub-problem can be processed within an upper limit of computation time and allocated memory. To ensure this we divide our input point cloud with an octree that has unrestrained leaf-node depth. The number of points in a leaf node is an adjustable parameter that regulates the overall memory consumption, a higher number usually improves completeness. From this octree we extract our sub-problems: A sub-problem consists of (up to) eight leaf nodes that lie in a local neighborhood and a leaf node can be part of multiple sub-problems. For each sub-problem we generate a Delaunay tetrahedralization and apply an energy minimization to split the tetrahedra into two sets: One set for tetrahedra that lie *inside* the surface and another one with tetrahedra that lie *outside* the surface. The set of triangles that separate these two sets form surface hypothesis for a sub-problem. We use the method of Labatut et al. [30, 31] to extract these *local* surface hypotheses. For one leaf node we now have multiple surface hypotheses that contain ambiguities. We introduce a graph cut formalization to resolve these conflicts: This formalization uses a surface hypotheses for a leaf node to optimally fill surface holes caused by ambiguities. With optimally we mean the minimization of the length of the surface boundary by adding suitable triangles from surface hypotheses. Thus we increase the completeness of our reconstructed surface. In our experiments we show that we are qualitative and quantitative on par with current state-of-the-art meshing approaches. In addition we are able to reconstruct a consistent mesh from a point cloud of 2 billion points that contain an enormous scale variety of 50 µm to 100 cm. An outlook of our reconstruction capabilities can be seen in Figure 1.1.

2. Related Work

The problem of all surface reconstruction methods is to find an unknown surface S that approximates a physical surface P. An unordered point set $\mathcal{P} = \{p_1, p_2, \cdots, p_n\}$ with $p_i \in \mathbb{R}^3$ sampled on P serves as input. In addition these points are usually not noise free and contain outliers. Thus, noise handling and outlier resilience are two major qualities any surface reconstruction has to fulfill. In this section we focus on methods that use no a priori information about the real surface P and thus have solely to rely on the sampled point set and their visibility information only. This section is split into two parts: In the first we give an overview of different surface reconstruction categories. In the second we discuss several state of the art and scalable surface reconstruction methods.

2.1. Surface Reconstruction - Overview

Surface reconstruction methods can be divided into several different categories that emerged in the last decades. The category pairs we discuss here are intra-class mutual exclusive and inter-class mixable. *Implicit* versus *explicit*: Explicit methods try to find the correct surface directly within the given point cloud. Usually they connect the sampling points of the point cloud by triangles and the reconstructed surface is an exact interpolation within these points. To function properly the samples of the point cloud have to be well aligned and too much noise within the samples can cause problems. Implicit methods on the other hand try to approximate a surface from the given point cloud. They do this by defining an implicit function (often a Signed Distance Function (SDF)) $f : \mathbb{R}^3 \to \mathbb{R}$ with $F(x) < 0$ inside an object and $F(x) > 0$ outside an object. The surface is then reconstructed by extracting the zero set $\{x : F(x) = 0\}$. To define an SDF additional inside/outside information is needed, this can for example be achieved by calculating (if not given) the normal vectors of the point cloud. Figure 2.1 shows a visual example for 2D implicit and explicit surface reconstruction methods.

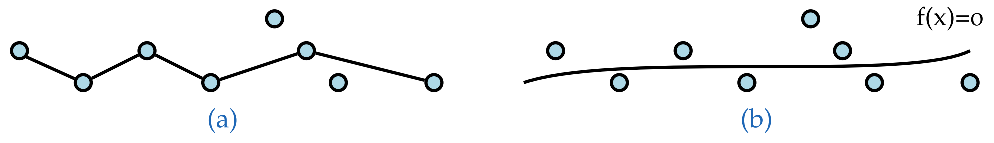


Figure 2.1.: (a) shows an explicit and (b) shows an implicit reconstructed surface for a 2D point cloud.

Volumetric versus *Delaunay*: In general volumetric approaches work on a discretized volume that encloses the object that is to be reconstructed. The volume is often uniformly divided in a regular grid of fixed elements (aka voxels). These volumetric data structures can either store simple state information such as occupancy (probability) [21] or samples of a continuous signed distance function [18, 12]. The former typically generate binary occupancy grids and reconstruct the visual hull of an object only. These occupancy grid based methods often lack in surface quality, but due to their ability of incremental point insertion and real-time capabilities they are preferred for obstacle avoidance in robotics [23]. For the latter a common technique is to convert a depth map in a signed distance field. This field is afterwards cumulatively averaged into the regular voxel grid. By using isosurface polygonization [33] the surface can be extracted as the zero level set of the signed distance function. Scalable surface reconstruction on regular voxel grids is not achievable, therefor several groups utilize recursive [14] data structures that are computationally expensive. Contrary, Delaunay methods work directly on the full point cloud. These methods usually are a variant of explicit surface reconstruction methods where the basis is a Delaunay complex. They are explicitly discussed here because they formed an own surface reconstruction community in the last decades. Following Cazals et al. [11], Delaunay based surface reconstruction methods can be classified in different categories:

Tangent plane methods - These methods derive a local triangulation around a sample point by using its surface normal vector. The idea behind these methods is that point neighbors are likely to lie in short distance to the surface tangent plane. This assumption only holds if the surface is sufficiently sampled. The tangent plane then can be approximated by the fact that the Voronoi [5] cell is elongated in the direction of the surface normal. Methods that exploit this property are [9] and [29].

Restricted Delaunay based methods - These methods try to extract a surface by restricting the sample points to a subset. The Delaunay triangulation of this subset serves as the approximated surface. The crust algorithm [1], for example, creates a Delaunay triangulation from the point samples and their *poles*. A *pole* is a Voronoi vertex that is associated with a sample point and every sample point has exactly two *poles*. The triangles of the Delaunay triangulation of this union point set, that are not connected to any pole, become the initial approximated surface (the crust). By applying different filters on this initial surface they compute their final surface approximation.

Inside-Outside labeling - These methods classify each tetrahedron of the Delaunay triangulation as inside or outside the unknown surface. The set of triangles that separate inside and outside tetrahedra define the approximated surface. The Power Crust [3] improved the concept of the Crust [1] algorithm: There they compute a weighted Voronoi diagram for the poles and label each pole as inside or outside. The crust is defined as the set of Voronoi faces that separate inside and outside cells. The reconstructed surface, the so called power shape, is a subset of a weighted Delaunay triangulation that lie inside the crust. Inside-Outside based methods, where visibility information of the sampled points is given, often use energy minimization for surface reconstruction. They usually create a directed graph of the Delaunay triangulation where tetrahedra become nodes and faces that connect two tetrahedra become edges. The energy weights are set according to the visibility information. Finally a graph cut optimization labels tetrahedra as inside or outside and the surface can be extracted. Several methods use this kind of energy optimization [43, 20, 30]. The main difference between these methods is how they define the smoothness term and what post-processing they apply. They all use a global graph cut optimization which excludes them from being scalable. Nevertheless our approach is based on [30]. In a Divide & Conquer manner we generate multiple surface hypotheses for a all volumetric elements and fuse them to reconstruct the final surface.

Empty Ball methods - Bernardini et al. [6] introduced the ball pivoting algorithm (PBA). It does not directly work on a Delaunay triangulation, although it uses a similar property of localness - the empty ball. An empty ball bounds a sphere that circumscribes a simplex and does not contain any other sample points. Since all simplices in a Delaunay triangulation are local in this sense this algorithm highly relates to Delaunay methods. The algorithm directly works on the point cloud, thus it does not require any triangulation beforehand and the surface is built incrementally. At first it looks for a potential seed triangle which has an empty ball of radius r. From there on the ball pivots around an edge of the seed triangle while keeping the edge points on its boundary. If the ball hits an unvisited sample point it adds the triangle formed by this point and the endpoints of the edge to the surface. The newly added triangle is then a new starting point for the next pivoting operation. If no new triangles where added by the pivoting, the algorithm continues with a new seed triangle until all points have been considered. The selection of the ball radius r is crucial for this algorithm. If r is too small the reconstruction will contain holes. If it is too big some of the points will not be reached and features of the real surface will be missed. To overcome this the authors propose to analyze the point density beforehand and apply the algorithm with different radii for the empty ball, starting with the smallest one.

2.2. Scalable Surface Reconstruction

In this section we give a short overview of common scalable surface and/or multi resolution reconstruction methods. We focus on volumetric and Delaunay methods and how they handle vast scale differences within the point samples. At the end we discuss two current state of the art and scalable surface reconstruction methods we compete with in our experiments.

It has been shown by Kazhdan et al. [25] that a consistent isosurface can be extracted from any octree structure. But high scale variances generated by modern MVS systems lead to new challenges, thus the integration of multi resolution and scale becomes more and more prominent.

To incorporate multi resolution Mücke et al. [37] presented an iterative reconstruction algorithm. Initially, they put the sample points into a regular octree and determine the crust (a subset of voxels that contain the unknown surface) at a low octree level. By applying a graph cut they generate a surface within the crust where mesh vertices are only placed at a voxel's center. Afterwards they identify surface regions where the sampled details are to fine to be reconstructed properly at the current level. For these regions they form new crusts on higher octree level and reconstruct higher resolution surfaces. This is repeated until eventually all fine details are reconstructed. Finally, they fuse the meshes from all different reconstruction levels together to extract their final surface. Due to the fact that the energy formulation for their graph cut is global, their approach is not scalable but handles multi resolution well.

A local approach worth to mention was proposed by Kuhn et al. [28]. They fuse disparity maps created with SGM [19] into an octree voxel grid. Their novelty was a Total Variation based regularization term that allows pixel wise classification of disparities into different error classes. The uncertainties of the classes are then considered a-priori when fusing the disparity maps into a probabilistic voxel grid. These probabilistic grid is then transformed into a 3D point cloud and a surface is generated using the incremental local meshing method of Bodenmueller [8]. To corporate visibility constraints they cast a ray along the line of sight for a voxel for ten times the voxel size to filter occluded measurements. In their experiments they outperformed FSSR [13] which will be described in the next section.

2.2.1. Floating Scale Surface reconstruction (FSSR)

FSSR [13] is an implicit local approach using Gaussian basis functions to extract the zero level set from a point cloud containing normal and scale information. Its implicit function $F(x)$ is defined as a sum of weighted basis functions:

$$F(x) = \frac{\sum_i c_i \omega_i(x) f_i(x)}{\sum_i c_i \omega_i(x)} \qquad W(x) = \sum_i \omega_i(x) \tag{2.1}$$

Where the weighting function ω_i and basis function f_i are parameterized by ith sample's position p_i, normal and scale s_i. A confidence value c_i is also added that can be omitted by setting it uniformly to 1. For f they use a derivative of Gaussian for f_x and normalized Gaussian for f_y and f_z. The direction of f_x is set to the sample's normal while σ is set to the sample's scale. A polynomical function with compact support serves as weighting function ω. Following Curless et al. [12] it gives samples in front of the surface a high weight and falls of quickly behind the surface. ω is defined by a non-symmetric component in x direction and rotation invariant components in y and z direction. $W(x)$ is the weight function: the zero level set is only evaluated where $W(x) > 0$.

At first all samples are stored in an octree. According to the sample's scale value it has to be put into a (leaf) node that's side length meets the following criterion:

$$\frac{1}{2}s_i < S_l \leq s_i \tag{2.2}$$

Where s_i is the scale of the ith sample and S_l is the side length of the node (at level l). To meet this criterion the octree will be iteratively expanded until it is fulfilled. When all samples are inserted they make the octree regular by adding additional nodes such that all branch nodes have exactly 8 leaf nodes. Afterwards the implicit function is evaluated at the corner points of each leaf node. For each corner point x they recursively traverse the octree and check if a node N contains samples that influences the implicit function $F(x)$. The radius of a sample's support is defined to $3s_i$, thus from Equation 2.2 it can easily be omitted if a node contains samples that support $F(x)$. To avoid mixing high and low resolution samples that will degrade the iso surface [27, 13] a continuous cut-off heuristic is used: A sample is only considered if $s_i < s_{max}$ where:

$$s_{max} = \frac{1}{5}\sum_i s_i \tag{2.3}$$

Where the ith sample is affecting the implicit function at position x. From this point on the original samples are of no further use. They extract the final isosurface with the surface extraction algorithm proposed by Kazhdan et al. [26]. It is a Marching-Cubes-like [34] algorithm that yields a crack free mesh from an octree hierarchy. Finally, they apply a simple cleanup procedure to remove degenerated triangles that on average leads to a triangle reduction of 40%.

They also evaluated their approach on a wide range of datasets with impressive results. Due to their local problem definition, their algorithm is well suited for mixed scale datasets. One drawback, due to their local nature, is the unability to deal with mutually supported outliers.

2.2.2. Global, Dense Multiscale Reconstruction for a billion points (GDMR)

GDMR [41] is a variational global approach for surface reconstruction of a point cloud with normal and scale information. The samples of the scenes are stored in an octree. Different to our approach, a leaf node of a certain depth only stores information of point samples of the corresponding input scale. The lower the scale of a sample the deeper the leaf node it will be assigned to. After the creation of this unconstrained octree they apply a balancing criterion where the depth difference of adjacent leaf nodes must be at most one depth level.

For every node in the octree they aggregate all points within a window to form 10 normal cluster centres f_n and their weights ω_n. In addition they store the signed distance values for each point and their weights into a histogram with 8 bins, g_m and ω_m respectively. This results in a compact memory footprint of 64 bytes for every node (including fields for average color information). From this point on the original point cloud is of no further need.

The following energy is then minimized:

$$E(u,v) = \lambda_1 E_{data_u} + \lambda_2 E_{data_v} + \alpha_1 E_{coupling} + \alpha_2 E_{smooth} \tag{2.4}$$

Where u is the signed distance function $u(x)$ and v is the normal vector field $v(x)$ and $x \in \mathbb{R}^3$. $\lambda_1, \lambda_2, \alpha_1$ and α_2 define the relative importance of the terms. For a detailed description of the terms we refer to [41]. They also include a scale function s in E_{data_u} and E_{smooth} make the energy function aware of the reconstruction scale which is worth mentioning.

After discretizing the problem by using various function approximations of $u(x)$ and $v(x)$ they compute the signed distance function u. To generate a surface from the signed distance function they use the dual contouring algorithm proposed by Ju et al. [24]. In addition the surface orientation from the vector field v are used to compute/improve the vertex positions. This global approach performed highly competitive in terms of robustness and completeness on various datasets with high scale variance [41].

3. Base Approach

Our surface generation method is based on the work of Labatut et al. [30]. In this section we describe how this method exactly works. It is a surface generation technique that is based on Delaunay triangulations and graph cuts. As input it uses a point cloud that contains visibility information. At first it generates the 3D-Delaunay triangulation of this point cloud. During the optimization process it labels a Delaunay tetrahedron as empty or occupied. This is done by applying an energy minimization algorithm that takes the visibility information into account. The set of triangles that separate empty tetrahedra from occupied are finally extracted and represent the surface.

3.1. Delaunay Triangulation

Every triangulation of a discrete point set of any dimension is a subdivision of its convex hull into simplexes. A simplex is a generalization of a tetrahedral region of space to n dimensions. A $0-$simplex represents a vertex, a 1-simplex an edge, a 2-simplex a triangle and a 3-simplex represents a tetrahedron. The set of simplexes of any triangulation are always intersection free [5].

The Delaunay triangulation is a special and well researched kind of triangulation where every simplex fulfills the Delaunay condition. In the case of 3 dimensions a simplex is Delaunay if there exists a circumsphere such that no vertices lie inside the circumsphere, except the vertices of the simplex itself. A simplex is strongly Delaunay if it is Delaunay and there lie no other vertices on the circumsphere. For the 2D case a circumcircle serves for this condition check, an example is plotted in Figure 3.1. If all simplexes are strongly Delaunay the Delaunay triangulation is unique. In fact, this means that the point set is in general position. In the case of 3 dimensions, this also means that there are no 5 points that are co-spherical (co-circular for 2 dimensions). If the point set is not in general position it is called degenerate and its corresponding Delaunay triangulation is not unique.

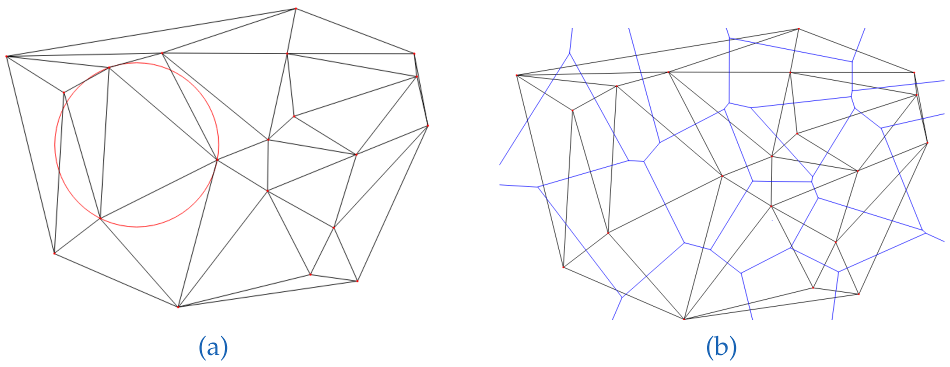


Figure 3.1.: The Delaunay triangulation of 20 randomly sampled points. In (a) the red circle visualizes the Delaunay condition for a single triangle. It does not contain or touch any other vertices except the vertices of the triangle it encloses. In (b) we additionally plotted the Voronoi diagram.

More generally a Delaunay triangulation of a point set of dimension n is a projection of a lower convex hull. To form the convex hull the points are transformed to the $n+1$ dimension by applying a parabolic lifting. For the 2D case the paraboloid $z^2 = x^2 + y^2$ is used form this transformation. For every point $P = (x, y)$ in the point cloud its vertical projection on this paraboloid is $P^* = (x, y, x^2 + y^2)$ in 3D. Consider S as the set of points in a 2 dimensional space and S^* as the projection of S on the paraboloid. Then the projection of the lower convex hull of S^* back into 2D determines the Delaunay triangulation in 2D. In Figure 3.2 an example for this parabolic lifting is visualized.

