Rheinische
Friedrich-Wilhelms-
Universität Bonn

Institute for Computer Science
Department VI
Autonomous Intelligent

Rheinische Friedrich-Wilhelms-Universität Bonn

Master Thesis

Textured Meshes for Semantic Mapping

Author:
Radu Alexandru Rosu

First Examiner:
Prof. Dr. Sven Behnke

Second Examiner:
Priv.-Doz. Dr. Volker Steinhage

Advisor:
Jan Quenzel

Date: October 5, 2018

Declaration

I hereby declare that I am the sole author of this thesis and that none other than the specified sources and aids have been used. Passages and figures quoted from other works have been marked with appropriate mention of the source.

Place, Date

Signature

Abstract

Scene understanding is an important capability for robots acting in unstructured environments. While most SLAM approaches provide a geometrical representation of the scene, a semantic map is necessary for more complex interactions with the surroundings. Current methods treat the semantic map as part of the geometry which limits scalability and accuracy.

This thesis proposes to separate the semantic map into a geometrical mesh and a semantic texture. The key idea is that in many environments the geometry can be greatly simplified without loosing fidelity, while semantic information can be stored at a higher resolution, independent of the mesh. We construct a mesh from depth sensors to represent the scene geometry and fuse information into the semantic texture from segmentations of individual RGB views of the scene. Making the semantics persistent in a global mesh enables us to enforce temporal and spatial consistency of the individual view predictions. For this, we propose an efficient method of establishing consensus between individual segmentations by iteratively retraining semantic segmentation with the information stored within the map and using the retrained segmentation to re-fuse the semantics.

We demonstrate the accuracy and scalability of our approach by reconstructing semantic maps of scenes from NYUv2 and a scene spanning large buildings.

Contents

1 Introduction

Robots acting in real-world environments need the ability to understand their surrounding, and know their location within the environment. While the problem of geometrical mapping and localization can be solved through SLAM methods (Zollhöfer et al. 2018), many tasks require knowledge about the semantic meaning of objects or surfaces in the environment. The robot should, for instance, be able to recognize where the obstacles are in the scene and also understand whether those obstacles are cars, pedestrians, walls, or otherwise.

The problem of building maps has been extensively studied (Kostavelis et al. 2015). Most approaches can be grouped into the following three categories based on map representation:

- Voxel-based: The scene is discretized into voxels, either using a regular grid, or an adaptive octree. Each voxel stores the binary occupancy value (occupied, empty, unknown) or the distance to the surface, commonly referred to as Signed Distance Function (SDF).

- Surfel-based: The map is represented by small surface elements, which store the mean and covariance of a set of 3D-points. Surfels suffer from less discretization errors than voxels.

- Mesh-based: The map is represented as a set of vertices with faces between them. This naturally fills holes and allows for fast rendering using established graphics pipelines.

Current semantic mapping systems treat the semantic information as part of the geometry, and store label probabilities per map element (voxel, sufel or mesh vertex/face). This approach has the intrinsic disadvantage of coupling the resolution of the geometrical representation with the semantics, requiring a large number of elements in order to be able to represent small semantic objects or surface parts. This is an undesirable effect as it leads to unnecessary memory usage especially in man-made environments, where the geometry is mostly planar, and high geometrical detail would be redundant. Often, it suffices to represent the semantics relative to a rough geometric shape.