The Delaunay triangulation is also closely related to the Voronoi [5] diagram. A Voronoi diagram is a spatial segmentation of an n-dimensional space into piecewise linear cells. Every point in the point set has its own cell, which is called the seed of a cell. At any point in a cell its distance to the seed of the cell is shorter than to any other seed. If the point set is in general position the Delaunay triangulation is the dual graph of the Voronoi diagram. For the case of 2 dimensions a Voronoi can be easily converted to Delaunay triangulation. If two cells share a border segment their seed cells are connected by a Delaunay edge. An example of a Voronoi diagram and its dual Delaunay triangulation can be found in Figure 3.1.

The Delaunay triangulation has some nice properties for surface reconstruction.

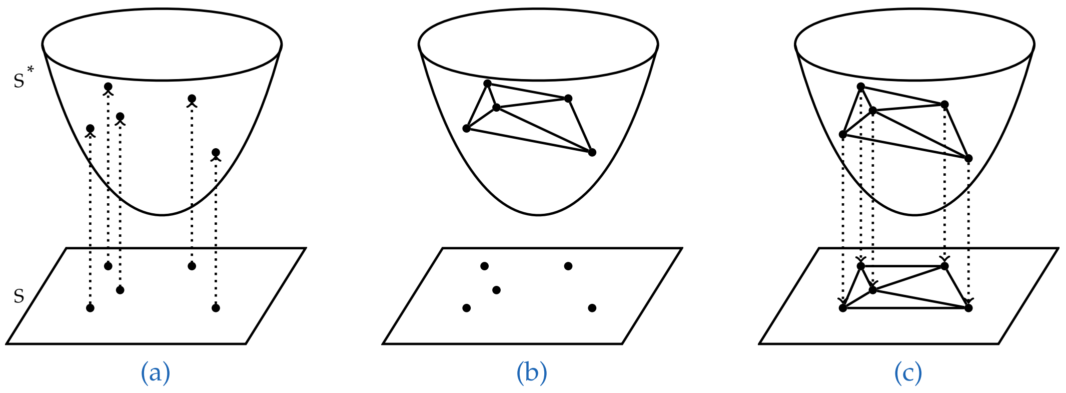


Figure 3.2.: Parabolic lifting example in 2D. In (a) you can see the projection of the 2D point set onto the 3D paraboloid. S represents the points in a 2D space, S* represents their corresponding projections onto the paraboloid. (b) shows the lower convex hull of S*. (c) illustrates the back projection of the lower convex hull in 3D into 2D and its yielded Delaunay triangulation for S.

It has been shown that, if a point set is adequately dense and accurate, a good approximation of the surfaces can be found in the subset of the Delaunay triangulation [1]. The algorithmic complexity of the Delaunay triangulation in 2D is $O(n \log n)$ and $O(n^2)$ in 3D. But if the points are distributed on smooth surfaces it has been shown that the complexity drops from $O(n^2)$ to $O(n \log n)$[4]. One drawback of the Delaunay triangulation is that it is not unique if the points are degenerate. However, there are implementations like CGAL [40] that always yield an unique Delaunay triangulation, even for this case. In terms of scalability we have to consider that for any Delaunay triangulation in 3D, the number of tetrahedra can be upper bounded by $\mathrm{O(n^2)}$ and is not known in advance [5].

For our surface reconstruction algorithm we use CGAL for the creation of the 3D Delaunay triangulation. It does not only deliver an unique solution for the degenerate case, but it also incorporates the concept of the infinite vertex. In addition, this causes the presence of infinite cells and faces in the Delaunay triangulation. These are essential for our dual graph generation for our energy minimization.

3.2. Energy Minimization using Graph Cut

As its name indicates, a graph cut is a global energy minimization method that works on a graphs. The graph $G = (V, E)$ has to be finite and consists of vertices (V) and directed edges (E). Also, each edge has a non negative weight which is called capacity. The graph has two special vertices the source s and the sink t that are called terminals. Someone can image this graph as a energy flow network where the source is the only vertex that can create energy and the sink is the only vertex that can absorb it. The goal is to find the maximum energy flow from the source to the sink in this network. Which is equivalent to finding the bottleneck of the network which is called the minimum cut. A cut C is a set of oriented edges that separates the vertices into two disjoint sets, where the source and the sink are always separated. The costs $|C|$ of a cut is equal to sum of its edge weights. There might be more than one possible minimum cut in such a network. Graph cuts can be used for binary labeling tasks such as image segmentation in computer vision. In this project we use the algorithm of Boykov et al. [10] to find the minimum cut C. It has a theoretical worst case complexity of $O(VE^2|C|)$, but it has near linear complexity in practice. The original implementation allocates $24|V| + 14|E|$ bytes [32]. An example of a network flow graph and a minimum cut can be found in 3.3.

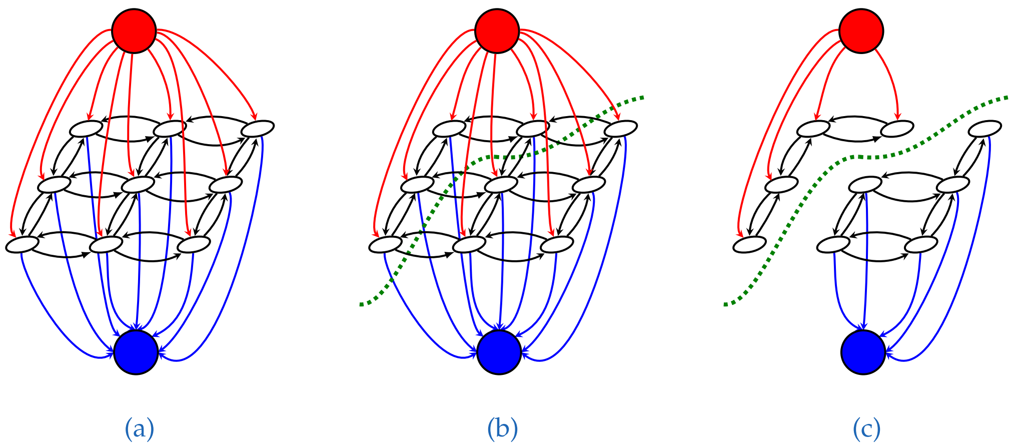

(a) (b) (c)

Figure 3.3.: An Example of a minimum cut in an energy flow network. The red node is the source and the blue node is the sink. In (a) you can see a simple flow network. In (b) a minimum cut is visualized (green dotted line). (c) shows the flow network reduced by the edges of the minimum cut, which leaves two disjoint sets. One is connected to the source, the other to the sink.

3.3. Energy Formulization

This section is mainly based on the work of Labatut et al. [30] with some minor adaptations from Mostegel [35]. The Delaunay triangulation of the point cloud and their visibility information (from which viewpoint a point was seen from) serve as input. The main concept of Labatut et al. [30] algorithm is to utilize the visibility information of the point cloud to approve or disapprove if a facet in the triangulation is part of the surface. As already explained graph cuts can solve binary labeling problems. At first we have to convert the task of surface extraction in a Delaunay triangulation to such a binary labeling problem. This can be done by considering that a tetrahedron can be *inside* or *outside* an object. After labeling all tetrahedra as *inside* or *outside* we can extract the surface by gathering those faces (2-simplex) that are shared by tetrahedra with different labels. Thus the goal is to separate the tetrahedra into two disjoint sets which exactly matches a binary labeling problem.

The next step is to generate a energy flow network graph and set its weights accordingly that can be solved by a graph cut. To do this we create the dual graph of the Delaunay triangulation. A tetrahedron becomes a node and a face that is shared by two tetrahedra becomes two directed edges that connects these nodes. This graph coincides with the Voronoi diagram of the Delaunay triangulation, except that the edges are orientated. In addition we add links from the source to the vertices and links from the vertices to the sink. These links do not have any corresponding simplices in the Delaunay triangulation, they are only needed for the flow network graph generation. To obtain a surface $\mathcal{S}$ we have to minimize the following energy terms, one for the visibility the other for the smoothness of the surface:

$$E(\mathcal{S}) = E_{vis}(\mathcal{S}) + \alpha \cdot E_{smooth}(\mathcal{S}) \tag{3.1}$$

where $E_{vis}(\mathcal{S})$ is the data term and represents the penalties for the visibility constraint violations. $E_{smooth}(\mathcal{S})$ is the regularization term and is the sum of all smoothness penalties across the surface. α is a balance factor between the data and the regularization term and thus it controls the degree of smoothness. In Figure 3.4 we plotted an example for the construction of the flow network. We use the same example for the whole data term description.

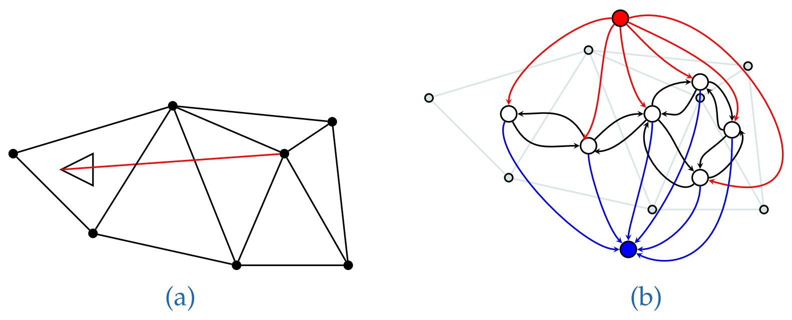


Figure 3.4.: In (a) we plotted a small point cloud and a camera. The red line represents the ray from the camera center to a seen measurement point. In (b) we plotted the corresponding flow network, with sink and source, of this point cloud as described in the energy formulization section.

3.3.1. Data Term - Visibility Information

This term serves to incorporate the visibility information into the flow network. Every vertex of the point cloud keeps its visibility information. This information is crucial and thus it has a major influence when designing the data term. If a vertex is part of the final surface $\mathcal{S}$ it should be visibly from the positions it was captured from. Hence all the tetrahedra (nodes) that are intersected by a ray from any capturing position to the vertex should be labeled as *outside* and the tetrahedron behind it should be labeled as *inside*.

At first we have to connect the terminal nodes to the graph. A tetrahedron is set to be *outside* by setting the weights from the source to it accordingly. Labatut et al. [30] recommend to define the tetrahedra where the capture positions are in as *outside*. They proposed a hard constraint for this connection and an infinity weight (λ_∞) is set. Thereby the graph cut will always assign this tetrahedron as outside, it cannot change its label. In [31] they proposed a soft constraint. Here a finite weight is added to the connection. This allows tetrahedra where capturing position to be labeled as *inside*. Labeling such tetrahedra as *inside* sounds not reasonable thus in our implementation we only considered the approach with the hard constraint. An example of this outside labeling can be found in Figure 3.5.

Mostegel [35] showed in his experiments that under certain circumstances this definition of *outside* may causes the scene to become unstable and fall apart. To

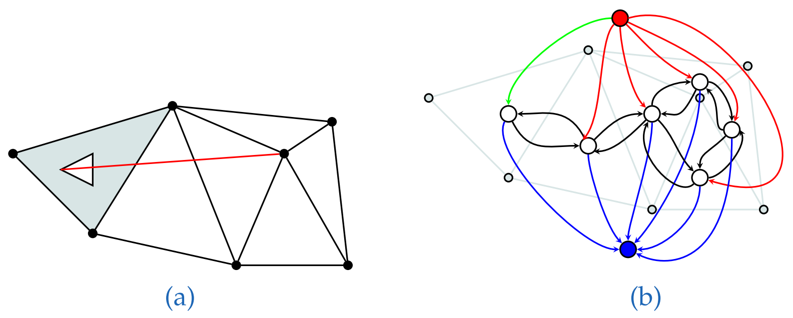


Figure 3.5.: In (a) we have highlighted the triangle the camera lies in. In (b) the green edge represents the connection in the flow network the outside labeling has impact on. In this case this edge gets the weight λ_∞ assigned.

avoid this undesired behavior they suggest assigning also all infinite tetrahedra to *outside*. An infinite tetrahedron does not physically exist, it always shares a facet of the convex hull of the Delaunay triangulation and is connected to the infinite vertex. As a result three of four facets of this tetrahedron do not exist and are artificial. The infinite vertex is a special vertex in the Delaunay triangulation that also does not have a real location and per definition it always lies outside the convex hull.

Defining every tetrahedra outside the convex hull as *outside* cannot invalidate the surface reconstruction. This is due to the fact that the surface we have to reconstruct always lies inside or on the convex hull. This adaption only has an impact on parts of the scene we have no information about.

For inside labeling finite weights have to be used otherwise the energy minimization would be weakened. Consider every captured measurement as a vote for the surface. Usually a set of measurements always contains outliers. If we use infinite weights for the inside labeling the outliers will have a big impact on the reconstructed surface and it will become noisy. Our goal is to extract a surface as noise free as possible. To set the weights accordingly we use a simple technique: We cast a ray from the capture position of the measurement to the position of it. Afterwards we add a finite constant weight to the connection from the tetrahedron, that lies behind the measurement in the the direction of the ray, to the trunk. Figure 3.6 shows an example of this concept.

After setting the initial inside and outside label weights for the tetrahedra we need to consider the empty space criterion. Some facets of the Delaunay trian-

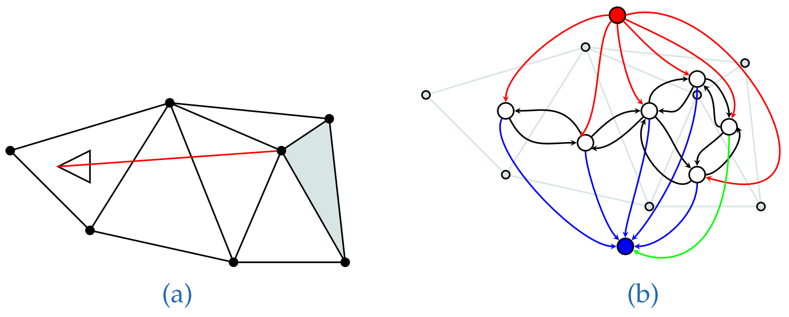


Figure 3.6.: In (a) we highlighted the triangle that lies behind the measurement. In (b) the green edge connects this triangle to the trunk in the flow network. A finite fixed weight is added to this edge in the flow network.

gulation cannot be part of $\mathcal{S}$ because the would occlude other measurements that were captured from several camera positions. This property is handled by penalizing ray conflicts. To do this we follow a ray from the capturing position to a measurement. Each facet we intersect is then updated in our flow network. Since every facets exists twice in the flow network, one for each orientation, we only update those that are crossed from the inside to the outside.

Labatut et al. proposed two concepts of calculating the weights for the ray conflicts. In [30] they suggest to add a fixed weight for these intersections. In a later work [31] they decided to use weights depending on a confidence parameter σ and the distance d of the ray intersection to the measurement. They used this concept to be able to deal with Gaussian scene noise. σ represents the estimated noise of the measurement, the final weight is then composed by the following formula:

$$w_{viz} = \alpha_{viz}(1 - e^{\frac{-d^2}{2\sigma^2}}) \tag{3.2}$$

Here, α_{viz} is a fixed penalization factor that controls its impact on the surface. In this work we also use this formula to obtain the penalization weights for the ray crossings. An example ray crossing penalization is shown in Figure 3.7.

3.3.2. Regularization Term

Regularization is essential for this method. If in a flow network an edge does not have any capacity its equivalent facet cannot be part of the solution. An

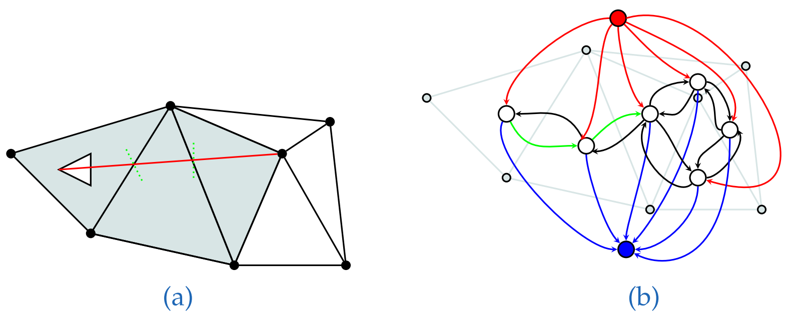


Figure 3.7.: In (a) we highlighted the ray crossings (green dotted lines). In (b) we marked the corresponding edges that are penalized.

edge without capacity is a non existing edge in the flow network thus it will never be part of the minimum cut. This means that a facet can only be part of the solution if it has a penalty assigned. Furthermore, this means that we require at least one measurement that lies behind this facet that that causes a ray crossing penalty for it.

To fulfill this requirement Labatut et al. [30] encourages smoothness by minimizing the area of the overall surface. The corresponding weights are calculated using the following formula:

$$E_{area}(\mathcal{S}) = \sum_{T \in \mathcal{S}} area(T) \tag{3.3}$$

Here T is a triangle in the Delaunay triangulation. This term can easily be incorporated into the flow network. We simply have to calculate the regularization term for a facet and update its corresponding edges (for both directions) in the network. This kind of regularization manages to exclude large facets that connect to a measurement outlier to be part of the surface. If we have a highly sampled surface (with a lot of low noise measurement points on it) it is most accurately described by a large set of small facets. Using the area of a facet for regularization causes the denial of this set to be part of the surface. It is very likely that we can find a big triangle close to the front to this surface that has a lower area than the accumulated area of the set. Thereby the energy minimization might assign this big triangle to be part of the surface. Thus regularization with area minimization is not well suited for a reliable and accurate surface reconstruction.

In [31] they overcame this problem by using an generalization of the β-skeleton that was proposed in [2] for curve reconstruction in 2D. The β-skeleton is a subset of Delaunay edges in the 2 dimensional case. To determine if an edge is part of the β-skeleton we have to test the circumcircles of its adjacent triangles. If both circumcircles are centered on opposite sides of the edge and both have a radius greater than $\beta/2$ times the length of the edge it is part of the β-skeleton. If the samples are dense enough this subset of edges outputs a correct reconstruction. Unfortunately the β-skeleton does not generalize well to 3 dimension. Close to a surface we can have flat tetrahedra that have small circumspheres thus the β-skeleton may introduce holes. Labatut et al. [31] introduced a generalization for the β-skeleton in the 3 dimension case and they fused an approximation of it into their energy optimization. For a given facet in the Delaunay triangulation they analyzed the circumspheres of the adjacent tetrahedra. Each circumsphere intersects with the plane of the shared facet at an angle φ and ψ. They used these angles to compute a weight:

$$\omega_\beta = 1 - \min(\cos\varphi, \cos\psi) \tag{3.4}$$

This weight is then added to both facets in the flow graph, one for each orientation. As a result a tetrahedron with a large empty circumsphere gets a low penalty and it is more likely that it is being cut by the energy minimization. This behavior is very welcome because it is also more likely that it belongs to the surface.

The last variant of regularization, and the easiest to think of, is to penalize all facets with a small finite weight. Mostegel [35] introduced this in their reimplementation. In their various different experiments it turned out that this simple regularization outperforms all other types of regularization in terms of accuracy. They give credits for this result to the Minimum Description length (MDL) principle. The conclusion of the MDL principle is that the best hypothesis is the one which leads to the best compression of data. Transfered this principle to surface reconstruction means that the best surface is the one with the fewest number of facets, which can be accomplished by assigning small constant penalties for all facets.

As a conclusion of Mostegel [35] experiments they showed that the highest accuracy can be achieved by using the hard visibility constraint and a constant regularization term. Therefore we use this setup for our surface reconstruction.

4. Method

With respect to scalability, the goal of our approach is to process a huge number of 3D measurements that contain visibility information, and produce a surface from it. Our approach is designed as a process-pipeline with 3 consecutive steps. In the first step we store the measurements in an octree where every leaf node is restricted to hold a maximum number of measurements. Afterwards we de-noise these measurements by using a simple k-nearest method and put them into a new octree.

In step 2 & 3 we reconstruct the surface from the de-noised octree. Also, from step 2 to 3 we decrease the required certainty for a triangle to be part of the reconstruction. Where in step 2 the extracted triangles have a 100% consensus, the applied triangles in step 3 are ambiguous. Which means that we increase the approximation of the final surface in each step.

To make it scale we divide the octree into tasks. A task is a set of leaf nodes that fulfill a neighborhood criterion (which we describe later). These tasks can be processed in serial or parallel. For our approach it is essential that a leaf-node can appear in more than one task: In step 2 we use the surface reconstruction algorithm of Labatut et al. [30] to create a surface hypothesis for every task. Thus, for a node, we have as many surface hypothesis, as it has been part of different tasks. The main idea behind our approach is to pick those surface hypotheses (or parts of it) that approximates the real, unknown surface well. Our final reconstructed global surface is a combination of many different reconstructed local ones.

After generating a surface hypothesis for every task we compare these hypothesis node-wise. We gather all triangles that have a 100% consensus (are present in all hypotheses for a node) and add them to our final surface. This final surface is self intersection free but it usually contains holes and gaps. The holes are caused by the nature of Labatut et al. [30] global optimization method: The energy terms for the optimization depend on the leaf-nodes within the task. If a leaf-node appears in more than one task, every task it appears in has different energy terms. This global energy minimization may not result in the same node surface hypothesis for every task. Thus a 100% consensus for a surface hypothesis for a node is very unlikely. But there is a high chance that parts of the different surface hypotheses are equal.

The purpose of step 3 is to minimize the length of the overall boundary of the final reconstructed surface. In this step we consider triangle patches, we call them patch-candidates, that got rejected by step 2 to be part of the final surface. The step is split into two parts: 3a and 3b. In step 3a we rank these patch-candidates by using a cost function and add them if they fully patch a hole in the final surface. In step 3b patch-candidates have not to fully patch a hole: An energy minimization is applied to the to the patch-candidate that minimizes the boundary length of a hole.

In Figure 4.1 an overview of the different processing steps can be found.

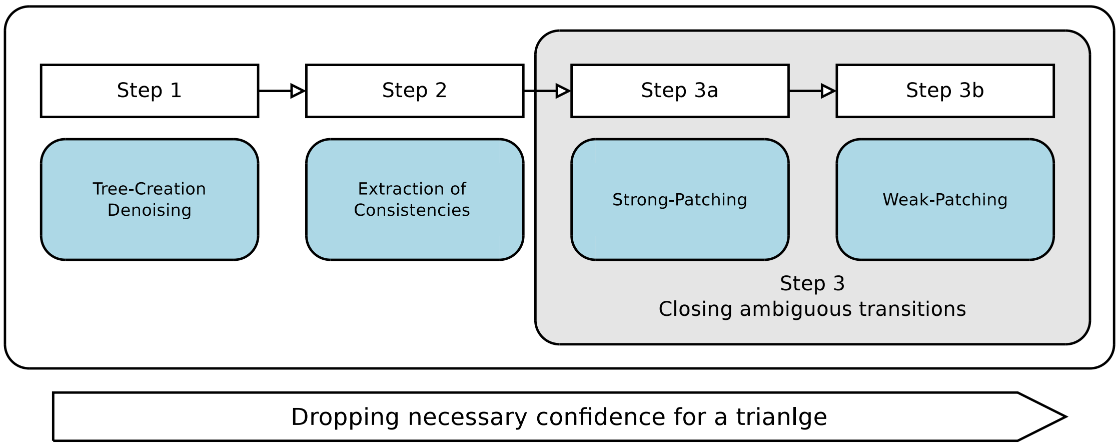
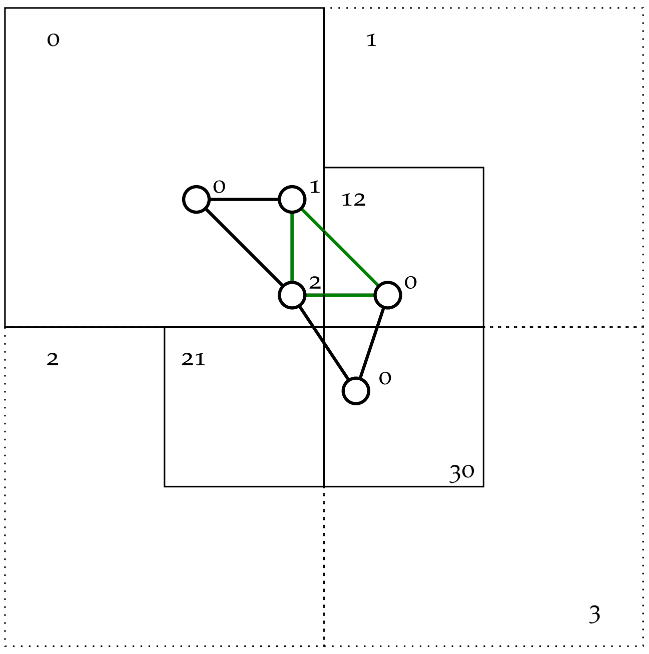

Step 1
Step 2
Step 3a
Step 3b
Tree-Creation Denoising
Extraction of Consistencies
Strong-Patching
Weak-Patching
Step 3
Closing ambiguous transitions
Dropping necessary confidence for a trianlge

Figure 4.1.: System overview of our approach

Surface Topology

Our generated surface is also stored in an octree and it contains vertices, edges and triangles. A vertex is uniquely defined by a pair of values, its leaf ID and an unique vertex ID within the leaf. An edge always contains two vertices it connects and in addition we store all adjacent triangles the edge is part of. We define that an edge can only be part of two triangles (local 2-manifoldness). At last, a triangle is identified by its three vertex IDs, furthermore we store its orientation using a normal vector. If the vertices of a triangle lie in different leaf nodes, the triangle, its edges and vertices are present in these leaf nodes too. This duplicate information is necessarily to check if a triangle is fully connected to an adjacent node globally. A visualization of this topology can be found in Figure 4.2.



Vertices:
{0},0
{0},1
{0},2
{12},0
{30},0

Edges:
{0},0 - {0},1
{0},0 - {0},2
{0},1 - {0},2
{0},1 - {12},0
{0},2 - {12},0
{0},2 - {30},0
{12},0 - {30},0

Triangles:
{0},0 - {0},1 - {0},2
{0},1 - {0},2 - {12},0
{0},2 - {12},0 - {30},0

Figure 4.2.: Surface topology example for 2 dimensions. Here you can see a quadtree with the leaf nodes 0, 12, 21 and 30. At the right side of the figure the extract vertices, edges and triangles with their IDs are plotted. A vertex is identified by its node-ID in {} and its unique vertex-ID within the node. As an example the green highlighted triangle and its simplices are present in the leaf nodes 0 and 12.

4.1. Step 1 - Data Fusion & Task Creation

4.1.1. Octree

As already mentioned scalability is one of the main concerns of this work. Due to the lack of unlimited memory and the requirement that the whole method has to run on a common personal computer we have to chunk the data into manageable parts. These parts then can be processed sequentially or in parallel depending how much computational capabilities and memory are available. A data structure that fulfills these requirements is an octree. An octree is a tree based data structure that provides partitioning and searching operations. The tree always has a root node and every node represents a 3D cube. A branch node has 8 child nodes that subdivide it. For this subdivision the cube is separated into 8 equally sized cubes. This subdivision can be done recursively up to any tree depth. An illustration of an octree is shown in Figure 4.3. For our system every leaf node in this tree is limited to hold a defined maximum number of 3D points. Limiting the number of measurements a leaf-node can hold affects the overall number of leaf-nodes, thereby maintaining scalability for further data processing.

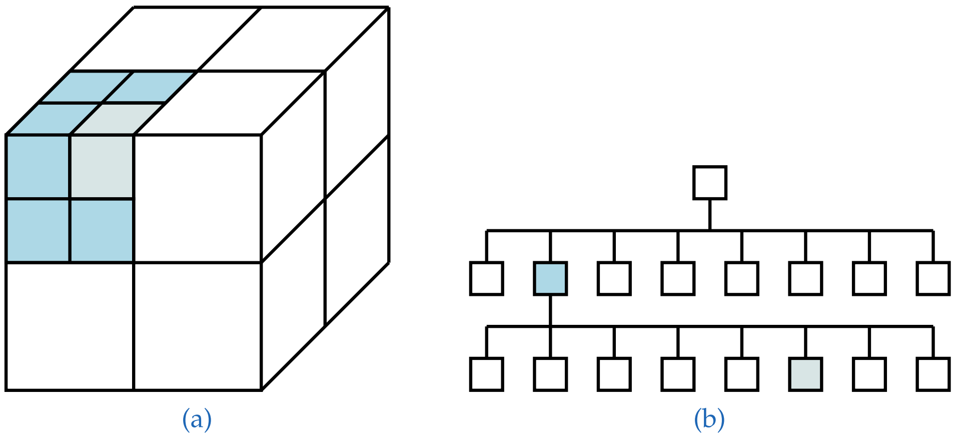


Figure 4.3.: An example subdivision of space and its corresponding octree. A branch node (blue) and a leaf node (gray) and their equivalents are highlighted in both figures.

4.1.2. Data Fusion

Our input point clouds can come from various different sensors. They may contain misaligned duplicate measurements and some noise. To fuse our input point clouds we first put them into an octree. For every leaf node in this tree we gather all leaf nodes that lie within a radius. This radius depends on the size of the actual leaf node. If the number of gathered leaf nodes exceeds a certain number and we filter them. To do this we calculate the center of mass for every gathered node. Afterwards we sort them by the distance of their center of mass to the center of mass of the actual node. We skip leafs with the highest distance until the scalability constraint is fulfilled. The remaining leafs and the current leaf are put into a kd-tree.

We randomly choose a measurement from the current node and start fusing it. We gather the k-nearest measurement that lie within a radius. The search radius depends on the ground sampling distance of the measurement. We calculate the mean position and the mean normal for this measurement set. Afterwards we apply the mean normal to the randomly chosen measurement. After that we project the mean position onto the randomly chosen measurement using its applied normal. A fusing example is visualized in Figure 4.4. Also, the fused measurement gets all visibilities, from all measurements in the search radius, applied.

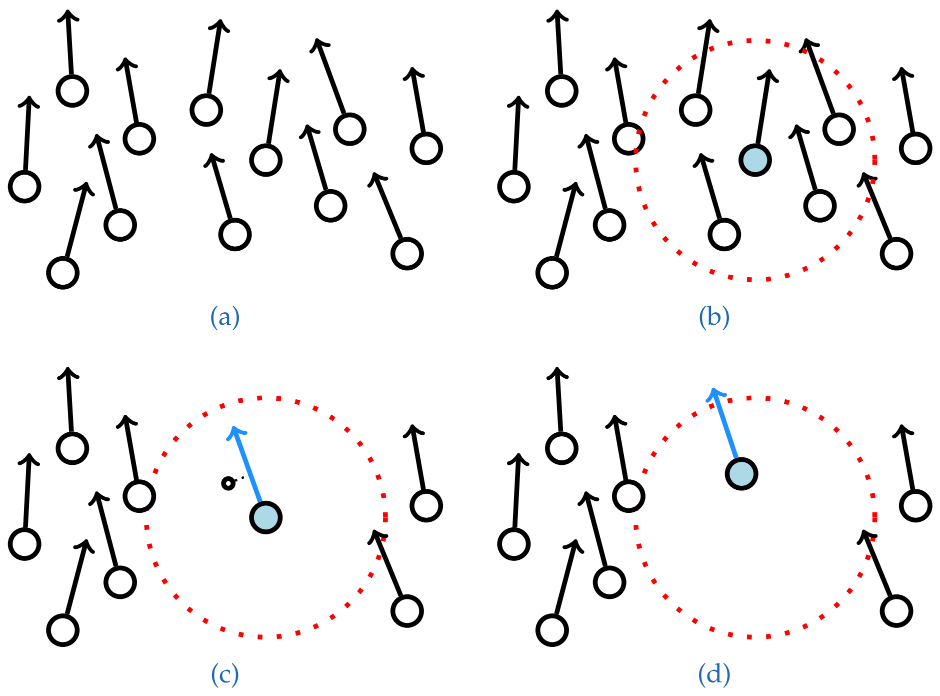


Figure 4.4.: In (a) we see the original measurements and their normals. (b) shows the chosen measurement and its search radius-circle. The k for the k-nearest search is set to 5. There lie 4 other measurements within the search radius. In (c) we apply the average normal to the chosen measurement. We also plotted the projection of the mean position onto the applied normal. (d) shows the new fused measurement and its normal.

After we finished all leaf nodes we put the fused data into a new octree. This tree will be used for task creation and further processing.

4.1.3. Task Creation

In our system a task is always a set of nodes in the octree. All nodes in a task have to fulfill a neighborhood condition. We define that this condition is fulfilled when all nodes in a task are spatially adjacent. In addition the octree has to be balanced: If the node size difference between two adjacent nodes is too big, we split the bigger node until this constraint is not violated any more. This avoids the reconstruction of enormously big triangles that span from big to smaller nodes. The maximum node size difference of adjacent nodes we allow is two octaves. An unbalanced quadtree and its balanced equivalent can be seen in Figure 4.5.

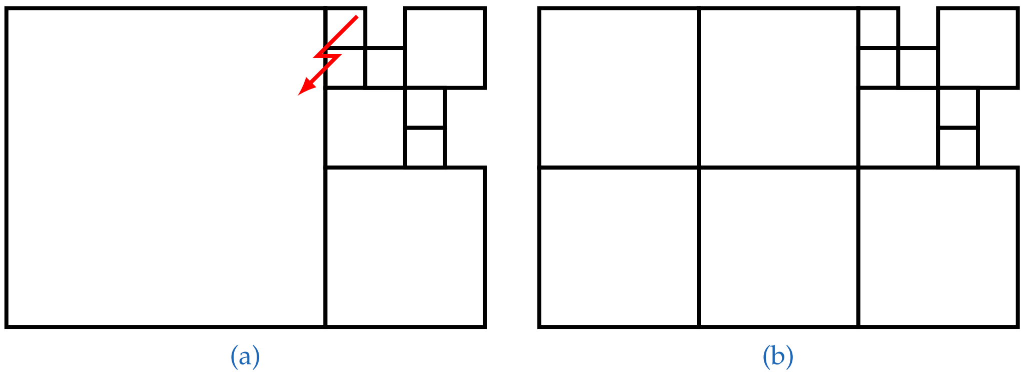


Figure 4.5.: An unbalanced (a) and balanced (b) Quadtree.

To extract those tasks we gather all corner points of the 3D cubes that are represented by the octree. For the neighborhood condition we check if a corner point is adjacent to any other 3D cube. We do this by creating a virtual cube that is centered at this corner point and has half the dimension of the smallest cube in the octree. Afterwards we estimate in which nodes of the octree the corner points of this virtual cube lie in. This set of nodes form a potential task. To minimize the total number of tasks we eliminate duplicates and those that are fully contained in a different task. This reduced set of tasks represents the final one. An example of the neighborhood condition in 2D is shown in Figure 4.6.

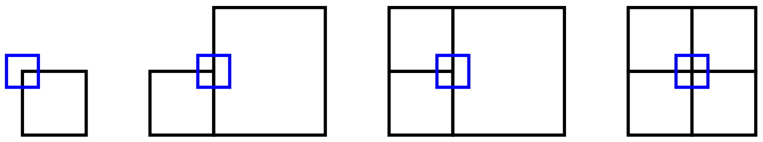


Figure 4.6.: The neighborhood condition in the 2D case. The black squares represent the nodes in a Quadtree (2D equivalent of the octree), the blue squares symbolize the virtual squares that are placed at the corner of a node. For better visualization only one virtual square is shown per task. For 2D there are up to 4 nodes in a task, in the 3D case there are up to 8.

4.2. Step 2 - Collecting Consistencies

For every task we create a surface hypothesis by using the approach of Labatut et al. [30]. Thus we have as many surface hypotheses for a node as it has been part of different tasks. We assume that these hypotheses highly coincide if the point set in a node is dense enough. With this assumption we firstly extract all facets for a node where all hypotheses exactly match. This set is our initial surface and it is unambiguous. If all surface hypotheses for a node have a consensus in certain parts of the surface, then these parts have the highest possible probability to approximate the real and unknown surface well. In a second iteration we look at facets that span through two nodes, we call them 2-facets. During the surface generation of the tasks beforehand we also kept track of how often a node pair was part in a task. If a 2-facet appears in all tasks that contain its node pair and if it is a *definite*-facet, it is selected for being part of the surface. We define a *definite*-facet as a Delaunay facet that would also be present within a task that contains all nodes. We can determine if a facet is a *definite*-facet by testing the two tetrahedra it is part of. A tetrahedron in a Delaunay Triangulation is definite if we can guarantee that there will not be any vertex added that lies within the circumsphere of the tetrahedron. We can test this by distance checks within the task and the spatial information of the octree. If a 2-facet is part of at least one definite tetrahedron,it is a *definite*-facet. This condition serves that the set of 2-facets is 100% self intersection free.

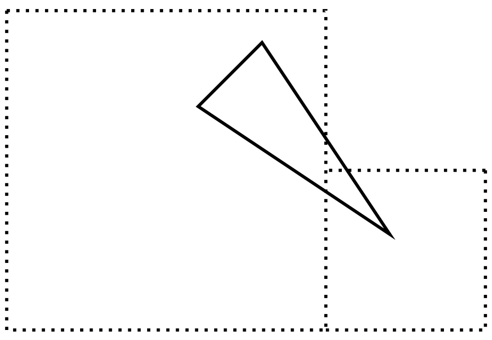
4.3. Step 3 - Closing ambiguous transitions

As our initial surface will always have some holes in it we need a strategy to patch them. Holes will likely be present at border regions of our tasks: Labatut et al. [30] surface reconstruction algorithm will always produce a closed surface. Facets that lie at the border of a task will always be adjacent to a facet that connects it to a point behind the surface to ensure that it is closed. This always results in a hole if there is not any other task that connects these two borders with *definite*-facets. In addition, we will always have holes within a node where its containing tasks do not produce the exact same surface, there it is ambiguous. If there are more different hypotheses for a part of the surface neither hypothesis is chosen which results in a hole. To overcome this problem we introduced a patching mechanism that uses a score function for the various different patch candidates.

4.3.1. Patch-Candidate Extraction

This action will always be performed within a task. At first, we load the initial surface results, created by Step-2[4.2], for a task. Secondly, we remove those triangles that are present in both reconstructions (the task and the initial result). After that, we remove all triangles that share an edge with an initial surface triangle that is 2-manifold. As the leaf nodes in a task have not necessarily the same size we also remove those triangles that reach through the task boundaries as visualized in Figure 4.7a. In a last step we remove all triangles of the task that intersect with triangles of the initial surface. Point intersections of triangle vertices are excepted. The first two steps can easily be done using our surface topology, where for the third step we had to use intersection tests in 3D. We achieved this by using CGAL [40] axis aligned bounding box (AABB) library. This library allows us to perform efficient intersection and distance queries against a set of finite 3D geometric objects. In this set of reduced task triangles we look for patch-candidates.

We cluster all remaining triangles that are edge-connected in linear time to find patch-candidates. Vertex connections are ignored, see Figure 4.7b. The minimum number of triangles in a patch-candidate is one, theoretically a patch candidate can consist of all remaining triangles if they are edge-connected.


(a)

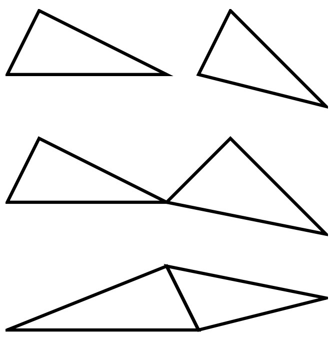

(b)

Figure 4.7.: In (a) a triangle where its three vertices lie inside a task is shown. One edge of the triangle intersects with the border of the task thus it cannot be part of a patch-candidate and it is ignored. In (b) you can see how triangles can be connected. The first row shows the unconnected case, the second shows the vertex-connected case. The last one shows the edge-connected case that is considered for our patch-candidate extraction.

4.3.2. Patch-Candidate Score Calculation

After the extraction we have an unordered set of patch-candidates. One should think that all candidates have the same quality for hole patching if they only fit perfectly into a hole. Unfortunately this is not the case, especially for patch-candidates that reach through two or more leaf nodes. The situation that we describe was observed during the execution of our early experiments and is a worst case scenario: If a patch-candidate, that mainly consists of backside triangles, gets chosen before a front-side patch-candidate, a hole would possibly be wrongly patched. This also denies any other correct patch-candidate, once a hole is patched it is final and assumed to be correct. To solve this problem we applied a score to each candidate which depends on its spatial position within the task. The implemented score function follows a heuristic: The greater the distance of a patch candidate to the task center, the lower its score should be. The arithmetic mean of all 3D-point simplices of a patch-candidate serves as its point representation. For the score calculation we measure the distance of the patch-candidate to different chosen *center* points within the task. For a small example see Figure 4.8.

We distinguish between four candidate-point distances: The first one is its distance to the center of a node within the task. If a candidate lies at the center,

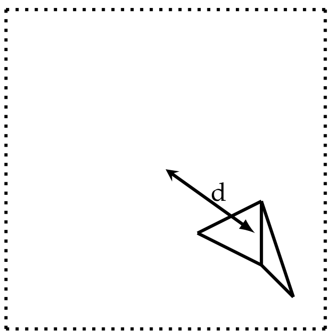


Figure 4.8.: In this figure we plotted a leaf node and a patch-candidate within in. In this example the distance for the score calculation is measured from the arithmetic mean of the candidate to the center of the node.

its score should be at a maximum. The other ones are based on point distances in an adjacent node setup.

In a task adjacent nodes are connected pair-wise, they share a plane. The center of such a shared plane serves as another point-distance, we call this distance plane-distance. A patch-candidate can lie next to its node border but within the hole task it does not. We consider this case by using the plane-distance for two adjacent nodes.

For the same reason we introduced the edge-distance. If we have an edge in the task that is shared by more than two nodes and the edge is inside the task (it is not part of the boundary), we add the center of this edge as edge-distance point.

In addition, if we find a node corner point in the task that is shared by at least five nodes, and again, the point lies at the inside of the task, this point is added for the corner-distance.

A visualization of the four distance types is shown in Figure 4.9. The final score of a candidate consist of the 1 minus the minimum distance of all four types and is calculated using the formula 4.1.

$$score(c) = 1 - min[d_{node}(c), d_{plane}(c), d_{edge}(c), d_{corner}(c)] \tag{4.1}$$

The particular distances are normalized to $\{0..1\}$, depending on the size of the node.

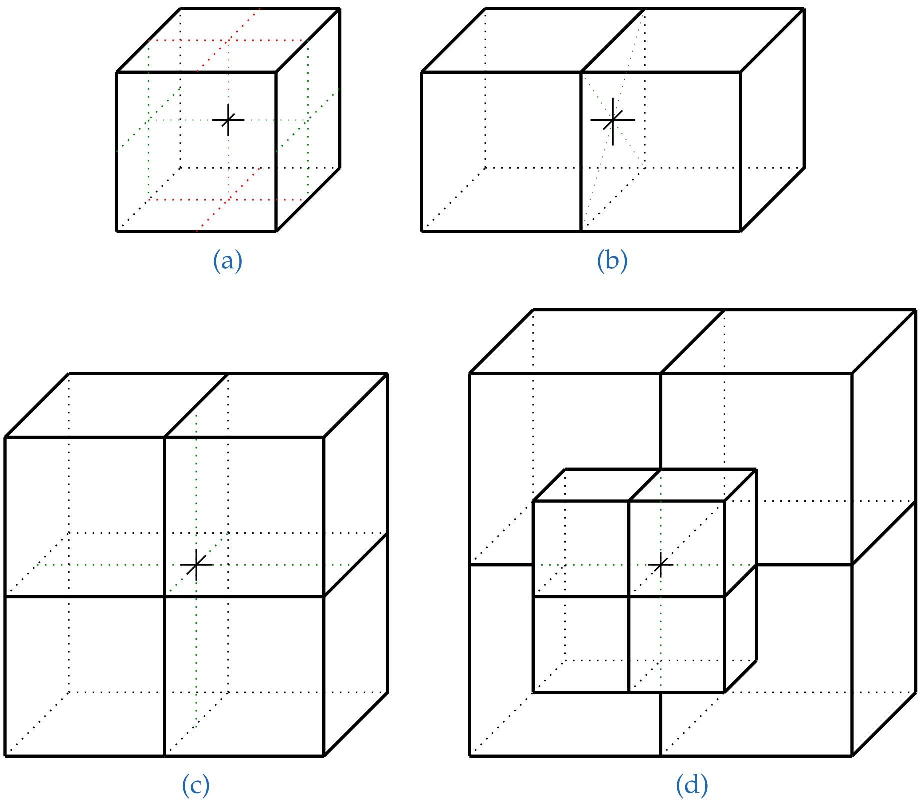


Figure 4.9.: The four distance types: (a) node-center-distance, (b) plane-distance, (c) edge-distance (d) corner-distance. All points lie at the inside of the task. A task with eight nodes has 25 distance points, which is the maximum.

Eventually, this gives us a list of patch-candidates, ranked and sorted by their score. We process them in descending order to minimize the chance of a false patch-candidate insertion.

4.3.3. Patch-Candidate Insertion

A patch-candidate is a set of edge-connected triangles and it can intersect with other patch-candidates. Due to our extraction process it is guaranteed that a patch-candidate does not intersect with the initial surface generated by Step2[4.2]. The insertion is separated in two parts, a *strong* and a *weak* insertion. These insertions are executed consecutively, when the *strong* insertion is finished the *weak* insertion is started. The insertion is always performed for a single node.

If a Patch-Candidate contains triangles from different nodes, these nodes are loaded on demand.

Step 3a - Strong Patch-Candidate Insertion

We define a *strong* patch-candidate insertion as an insertion where the whole patch-candidate is added to the initial surface. That also means that this patch-candidate fully closes a hole in the initial surface. To apply a patch-candidate to the surface we execute the following steps: At first we extract the boundary edges of the patch-candidate, which are all edges that are not 2-manifold. After that we check if the boundary edges connect to the surface properly. They do if all boundary edges of the patch-candidate are present in the surface and they are not 2-manifold too. When inserted, this ensures that the boundary edges become 2-manifold and a hole is completely patched.

Step 3b - Weak Patch-Candidate Insertion

The difference between a *weak* and a *strong* patch-candidate insertion is that in the former a subset of triangles is allowed to be inserted. In the latter all triangles of the patch-candidate have to be inserted. Thus *weak* patch-candidate insertion is at a higher degree of approximation of the original surface with the target to decrease the overall boundary length of the surface. We use the same set of patch-candidates as in the *strong* insertion but we skip all candidates that have already been inserted by the the *strong* version. Before we test if a *weak* patch-candidate fits we have to make sure it does not intersect with the already generated surface. We do this by applying the same checks as in the patch-candidate extraction, see 4.3.1. If the candidate does intersect, we remove the intersecting triangles and re-cluster it. The re-clustering splits the intersecting patch-candidate into a new subset of candidates that can be tested for a *weak* insertion. After we ensured that a patch-candidate does not intersect with the already generated surface we check if it is edge-connected to at least one boundary edge of the surface. If not, the patch-candidate will be skipped and we will continue with the next one. If it is connected to a boundary edge we apply an energy minimization that gives us a (sub)set of the patch-candidate that decreases the boundary length optimally.

The minimization works as followed: We define it as a binary labeling problem. A patch-candidate should be separated into two disjoint sets of triangles. One set will be added to the current surface and the other one will be discarded. We solve this by using a graph cut that has a two layered flow network as input, where a triangle is represented as a node.

In the first layer we put all boundary triangles of the current surface that form an edge-connection to the patch-candidate. We connect these triangles to the source using infinite weights, thus these triangles are always part of the minimization result (as they are already part of the current surface).

The triangles of the patch-candidate are put into the second layer. If a triangle in this layer is (spatially) edge connected to a triangle of the first layer we add a connection in the flow network between them. This connection is always directed from the first to the second layer. The weight of the connection is equal to the length of the shared spatial edge.

If two triangles of the second layer share an edge, we add two connections with different directions between them. Again, the weight is equal to the length of the shared spatial edge.

At last we have to connect the triangles to the sink. Triangles of the first layer will never be connected to the sink, we only consider triangles of the second layer to be connected to the sink. If a triangle of the second layer has an edge that is not shared with any triangle of the first or second layer, we connect it to the sink. The weight of the sink connection is the accumulated sum of edges lengths, that are not shared, for the triangle.

Applying the minimum cut (3.2) gives us now a set of triangles that minimizes the overall border edge. Also, if a weak-patch-candidate perfectly close a hole in the surface, no connection to the sink would be present. In this case all triangles of the weak-patch-candidate would be added to the surface. A sandbox example of *weak*-patching can be found in Section 4.3.3.

Weak-Patching Example

In this section we showcase the process of *weak*-patching on the basis of a small initial surface and a *weak*-patch candidate. In Figure 4.10 we plot the initial surface, a patch candidate and the final weak-patched surface. In this example

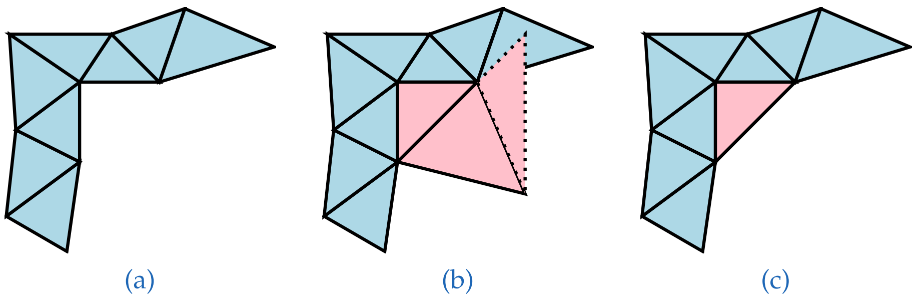


Figure 4.10.: (a) shows surface that is to be *weak*-patched. In (b) a *weak*-patch candidate is drawn with light-red triangles. All triangles, except the dashed one lie on a plane. The dashed one has a different orientation, thus it does not intersect with the initial surface. It only blocks the vision. (c) shows the *weak*-patched surface.

the *weak*-patch candidate is already reduced and does not intersect with the initial surface. The corresponding flow network and its minimum cut can be seen in Figure 4.11.

For a better visualization how the *weak*-patching impacts on a real dataset, we plotted an example *weak*-patch insertion in Figure 4.12. This example is a subset of the Citywall [15] dataset and shows the un-patched surface, a *weak*-patch candidate, the optimized candidate, and the patched surface.

4.4. Limitations

The former sections describe how we add patch-candidates to our initial surface. This section explains the limitations of our processing pipeline. If a hole expands through more than one task, there is a high probability that our pipeline will never fully close it. Our pipeline processes the mesh generation task-wise, thus we cannot deal with holes that spread through more than one task appropriately. The *weak*-patch candidate insertion will most likely reduce the size of the hole but it will not fully patch it. There are two situations where a hole that crosses multiple tasks can be present: A too big measurement density difference and a missing leaf node in the octree.

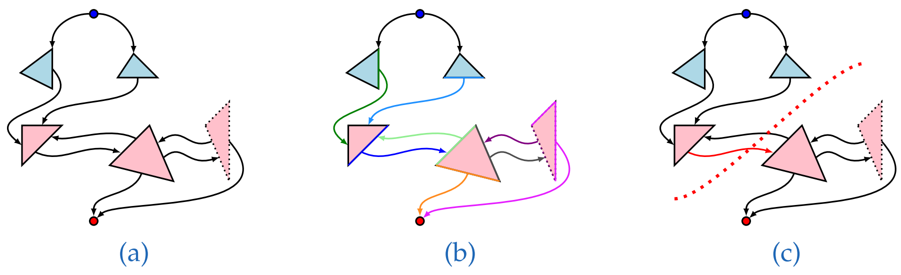


Figure 4.11.: (a) shows the initial flow network. The network consists of triangles of the surface, that are edge-connected to the patch-candidate, and all triangles of the candidate. Triangles of the first layer have the only connection to the source. Triangles of the second layer that are not fully edge-connected have the only connection to the sink. (a) visualizes the edge-weights of this flow network: Edge weights are equal to the length to the shared edge between two connected triangles. Except for the source and sink connections: Source connections are always infinite, sink connections are equal to the sum of triangle-edge lengths that are not connected. (a) shows the minimum cut (red-edge) of this example: Triangles of the patch-candidate that are still connected to the source will be added to the surface.

4.4.1. Measurement Density Difference

When two adjacent nodes have a big measurement density difference it may causes holes that cannot be patched: This is due to the property that a Delaunay triangulation of sparsely distributed measurements has special tetrahedra. In such a tetrahedron one facet is much bigger (area-wise) than the other three it consists of. Which also means that its circumsphere lies outside the tetrahedron and it can be enormously big. Thus the tetrahedron will not be marked as definite, see Section 4.2, and its facets will never be considered for the initial surface. In a worst case scenario such a node in a task forms an island of triangles. This island is not and never will be connected to the rest of the surface. This often happens when a small node only contains a few and sparsely distributed measurements. We evaluated which measurement density difference our pipeline can handle, see Section 5.5.

4.4.2. Missing Leaf Nodes

This section describes the absence of leaf nodes in the octree. As general input we can have point clouds from from various different sensors. We have no

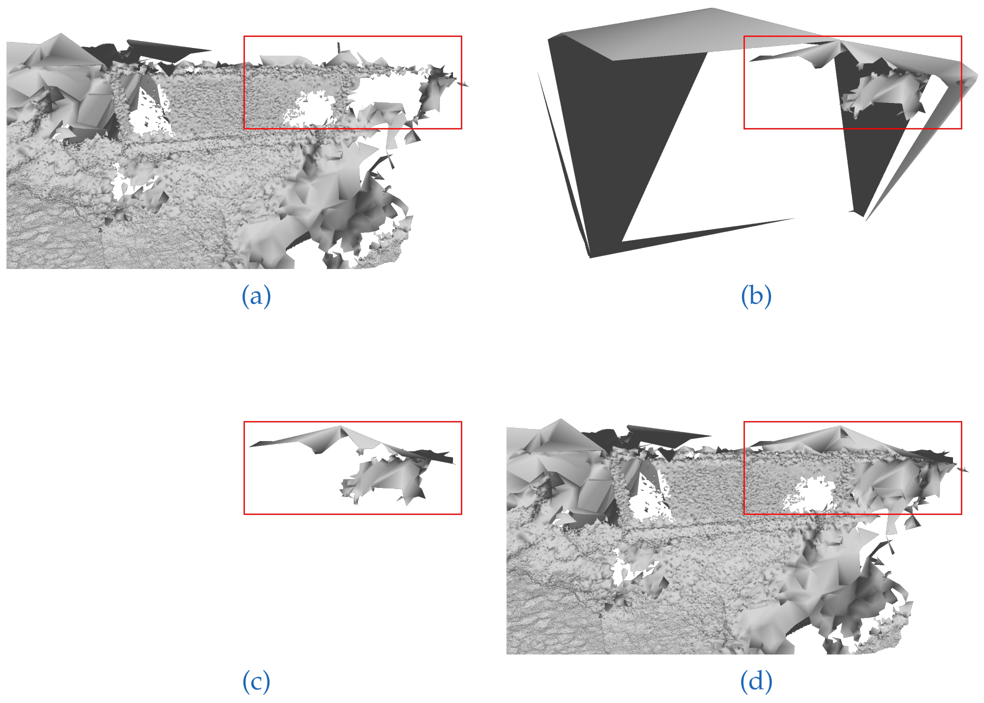
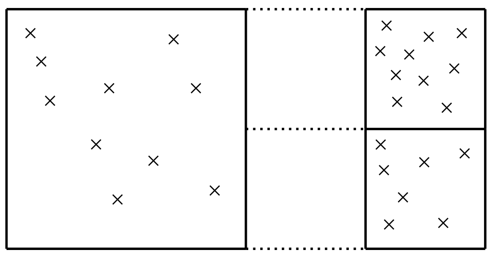


Figure 4.12.: In (a) we see the original surface. In (b) we plotted one patch-candidate. After the optimization the patch-candidate is reduced to the subset plotted in (c). (d) shows the patched surface.

constraint how densely or sparsely distributed their measurements have to be, they only need to be aligned. If we chunk a dense and a sparse point cloud using an octree, the dense one will generate small leaf nodes and the sparse one will tend to generate big ones. This might result in a gap in the octree where the data transitions from dense to sparse measurements. A visualization of such a situation can be seen in Figure 4.13. Since we do not create tasks that expand through such a holes we cannot close them.

4.5. Implementation Detail - Parallelism

Since our data structures are stored in an octree and all steps are performed within a node or a task, our approach has a high potential for parallelism. The base approach, which runs task-wise, has the most potential: It can be



(a)

Figure 4.13.: Visualization of missing leafs in an Quadtree for the 2D case. In this example a leaf node can hold 10 measurements. We have two clusters of measurements, a sparse one at the left and a denser one at the right side. These nodes cannot be merged to a bigger node because it would violate the size constraint

fully parallelized and it is only limited by the computation capabilities and the amount of accessible main memory. For all following steps, a synchronization mechanism had to be taken into account to avoid race conditions. Thus the data structure that contains the final surface for a node is always protected by a locking mechanism. We follow a strict rule: Only one process at a time is to be allowed to perform read and write operations on a node!

Our software is designed in a master/worker hierarchy: For each step we have a master process that overviews the actual progress. The master process is responsible for forking and applying tasks to worker processes. If a worker finishes a task it notifies the master, the master then forks a new worker until all tasks are finished. The number of active workers is configurable and can be set at the start-up of the master process.

Due to our strict Read & Write rule, sometimes workers have to wait for a node resource that was acquired by an other worker. To minimize this *sleeping* time the workers are forked in random order.

5. Experiments

In this chapter we discuss the results of our experiments. We split the the evaluation in a qualitative and quantitative part and compare our results with two state of the art volumetric meshing approaches GDMR [41] and FSSR [13]. As mentioned in Mostegel [35] technical report, the base approach does not handle Gaussian Noise well. Thus we performed two iterations of HC-Laplacian smoothing [42] as post-processing on our generated meshes.

The quantitative evaluation was performed on the DTU [22] and the Middleburry *Temple Full* [39] datasets. Both are single scale datasets that provide ground truth which allow us to benchmark our approach in terms of *accuracy* and *completeness*.

Since there are not any multi scale 3D reconstructions benchmarks available yet, we solely rely on a visual comparison. We perform this qualitative evaluation on two datasets: The first evaluation was done on a public available multi-view dataset [15], we refer to it as the *Citywall* dataset. Beside the visual evaluation we also evaluated the memory and runtime consumption with different parameter sets for our approach on this dataset. The second dataset, called *Valley* dataset, is a cultural heritage dataset and covers an area of 6km^2. Where the images were taken at 4 different scale levels with a ground sampling distance range from 50 µm to 100 cm.

Last, but not least we challenge our approach with a self-created, synthetic dataset to find out under which circumstances our generated surface lacks completeness.

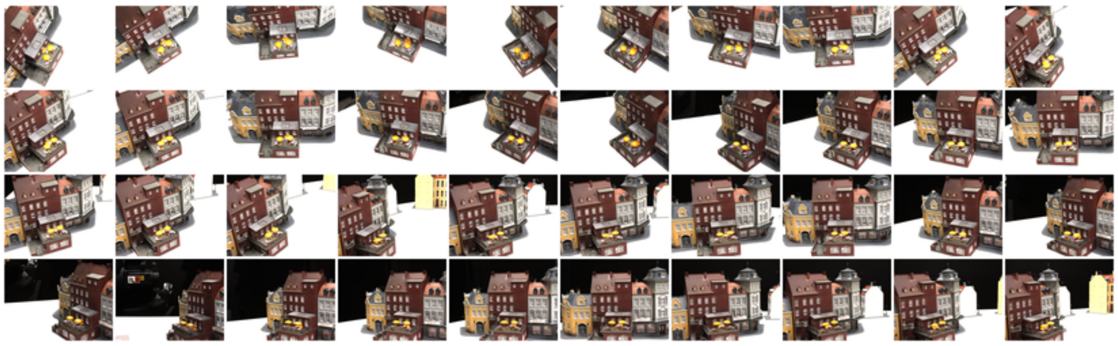


Figure 5.1.: DTU Scene 25 views (subset). Images were taken at a resolution of 1600x1200.

5.1. MVS Dataset 2014 - DTU

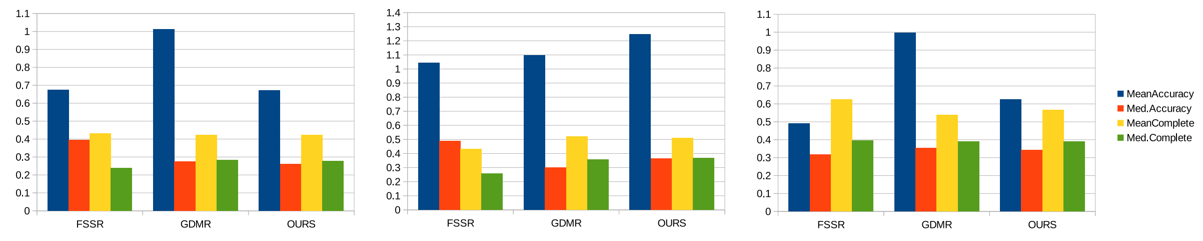
This dataset consists of 124 different miniature scenes and is freely available [22]. It focuses on the multiple view stereo (MVS) evaluation and it contains ground truth data for comparison. The ground truth was acquired by using an industrial robot arm mounted with a structured light scanner. For each structured light scan they have one corresponding image in the dataset, which was taken by one camera in the scanner.

They also delivered a comparison framework for reconstructed 3D points (MVS results) and meshes. The comparison is closest-point-distance based, thus they convert a mesh back to a 3D point cloud. This is done by super-sampling the mesh and a subsequent reduction of high point densities. The reduction is necessarily to meet their requirements of an equal and fair comparison. The evaluation is based on *completeness* and *accuracy*. Accuracy is measured as the distance from the sampled 3D points to the structured light reference, and the completeness is measured from the reference to the sampled 3D points [22].

The main focus of the evaluation is to rate the results of different MVS approaches. Since the meshing quality highly depends on these results we did not run our approach on all scenes. Instead we only choose one scene and run three different state of the art MVS approaches (PMVS [16], MVE [15] and SURE [38]) on it. Their results serve as input for our surface reconstruction approach. In addition, we compare our results with two recently published volumetric meshing approaches, GDMR [41] and FSSR [13].

For this experiment we picked scene 25 which has 49 images, views from different angles can be seen in Figure 5.1. It contains a miniature model of an old building with some challenging meshing properties: A terrace on the first floor which is fenced by some railings. Furniture, like tables, chairs and benches, umbrellas, an entrance with arc and a detached sign-plate. To make the evaluation more accurate we removed obvious outliers from the structured light reference point cloud.

For our approach we used a leaf-size of 128k. For GDMR and FSSR we used different parameter sets to find their best evaluation outcome. For FSSR we changed the scale multiplication factor and for GDMR λ_1 and λ_2 in multiples of two. The detailed evaluation outcomes for accuracy and completeness are plotted in Table 5.1. There, the highlighted rows mark the best evaluation outcome (where the mean of the median-accuracy and the median-completeness is at a minimum). These tables are only supplemental and are only plotted to proof that we took the effort to find the best evaluation scores for the competing approaches.



(a) MVE (b) SURE (c) PMVS

Figure 5.2.: Accuracy and completeness plots on scene 25 of the DTU dataset [22]. We evaluated three surface reconstruction approaches: GDMR and FSSR and OURS. We used three different MVS approaches (MVE [15], SURE [38] and PMVS [16]) for the input point cloud generation. The plots show the best mean and median values for the accuracy and completeness, lower values are better.

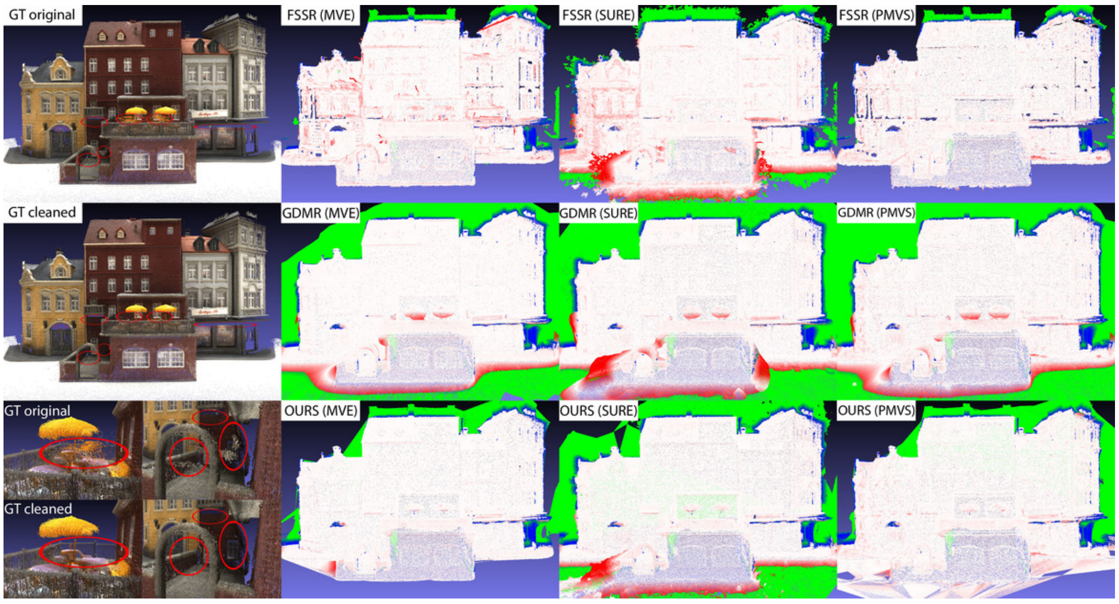
For a better comparison, the best outcomes for this evaluation are also shown in Figure 5.2 and a visual-only evaluation of the accuracy of all three approaches is shown in Figure 5.3.

In Figure 5.3a we visualized the accuracy evaluation provided by Jensen et al. [22]. In the first column we show the original Ground-Truth (GT), and the modified GT where we removed obvious outliers. We put the labels for every sub-image in its top-left corner: FSSR, GDMR and OURS. In brackets we note the MVS algorithm we used to obtained the input point cloud: MVE [15], SURE [38] and PMVS [16]. Points from blue to green were excluded from the evaluation, points from white (no error) to red (error >10mm) were evaluated.

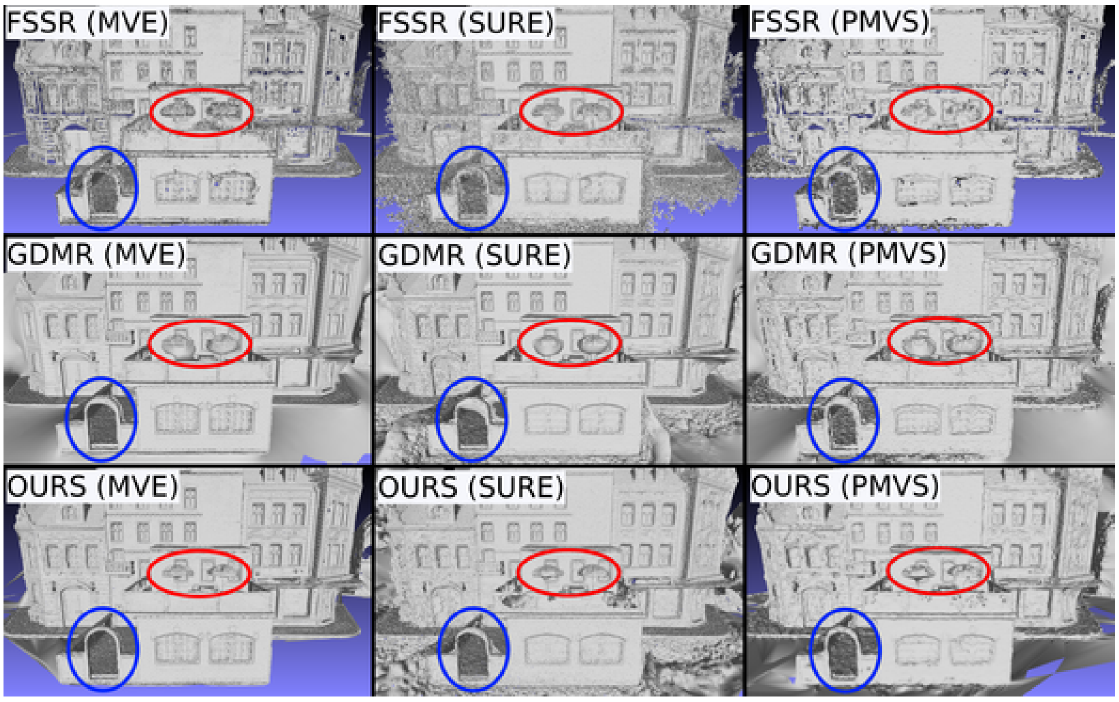
In Figure 5.3b we plotted the non-textured surface meshes for all approaches. The nomenclature for the sub-images is the same as in Figure 5.3a.

5.1.1. Discussion

If we take a closer look at Figure 5.2 we can see that GDMR and our approach are more robust to outliers than FSSR. We can confirm this by the median-accuracy with the MVE and SURE input. On a dataset that does only have a few outliers (PMVS), FSSR has the highest accuracy. This high accuracy comes with a trade-off, as we can see in Figure 5.3b in the first row, FSSR generally leaves a lot of holes in the reconstruction. If we (visually) compare GDMR to our approach we can conclude that both perform very similar on the facade. Yet, if we focus on the arc and the umbrellas (highlighted in Figure 5.3b) we notice that GDMR tends to form unwanted bubbles. This is caused by the formulation of GDMR which prefers smooth normal transitions if points that constrain the algorithm are absent. In this case our approach generally closes holes, where GDMR focuses on a smooth transition. For this example our approach leads to a better accuracy in these regions, see Figure 5.3a. From this experiment we can conclude that the quality of a surface reconstruction method strongly depends on the quality of the input and that on this dataset our method is on par with the others.



(a)



(b)

Figure 5.3.: See text for description.

MVS	Approach	Scale Factor	Accuracy			Completeness		
			Mean	Median	Variance	Mean	Median	Variance
PMVS	FSSR	2	0.332	0.245	0.079	1.110	0.476	4.867
		4	**0.491**	**0.318**	**0.273**	**0.624**	**0.395**	**1.772**
		8	0.593	0.327	0.551	0.764	0.413	3.126
	GDMR	1	1.651	0.456	9.159	1.280	0.621	3.785
		2	1.372	0.373	6.855	0.789	0.468	1.510
		4	1.149	0.343	4.991	0.581	0.404	0.735
		8	**0.996**	**0.355**	**3.024**	**0.537**	**0.389**	**0.613**
		16	0.988	0.377	2.697	0.529	0.381	0.615
		32	1.003	0.402	2.586	0.518	0.368	0.633
	OURS	**-**	**0.626**	**0.341**	**0.755**	**0.567**	**0.390**	**0.743**
SURE	FSSR	**1**	**1.044**	**0.490**	**2.487**	**0.431**	**0.257**	**1.218**
		2	1.594	0.523	5.935	0.501	0.353	0.985
		4	1.975	0.496	9.503	0.485	0.370	0.450
		8	2.395	0.520	14.259	0.520	0.387	0.462
	GDMR	0.5	1.101	0.295	4.931	0.565	0.373	0.917
		1	**1.099**	**0.301**	**4.693**	**0.519**	**0.357**	**0.744**
		2	1.163	0.322	4.813	0.494	0.339	0.753
		4	1.358	0.373	5.687	0.465	0.317	0.703
	OURS	**-**	**1.247**	**0.365**	**5.013**	**0.509**	**0.368**	**0.512**
MVE	FSSR	**1**	**0.673**	**0.396**	**0.685**	**0.430**	**0.239**	**1.638**
		2	0.833	0.403	1.213	0.474	0.285	1.630
		4	0.878	0.338	1.698	0.490	0.289	1.625
	GDMR	0.5	1.048	0.269	4.846	0.460	0.290	0.716
		1	**1.013**	**0.275**	**4.262**	**0.423**	**0.284**	**0.587**
		2	1.021	0.287	4.015	0.410	0.278	0.496
		4	1.008	0.304	3.539	0.407	0.270	0.524
		8	0.971	0.332	2.772	0.399	0.260	0.564
	OURS	**-**	**0.671**	**0.262**	**1.330**	**0.423**	**0.279**	**0.575**

Table 5.1.: Detailed DTU Evaluation Results - with PMVS, SURE and MVE as MVS.

5.2. Middlebury Dataset

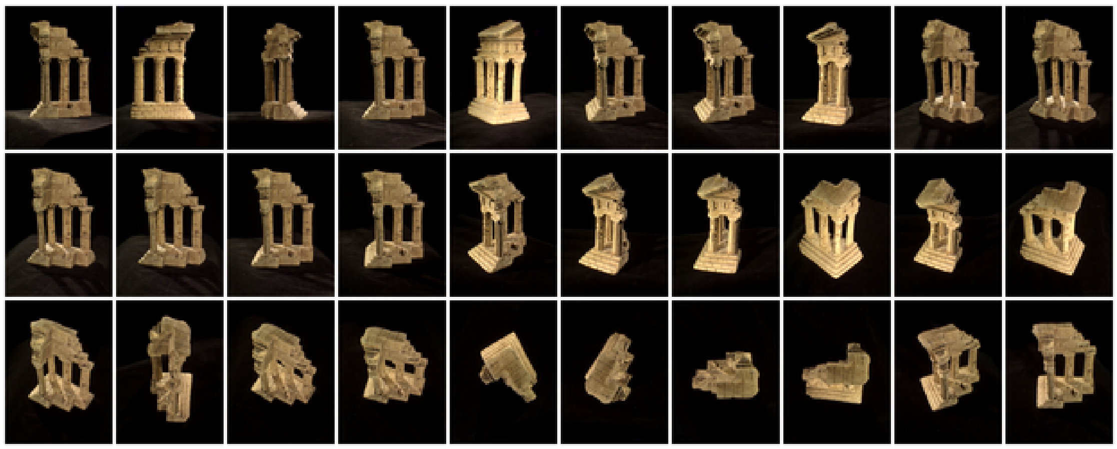


Figure 5.4.: A subset of images of the Middlebury *Temple Full* dataset.

As GDMR [41] and FSSR [13] evaluated their approaches on the Middlebury *Temple Full* [39] dataset, we also challenged our approach by proposing our result mesh to this non public evaluation system. An image subset of this dataset can be seen in Figure 5.4. The object to be reconstructed is a model of the *Temple of the Dioskouroi* made of mortar. The benchmark contains 312 views that are sampled on a hemisphere with an image resolution of 640 times 480 pixels. The ground truth is not public and has a resolution of 250 µm. We computed a point cloud using MVE [15] and submitted our reconstructed mesh. In Figure 5.5 we plotted the two main views of the reconstructed meshes and their corresponding ground truths for OURS, FSSR and GDMR.

5.2.1. Discussion

If we take a look at the accuracy and completeness scores printed in Table 5.2, we see that our approach is higher ranked than FSSR and GDMR. A possible explanation for this result can be seen in Figure 5.6. There we cropped out a magnified detail that proves that our approach preserved fine details and edges more accurate than GDMR and FSSR. We assume that this property led to the higher accuracy score. If we compare our result score to all submitted and evaluated MVS approaches we are ranked second on this dataset[1] over all

[1] `http://vision.middlebury.edu/mview/data/` (July 15, 2018)

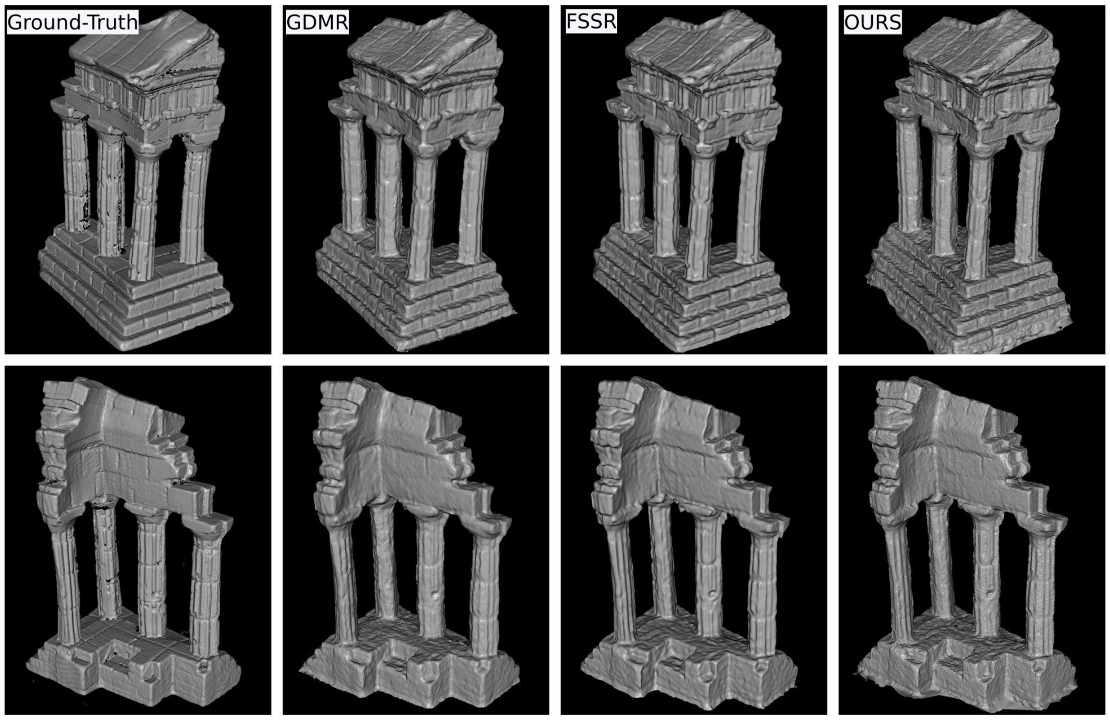


Figure 5.5.: Middleburry Full Temple Dataset. First row shows *"view 1"* and the second shows *"view 2"* from the evaluation system.

approaches. Note that the only approach better than ours [44] uses a different input point cloud. While the other approaches reported here use the exact same input as we do. This means on this input, our approach leads to state-of-the-art results on this dataset.

Threshold	Accuracy[mm]			Completeness		
	FSSR	GDMR	OURS	FSSR	GDMR	OURS
90 %	0.40	0.42	0.35	99.4 %	99.3 %	99.7 %
97 %	0.63	0.61	0.55			
99 %	0.84	0.78	0.71			

Table 5.2.: Accuracy and Completeness report for the Middleburry Dataset.

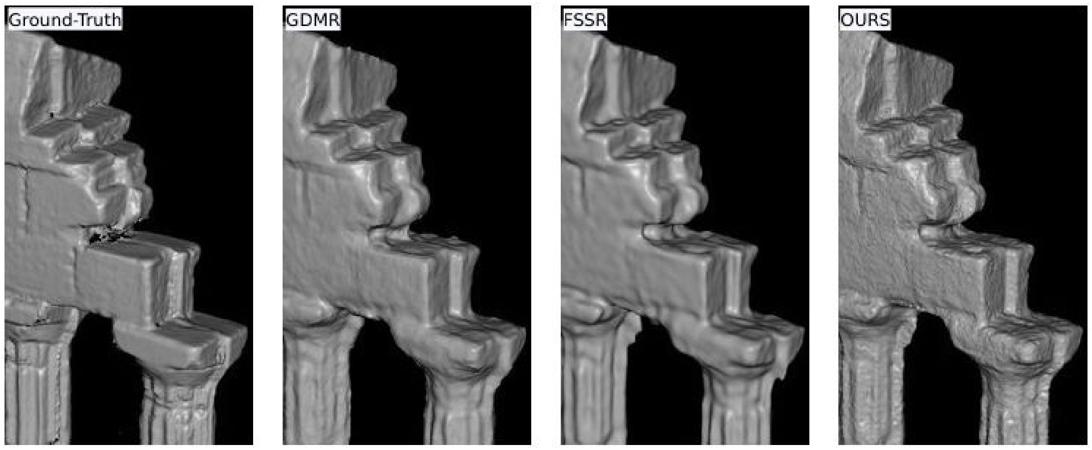


Figure 5.6.: Middleburry Full Temple Dataset. Cropped out detail of *"view 2"* which shows that our approach preserved more details an edges. This property probably led to the higher accuracy score.

5.3. Citywall Dataset

Figure 5.7.: A small subset of the Citywall dataset images.

The Citywall dataset is a freely available multi-view dataset [15]. It consists of 564 images that were captured hand held. The scene covers an old historic wall with a fountain and a miniature model of a village, a sub-sample is shown in Figure 5.7 The images were captured from different distances: While most of the images cover the scene from a certain distance they also captured low-distance images to preserve small details. Due to this high scale difference, this dataset is highly suitable for testing our reconstruction pipeline. Unfortunately there is no ground truth available, thus evaluation can only be done visually. The number of measurements a leaf node can hold is the only parameter that has a major impact on the runtime, quality and completeness of our reconstructed surface. We refer to this number as leaf-size. Our experiment consists of eight runs with different leaf-sizes: from 8K up to 1M points.

At first we use the incremental structure from motion tool from MVE (Multi-View Environment)[15] to generate a sparse geometry from the scene. To densify this geometry we use MVS (Multi-View Stereo) [17] at scale level L_1 which gives us a depth-map for every camera at a half image resolution. The total number of (dense) measurements is 295M points and every measurement is seen by only one camera. This dense geometry and the low visibility information serves as input for our surface reconstruction pipeline. For Step1 we used a $k = 30$ for the k-nearest-neighbor fusion which resulted in a fused point cloud of 22.5 million points.

We also visually compare our result mesh with GDMR [41] and FSSR [13]. We chose a leaf-size of 128k for our, and default parameters for the other approaches.

5.3.1. Runtime & Memory Consumption

For this experiment we logged the runtime and peak memory consumption for Step2, Step3a & Step3b. The runtime of Step1 is negligible compared to the meshing itself, it takes less than 1% of the total runtime. The experiments were performed on a server with 210GB of RAM and 2 Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz. Which sums up in 20 physical cores that can process 40 threads at once. They server was shared with other users, thus the run times can only be interpreted as an indication. In addition, we did not use the same number of CPU cores for every step and leaf-size. Therefore we normalized the runtime to a setup where only one process can be processed at once. These timings and the PMCP (Peak-Memory-Consumption per Process) can be found in Table 5.3. With a leaf-size of 8K our approach finishes this dataset on a 4-core CPU (with hyperthreading) and 8GB of RAM within 2 days.

	1M	512K	256K	128K	64K	32K	16K	8K
#Nodes	142	194	378	695	1392	2675	4820	10123
#Tasks	172	232	499	938	1805	3604	6317	13254
Runtime/Node	1h12m	42m	15m	7m	3m	1m	40s	20s
Runtime/Task	4h20m	5h	2h20m	1h	25m	12m	4m	1m30s
Total Runtime~	**1260h**	**1296h**	**1258h**	**1019h**	**821h**	**765h**	**473h**	**388h**
PMCP [GB]	44.3	25.3	16.1	8.9	5.1	3.1	2.2	1.8

Table 5.3.: Runtime and Peak Memory Consumption.

5.3.2. Discussion

In Figure 5.8 we show a visual comparison of our result meshes. As we can see in the Fountain view-port (1st column) the bars are reconstructed with some outliers for a leaf-size of 8k and 16k. In these cases there are some leaf nodes in the octree that mainly contain outliers and they form a task. Since the visibility casts (see Section 3) are only performed within a task we cannot distinguish outliers from valid measurements on a global base. Thus, with increasing node size this effect becomes negated: From a node size of 32K to 1M our approach is more outlier resilient on this dataset.

In regions where the point density is constant (Wall, Lions and City view-ports) our approach is capable to preserve even small details and produce a closed surface, even for a leaf-size of 8k. The Stairs view-port shows the impact on the completeness: A large leaf-size leads to a more complete result. With a large leaf-size we have less tasks but a task contains more measurements and covers a larger area, thus the base optimization has more information.

For the visual comparison of this dataset with GDMR and FSSR we use similar view points as used in [41]. These visual results are shown in Figure 5.9. The big advantage of GDMR is that they always reconstructs a fully connected mesh without any holes. However, with their standard parameters they tend to overly smooth the scene and they did not propose their used parameter set in [41]. Both, FSSR and GDMR have some artifacts in the basin of the fountain, where our approach does not. In addition, FSSR is often not able to reconstruct parts of the scene which leads to big holes and gaps.

From the former visual comparison and the runtime and quality evaluation we conclude that a leaf-size of 128K is preferable for our approach. With this

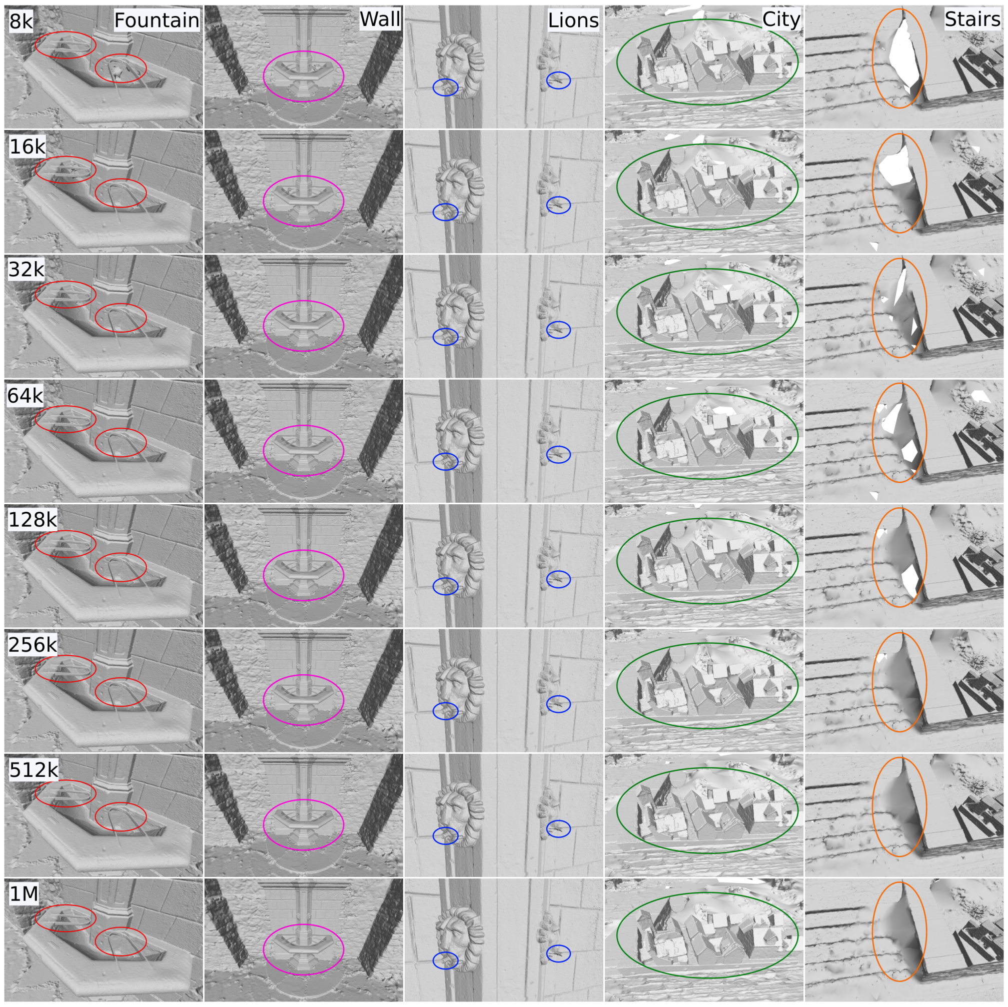


Figure 5.8.: Citywall - Surface reconstruction results for different leaf-sizes. Prominent features are highlighted.

leaf-size we are visually definitely on par with GDMR and FSSR on this dataset while the runtime of our approach is also in an acceptable range.

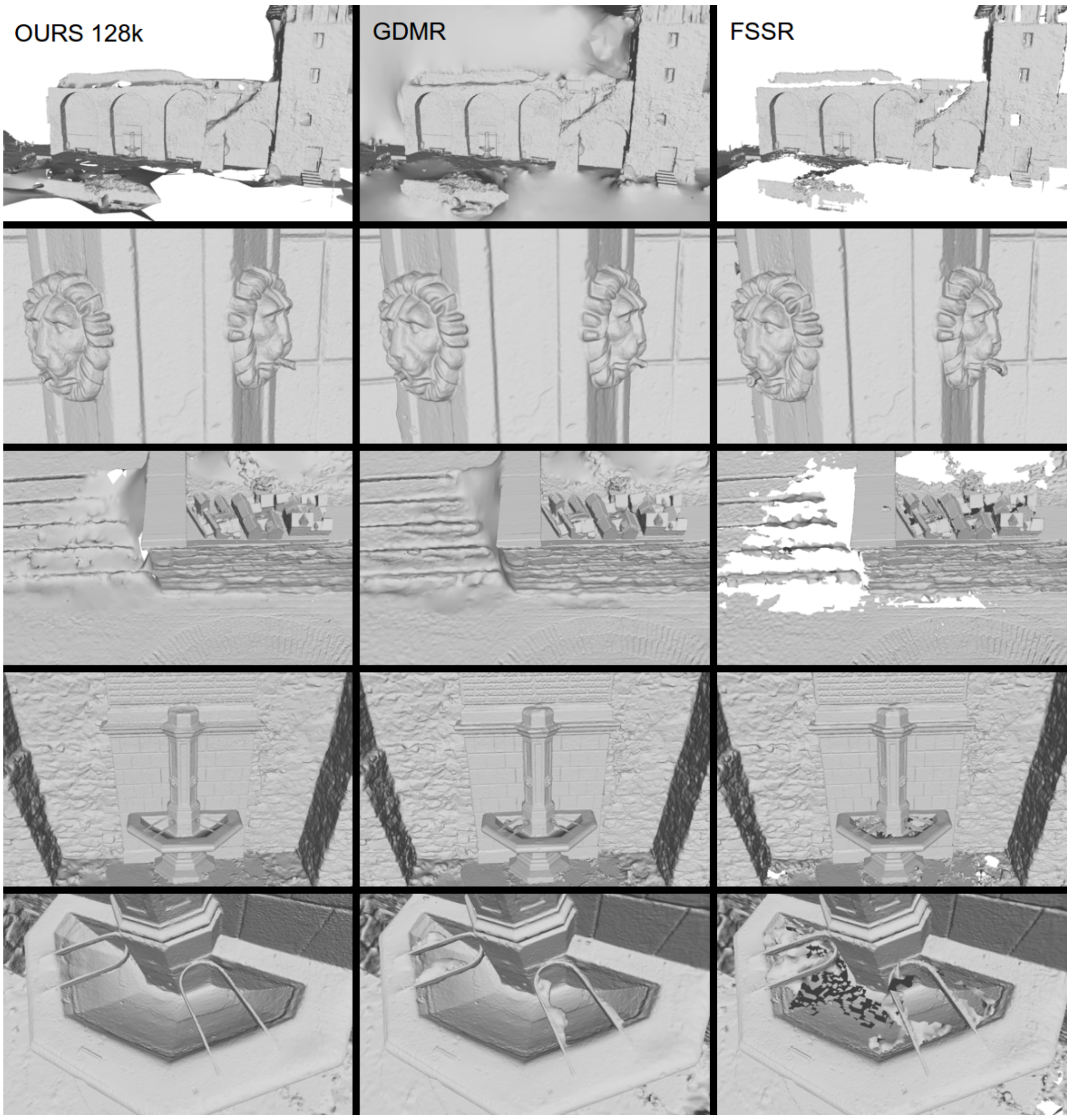


Figure 5.9.: First column shows the result mesh of our approach with a leaf size of 128k. The second & third column show the result meshes for GDMR ($\lambda_1 = \lambda_2 = 40$) and FSSR (default parameters).

5.4. Valley Dataset

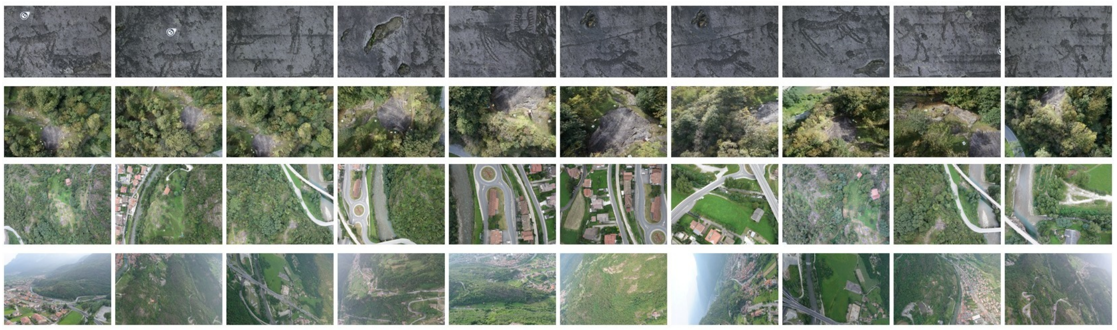


Figure 5.10.: A small image subset of the Valley dataset. The images are ordered row-wise from scale level 1 to 4.

This experiment was chosen to show the surface reconstruction power of our approach. The dataset contains an extreme scale variance, it has four scale levels with a ground sampling distance from (about) 1 m to 50 µm. It also covers an area of about 6 km² and zooms in to highly dense sampled area of $0.1\,\mathrm{m}^2$. The initial data of each scale level was acquired vision based and aligned with geo-referenced ground control points. We also processed this dataset with GDMR [41] and FSSR [13] and (visually) compare their results with ours.

5.4.1. Data Acquisition

The raw data was acquired in the Valcamonica Valley, Italy during the proceeding of the 3D-pitoti [2] EU-project. A small image subset can be seen in Figure 5.10. Each scale level was reconstructed individually which resulted in four different sparse point sets and their visibility information. The most coarse scale was captured with a manned hang glider. The second and third were both recorded with an unmanned aerial vehicle, a fixed wing and octocopter respectively. The finest scale was captured with a terrestrial stereo setup. After reconstruction, the datasets were geo-referenced using a prism on the total station and ground control points of offline differential GPS, and total station, measurements. We further optimized the relative alignment by using the iterative closest point algorithm [7]. After that we densified each scale level

[2]http://www.3d-pitoti.eu/

point cloud using SURE [38]. The sizes and ground sampling distances of the four densified datasets can be found in Table 5.4a. This sums up in a combined dataset that consists of 1925 million points with a ground sampling distance range from 42 µm to 100 cm.

Scale Level	#points	min. GSD	max. GSD
1	572M	42 µm	47 µm
2	162M	3.5 mm	15 mm
3	64M	3 cm	5 cm
4	1127M	10 cm	100 cm

(a)

Approach	Runtime[Days]	PMC[GB]
GDMR	1.5	150
FSSR	0.5	170
OURS	9	119

(b)

Table 5.4.: (a) shows the size and ground sampling distances of the four scale levels. (b) shows the runtimes and memory consumptions.

5.4.2. Execution Setup

For a fair comparison we used the parameters for GDMR and FSSR where they performed best (in terms of accuracy and completeness) on SURE densified input data. We obtained these parameters from our DTU experiment where we used SURE as MVS, see Table 5.1. Both approaches ran out of memory on our evaluation server (further specification in Section 5.3.1) with 210 GB of accessible RAM. However, to achieve any result we increased their scale parameter in multiples of two until both could be successfully executed. This resulted in a scaling factor of 4 for both approaches. Another problem was the scale difference of our dataset: The reference implementations of GDMR and FSSR use a fixed octree depth of 21 levels, but our dataset requires a higher depth. Thus both approaches ignored the finest scale level. Therefore we also executed both approaches on a subset of our dataset that contained only scale 3 and 4. To evaluate the transition capabilities of GDMR and FSSR on the two lowest scale levels we executed both approaches also only on these scale levels. FSSR and GDMR ran on a reduced dataset without scale level 1 and 2 which represents a data reduction of more than a half.

5.4.3. Discussion

In Figure 5.12 we plotted the reconstructed meshes. Our approach consistently connects all scales, where GDMR and FSSR have their difficulties with the lowest scale. This is due to the fact that our approach uses the Divide & Conquer principle and can thus process data from all scale levels. As long as the density transition between to adjacent nodes is smooth enough, our approach will try to reconstruct a surface that interfere within these nodes. If we take a closer look at the smallest scale in Figure 5.12 we can see individual small holes at density transitions. However, if we focus on the density transition between the two lowest scales (see Figure 5.11), we see that our approach is able to close nearly the whole transition despite the immense jump in the point density,

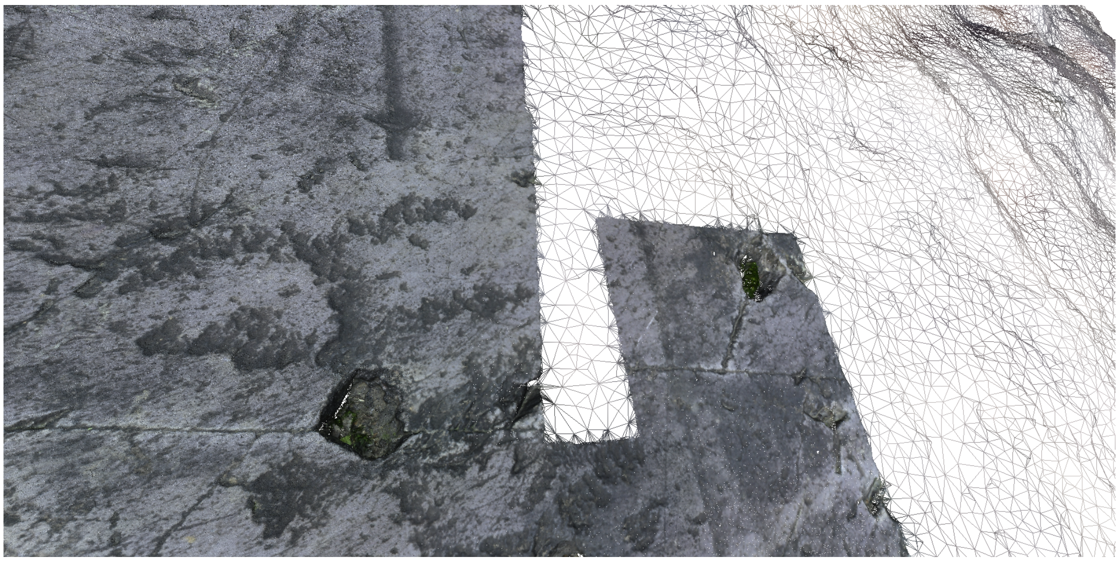


Figure 5.11.: Density transition between the two lowest scales of the valley dataset.

The overall runtimes and PMC (Peak Memory Consumption) can be found in Table 5.4b. Our approach used 16 processes and the maximum memory consumption was less than 9 GB per process. The main disadvantage of our approach is the runtime as shown in Table 5.4b. With further improvement of parallelism we are sure to be able to drop the runtime significantly.

However, it is important to note that our approach was the only one that could process the whole, unfiltered dataset with all four scale levels, at once.

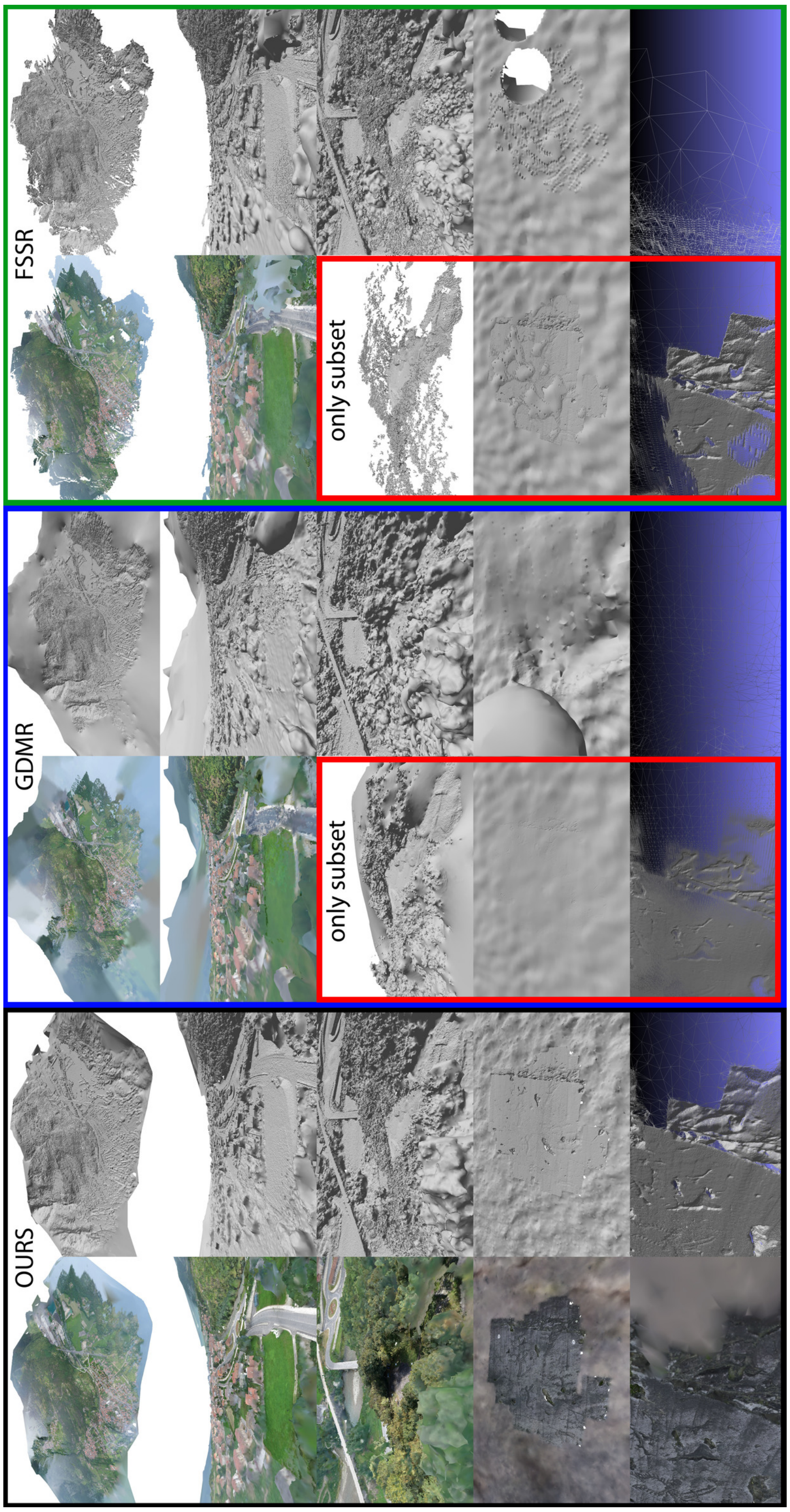


Figure 5.12.: Valley comparison. We pass over the massive scale changes from top to bottom. We show our results (black box), GDMR (blue box) and FSSR (green box). As both, GDMR and FSSR were not able to fully process the whole dataset we also plotted their result on a subset that contained only points of scale level 3 and 4 (red boxes).

5.5. Synthetic

We processed lots of real world sensor data where the reconstructed surface contained holes. As described in 4.4.1 we figured out that this mainly happens at point cloud density transitions in the data. Chunking such data results in an octree where adjacent leafs have different densities and often different sizes. We decided to challenge our pipeline by generating worst case datasets with increasing density transitions.

We do this by creating a set of points that lie on a plane. The x and y coordinates of the points are sampled on a regular grid. We add some small normal distributed noise to the z coordinates of the points to ensure that the tetrahedra in the Delaunay triangulation are not flat. We place 4 virtual cameras that are positioned fronto-parallel to the grid plane and project the sampled points to get the visibility information. This initial set of point has 2.4 Million points.

Afterwards we start to reduce the number of points in the center region. The boundary of this region is computed that a reduction of points leads to an octree with bigger inner nodes that are surrounded by smaller and denser outer nodes, see Figure 5.13. The reduction ratio for the center region is computed as shown in Equation 5.1.

$$ratio(n) = \frac{1}{4^n}, n = \{0, 1, 2, 3, 4, 5, 6\} \tag{5.1}$$

Where n is the current iteration. After the last iteration we have 1.4 Million points left.

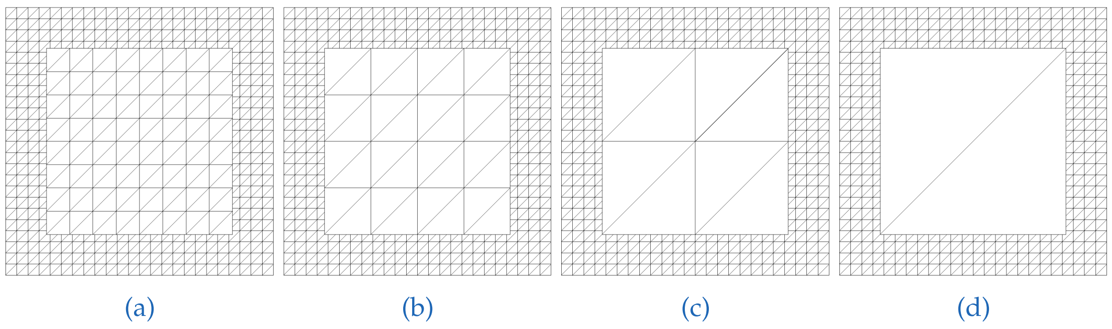


Figure 5.13.: Octree for synthetic experiments, iteration 1,2,3 and 4.

5.5.1. Discussion

For $n < 3$ our pipeline generates a fully closes mesh of the point cloud. For $n >= 3$ the number of un-patched holes increases continuously. We plotted the result meshes for $n = 2 \ldots 6$ in Figure 5.14. As we can see in iteration 6, Step2 mainly produces an island in the center node, this is due to the lack of *definite*-facets. Therefore Step3 is not capable to fully connect this island with the surrounding surface. The hole spans through more than one task, thus there are no Patch-Candidates that can fully connect to the surface. We can clearly see the impact of Step3b, it always reduces the size of holes. Even in iteration 6 the size of the holes is reduced to more than 50% (from Step3a to Step3b). Until a density transition of 1/256 the number of holes in the surface is negligible low.

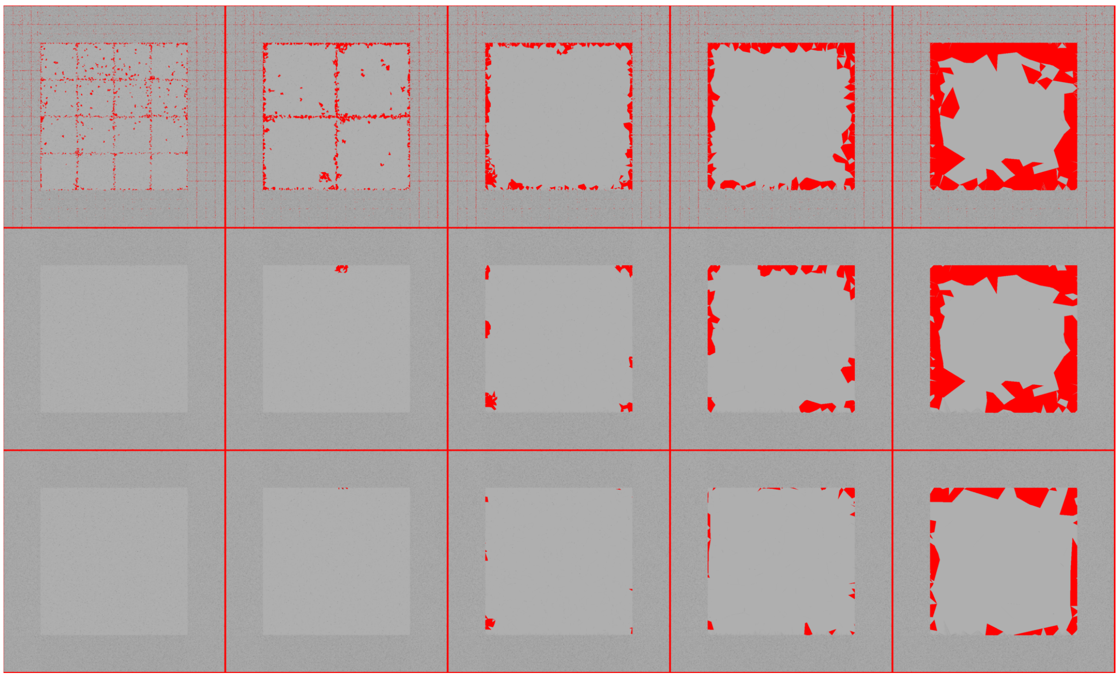


Figure 5.14.: Generated meshes for our synthetic test data set.
Rows: Sub-result meshes of Step2, Step3a and the final result Step3b.
Columns: iterations 2,3,4,5 and 6.
Holes are colored red.

5.6. Discussion - Summary

Overall our approach performed well in all of the presented experiments and proofed to be competitive with state-of-the-art surface reconstruction methods. Nevertheless, during the execution a few topics rose that we discuss and summarize in this section.

One of the reasons for the Citywall experiment (5.3) was to find the most practicable *leaf-size*. There we observed that a higher *leaf-size* tends to improve the completeness and overall quality of the reconstructed mesh at the cost of a higher memory consumption and runtime (to a certain limit). Where the runtime mostly flattens at a higher *leaf-size*, the memory consumption rises intelligibly, see Table 5.3. Thus we see the selection of this parameter as a trade-off between overall mesh quality and computational resource consumption. From this experiment we concluded that a *leaf-size* of 128k suits us best, and was therefore used in all further experiments.

Due to the fact that we allow an unlimited leaf depth in our octree data structure, the question of how density jumps impact the completeness, came up. This line of thought led to the creation of the Synthetic experiment (5.5) where we simulated worst case density jumps. The outcome of this experiment was that our hole filling mechanism works well up to point density jump of 1/256. However, in the valley experiment (5.4), much higher point density jumps occurred, which did not lead to a significant number of holes. In addition, the decision to allow unlimited leaf depths grant our approach to be *genuinely-scalable*.

The main disadvantage of our approach is the overall runtime. In this aspect other approaches are simply more efficient, see for example Table 5.4b. We think that further effort in parallelism and a more intelligent locking mechanism (4.5) could reduce the overall runtime by a lot. In general, consumer CPUs tend to get more and more cores, which naturally improves our runtime.

The capability of running on a consumer desktop without high-end hardware is a major advantage of our approach. In all of our experiments the allocated memory never exceeded 9GB per process. When decreasing the *leaf-size* the memory consumption drops significantly more. Thus, with our approach a surface can be reconstructed on a less potent hardware setup.

6. Conclusion & Outlook

While the other approaches are *implicit* approaches, ours works directly on the point cloud and is thus *explicit*. The formulation as a hybrid between *volumetric* and *Delaunay* based surface reconstruction approaches grants us the ability to reconstruct a surface from a point cloud of any size at a constant memory consumption. One further difference between our approach and GDMR [41] and FSSR [13] is that we use an octree with unlimited depth as data structure. It is only constrained by the number of samples a voxel can hold which guarantees to evenly partition the input samples into chunks that can be processed. In addition, the scale of a sample does not influence the leaf-depth it will be stored in. Our approach does not require any normal or scale information like the others but it highly depends on visibility information.

In our experiments our approach was the only one that were able to reconstruct a high quality and consistent surface from a dataset with roughly 2 billion points (5.4) that contains vast scale changes. Since our process pipeline is based on the Divide and Conquer principle, we were well suited for processing such large amount of data. In terms of reconstruction quality our experiments proof that we are on par with state-of-the art multi-scale surface reconstruction methods. Our memory consumption directly depends on the only parameter our approach needs, the *leaf-size*. The overall runtime is a minor drawback that can be counteracted by decreasing the *leaf-size*.

To further increase the quality of the reconstructed surface we could replace our base approach [30] with another one. With some minor adaption we would be able to make some of Delaunay based surface reconstruction methods scalable! To reduce the overall runtime we could beforehand calculate the best task execution order. A minimization of the time a process needs to wait before it can access its required resources would be, in our opinion, a good starting point.

Since the computation capabilities in terms of multi-processing is still increasing continuously, even on the consumer market, we see a potential for methods like ours that use a Divide and Conquer principle. When we think of cloud computing with the sheer amount of available processing units can drastically speed up our reconstruction runtime, which would make it more practical.

However, the race between *explicit* and *implicit* methods is still going on. With both having their set of advantages and disadvantages, someone cannot predict which variant will overcome the other. In our opinion the property of being scalable will become more and more important in the future for a modern surface reconstruction method, independently of its category.

A. Appendix - CVPR Submission

The method we presented in this thesis was also submitted as a paper to CVPR, 2017 and got accepted [36]. The apportionment of the work was the following: The ideas that are used in our approach were formed and exhaustively discussed between Christian Mostegel and me. The C++ software implementations needed for the paper were mainly done by myself. The experiments were planned and executed by Christian Mostegel and me. Friedrich Fraundorfer and Horst Bischof supported us with their knowledge and experience throughout the whole project and were always available for a detailed discussion and critics.

At this point I want to thank all my project members and former colleagues for their patience and great support. Special thanks to Christian Mostegel; he was always sympathetic and supportive and he also gently pushed me to finish my master thesis.

Bibliography

[1] Nina Amenta and Marshall Bern. "Surface Reconstruction by Voronoi Filtering." In: *Proceedings of the Fourteenth Annual Symposium on Computational Geometry*. SCG '98. Minneapolis, Minnesota, USA, 1998, pp. 39–48. ISBN: 0-89791-973-4. DOI: 10.1145/276884.276889. URL: http://doi.acm.org/10.1145/276884.276889 (cit. on pp. 5, 13).

[2] Nina Amenta, Marshall Bern, and David Eppstein. "The crust and the β-skeleton: Combinatorial curve reconstruction." In: *Graphical models and image processing* 60.2 (1998), pp. 125–135 (cit. on p. 20).

[3] Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. "The Power Crust." In: *Proceedings of the Sixth ACM Symposium on Solid Modeling and Applications*. SMA '01. Ann Arbor, Michigan, USA: ACM, 2001, pp. 249–266. ISBN: 1-58113-366-9. DOI: 10.1145/376957.376986. URL: http://doi.acm.org/10.1145/376957.376986 (cit. on p. 5).

[4] Dominique Attali, Jean-Daniel Boissonnat, and André Lieutier. "Complexity of the delaunay triangulation of points on surfaces the smooth case." In: *Symposium on Computational Geometry*. Ed. by Steven Fortune. ACM, 2003, pp. 201–210. ISBN: 1-58113-663-3. URL: http://dblp.uni-trier.de/db/conf/compgeom/compgeom2003.html#AttaliBL03 (cit. on p. 13).

[5] Franz Aurenhammer et al. *Voronoi Diagrams and Delaunay Triangulations*. 2013 (cit. on pp. 4, 11–13).

[6] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Cláudio Silva, and Gabriel Taubin. "The ball-pivoting algorithm for surface reconstruction." In: *IEEE transactions on visualization and computer graphics* 5.4 (1999), pp. 349–359 (cit. on p. 5).

[7] Paul J Besl and Neil D McKay. “Method for registration of 3-D shapes.” In: *Sensor Fusion IV: Control Paradigms and Data Structures*. Vol. 1611. International Society for Optics and Photonics. 1992, pp. 586–607 (cit. on p. 52).

[8] Tim Bodenmueller. “Streaming surface reconstruction from real time 3D measurements.” PhD thesis. Technische Universität München, 2009 (cit. on p. 7).

[9] Jean-Daniel Boissonnat and Arijit Ghosh. “Manifold reconstruction using tangential Delaunay complexes.” In: *Discrete & Computational Geometry* 51.1 (2014), pp. 221–267 (cit. on p. 5).

[10] Yuri Boykov and Vladimir Kolmogorov. “An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision.” In: *IEEE Trans. Pattern Anal. Mach. Intell.* 26.9 (Sept. 2, 2004), pp. 1124–1137. URL: `http://dblp.uni-trier.de/db/journals/pami/pami26.html#BoykovK04` (cit. on p. 14).

[11] Frédéric Cazals and Joachim Giesen. “Delaunay triangulation based surface reconstruction.” In: *Effective computational geometry for curves and surfaces*. Springer, 2006, pp. 231–276 (cit. on p. 4).

[12] Brian Curless and Marc Levoy. “A volumetric method for building complex models from range images.” In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM. 1996, pp. 303–312 (cit. on pp. 4, 7).

[13] Simon Fuhrmann and Michael Goesele. “Floating scale surface reconstruction.” In: *ACM Transactions on Graphics (TOG)* 33.4 (2014), p. 46 (cit. on pp. 7, 8, 39, 40, 45, 48, 52, 59).

[14] Simon Fuhrmann and Michael Goesele. “Fusion of depth maps with multiple scales.” In: *ACM Transactions on Graphics (TOG)*. Vol. 30. 6. ACM. 2011, p. 148 (cit. on p. 4).

[15] Simon Fuhrmann, Fabian Langguth, and Michael Goesele. “MVE-A multiview reconstruction environment.” In: *Proceedings of the Eurographics Workshop on Graphics and Cultural Heritage (GCH)*. Vol. 6. 7. 2014, p. 8 (cit. on pp. 34, 39–42, 45, 47, 48).

[16] Yasutaka Furukawa and Jean Ponce. “Accurate, dense, and robust multiview stereopsis.” In: *IEEE transactions on pattern analysis and machine intelligence* 32.8 (2010), pp. 1362–1376 (cit. on pp. 40–42).

[17] Michael Goesele, Noah Snavely, Brian Curless, Hugues Hoppe, and Steven M Seitz. “Multi-view stereo for community photo collections.” In: *2007 IEEE 11th International Conference on Computer Vision*. IEEE. 2007, pp. 1–8 (cit. on p. 48).

[18] Adrian Hilton, Andrew J Stoddart, John Illingworth, and Terry Windeatt. “Reliable surface reconstruction from multiple range images.” In: *European conference on computer vision*. Springer. 1996, pp. 117–126 (cit. on p. 4).

[19] Heiko Hirschmuller. “Accurate and efficient stereo processing by semi-global matching and mutual information.” In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 2. IEEE. 2005, pp. 807–814 (cit. on p. 7).

[20] Christof Hoppe, Manfred Klopschitz, Michael Donoser, and Horst Bischof. “Incremental Surface Extraction from Sparse Structure-from-Motion Point Clouds.” In: *BMVC*. 2013, pp. 94–1 (cit. on p. 5).

[21] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. “OctoMap: An efficient probabilistic 3D mapping framework based on octrees.” In: *Autonomous Robots* 34.3 (2013), pp. 189–206 (cit. on p. 4).

[22] Rasmus Jensen, Anders Dahl, George Vogiatzis, Engil Tola, and Henrik Aanæs. “Large scale multi-view stereopsis evaluation.” In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2014, pp. 406–413 (cit. on pp. 39–41).

[23] J Jessup, Sidney Nascimento Givigi, and Alain Beaulieu. “Robust and efficient multi-robot 3d mapping with octree based occupancy grids.” In: *Systems, Man and Cybernetics (SMC), 2014 IEEE International Conference on*. IEEE. 2014, pp. 3996–4001 (cit. on p. 4).

[24] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. “Dual contouring of hermite data.” In: *ACM transactions on graphics (TOG)*. Vol. 21. 3. ACM. 2002, pp. 339–346 (cit. on p. 9).

[25] Michael M. Kazhdan, Allison W. Klein, Ketan Dalal, and Hugues Hoppe. “Unconstrained isosurface extraction on arbitrary octrees.” In: *Symposium on Geometry Processing*. 2007 (cit. on p. 6).

[26] Michael Kazhdan and Hugues Hoppe. “Screened poisson surface reconstruction.” In: *ACM Transactions on Graphics (ToG)* 32.3 (2013), p. 29 (cit. on p. 8).

[27] Ronny Klowsky, Arjan Kuijper, and Michael Goesele. “Modulation transfer function of patch-based stereo systems.” In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE. 2012, pp. 1386–1393 (cit. on p. 8).

[28] Andreas Kuhn, Heiko Hirschmüller, Daniel Scharstein, and Helmut Mayer. “A tv prior for high-quality scalable multi-view stereo reconstruction.” In: *International Journal of Computer Vision* 124.1 (2017), pp. 2–17 (cit. on p. 7).

[29] Chuan-Chu Kuo and Hong-Tzong Yau. “A Delaunay-based region-growing approach to surface reconstruction from unorganized points.” In: *Computer-Aided Design* 37.8 (2005), pp. 825–835 (cit. on p. 5).

[30] Patrick Labatut, Jean-Philippe Pons, and Renaud Keriven. “Efficient Multi-View Reconstruction of Large-Scale Scenes using Interest Points, Delaunay Triangulation and Graph Cuts.” In: *ICCV*. IEEE, Nov. 4, 2008, pp. 1–8. URL: `http://dblp.uni-trier.de/db/conf/iccv/iccv2007.html#LabatutPK07` (cit. on pp. 2, 5, 11, 15, 16, 18, 19, 21, 27, 28, 59).

[31] Patrick Labatut, J-P Pons, and Renaud Keriven. “Robust and efficient surface reconstruction from range data.” In: *Computer Graphics Forum*. Vol. 28. 8. Wiley Online Library. 2009, pp. 2275–2290 (cit. on pp. 2, 16, 18, 20).

[32] Herve Lombaert, Yiyong Sun, Leo Grady, and Chenyang Xu. “A Multilevel Banded Graph Cuts Method for Fast Image Segmentation.” In: *ICCV*. IEEE Computer Society, Nov. 21, 2005, pp. 259–265. ISBN: 0-7695-2334-X. URL: `http://dblp.uni-trier.de/db/conf/iccv/iccv2005-1.html#LombaertSGX05` (cit. on p. 14).

[33] William E. Lorensen and Harvey E. Cline. “Marching Cubes: A High Resolution 3D Surface Construction Algorithm.” In: *SIGGRAPH Comput. Graph.* 21.4 (Aug. 1987), pp. 163–169. ISSN: 0097-8930. DOI: `10.1145/37402.37422`. URL: `http://doi.acm.org/10.1145/37402.37422` (cit. on p. 4).

[34] William E Lorensen and Harvey E Cline. “Marching cubes: A high resolution 3D surface construction algorithm.” In: *ACM siggraph computer graphics*. Vol. 21. 4. ACM. 1987, pp. 163–169 (cit. on p. 8).

[35] Christian Mostegel. *Robust Surface Reconstruction from Noisy Point Clouds using Graph Cuts*. 2012 (cit. on pp. 15, 16, 20, 39).

[36] Christian Mostegel, Rudolf Prettenthaler, Friedrich Fraundorfer, and Horst Bischof. "Scalable Surface Reconstruction from Point Clouds with Extreme Scale and Density Diversity." In: (2017) (cit. on pp. iii, iv, 61).

[37] Patrick Mücke, Ronny Klowsky, and Michael Goesele. "Surface Reconstruction from Multi-resolution Sample Points." In: *VMV*. 2011, pp. 105–112 (cit. on p. 6).

[38] Mathias Rothermel, Konrad Wenzel, Dieter Fritsch, and Norbert Haala. "SURE: Photogrammetric surface reconstruction from imagery." In: *Proceedings LC3D Workshop, Berlin*. Vol. 8. 2012 (cit. on pp. 1, 40–42, 53).

[39] Steven M Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. "A comparison and evaluation of multi-view stereo reconstruction algorithms." In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 1. IEEE. 2006, pp. 519–528 (cit. on pp. 39, 45).

[40] The CGAL Project. *CGAL User and Reference Manual*. 4.7. CGAL Editorial Board, 2015. URL: http://doc.cgal.org/4.7/Manual/packages.html (cit. on pp. 13, 28).

[41] Benjamin Ummenhofer and Thomas Brox. "Global, dense multiscale reconstruction for a billion points." In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 1341–1349 (cit. on pp. 8, 9, 39, 40, 45, 48, 49, 52, 59).

[42] Jörg Vollmer, Robert Mencl, and Heinrich Mueller. "Improved laplacian smoothing of noisy surface meshes." In: *Computer Graphics Forum*. Vol. 18. 3. Wiley Online Library. 1999, pp. 131–138 (cit. on p. 39).

[43] Hoang-Hiep Vu, Patrick Labatut, Jean-Philippe Pons, and Renaud Keriven. "High accuracy and visibility-consistent dense multiview stereo." In: *IEEE transactions on pattern analysis and machine intelligence* 34.5 (2012), pp. 889–901 (cit. on p. 5).

[44] Jian Wei, Benjamin Resch, and Hendrik PA Lensch. "Multi-View Depth Map Estimation With Cross-View Consistency." In: *BMVC*. 2014 (cit. on p. 46).