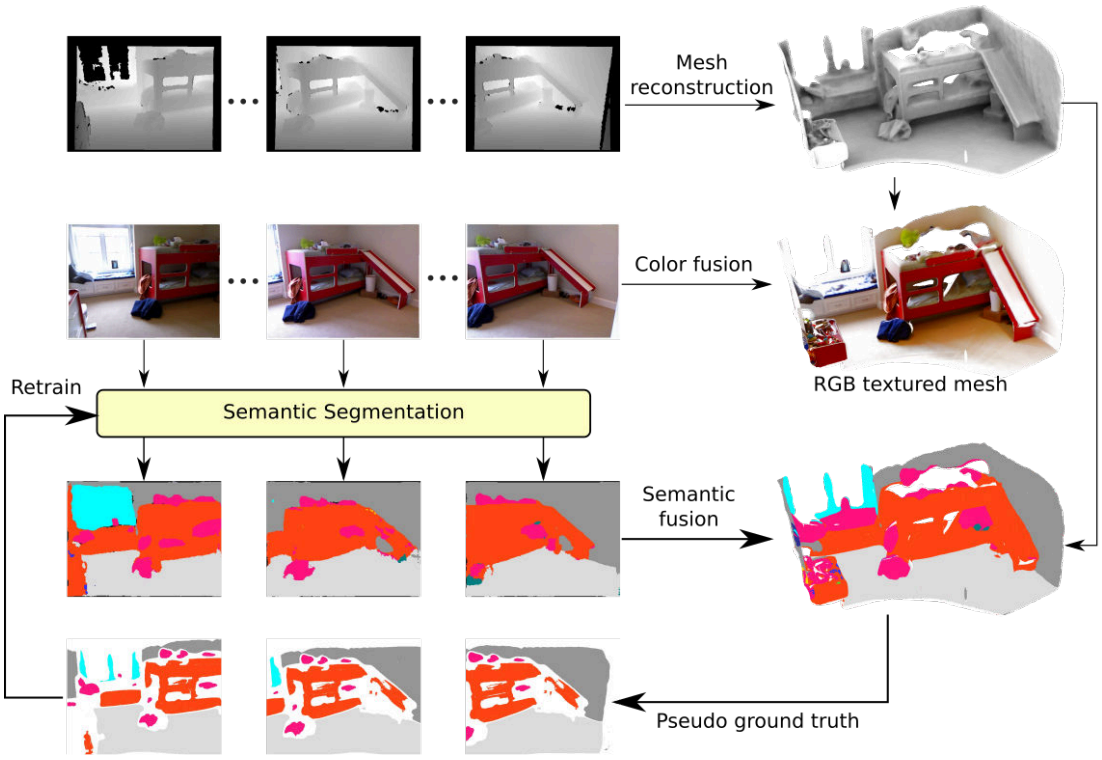


Figure 1.1: **Semantic Reconstruction**: We generate a mesh with RGB texture and semantic annotations. The mesh enables us to ensure temporal and spatial consistency between semantic predictions and allows us to perform label propagation for improved semantic segmentation. Color coding of semantic labels correspond to the NYUv2 dataset (Silberman et al. 2012).

The key idea of our approach, visualized in Fig. 1.1, is to decouple the scene geometry from the semantics by using a semantic texture mesh. In this, the scene geometry is represented by vertices and faces, whereas the semantic texture categorizes the surface with higher resolution. This allows us to represent semantics and geometry at different resolutions in order to build large semantic maps, while still keeping a low memory usage. As our segmentation module we make use of RefineNet (Lin et al. 2017) to predict a semantic segmentation for each individual RGB view of the scene. These predictions are probabilistically fused onto the semantic texture that is supported by a coarse mesh representing the scene geometry. Having a globally persistent semantic map enables us to establish a temporal and spatial consistency that was previously unobtainable for individual-view predictors. To this end, we propose to propagate labels from the stable mesh by projecting onto each camera frame, in order to retrain the semantic segmentation in a semi-supervised manner. Expectation Maximization (EM) is then carried out by alternating between fusing semantic predictions and propagating labels. This

iterative refinement allows us to cope with view points which were not common in the training dataset. For example, a predictor pre-trained on street-level segmentation will not work well on images captured by a micro aerial vehicle (MAV) at higher altitudes or close to buildings. However, projecting confident semantic labels fused from street level onto less confident parts of views will make it possible to learn the semantic segmentation of new viewpoints (see Fig. 5.2).

We compare our method with SemanticFusion (McCormac et al. 2017), and evaluate the accuracy on the NYUv2 dataset (Silberman et al. 2012). We show that the increased resolution of the semantic texture allows for more accurate semantic maps. Finally, propagation and retraining further improve the accuracy, surpassing SemanticFusion in every class.

To showcase the benefits of textured meshes in terms of scalability and speed, we also recorded a dataset spanning multiple buildings, annotated with the 66 classes of the Mapillary dataset (Neuhold et al. 2017). We demonstrate that we are able to construct a large map using both RGB and semantic information in a time- and memory-efficient manner.

2 Related Work

Robotic mapping addresses the problem of building and updating a model of the environment using the sensors available on the robot (cameras, range finders, sonars, GPS, etc.) There have been many solutions proposed to the problem, each tackling the problem in a different way. We will shortly present some of the works in the field.

2.1 Map representations

Storing a representation of the environment is a challenging problem as the choice of underlying data structures comes with certain advantages and disadvantages. We will briefly introduce the most common choices for map representation (be it in 2D or 3D).

2.1.1 Occupancy grid

A popular approach to building spatial maps of the environment is to discretize the space using a grid of cubical volumes of equal size, called voxels. Each voxel of the map stores the probability of being occupied, free space or unknown. As more readings of the environment are obtained (for example a laser range scanner), the occupancy probabilities of the voxels are updated accordingly.

One of the first approaches to building a spatial model of the environment is attributed to Moravec et al. (1985).They used sonar range measurements from a moving robot and aggregated the readings into a 2D occupancy grid. They presented as result a map which stored the probability for each cell of being occupied, empty or unknown. One of the drawbacks of the approach is that the occupancy grid needs to be initialized with the expected map size. In large outdoor areas the extent of the map may not be known beforehand. Also in scenarios where a fine resolution of the grid is necessary, memory consumption can become prohibitive as the scene was represented as a uniform grid.

A more recent approach to occupancy grids can be seen in Octomap (Hornung et al. 2013). They represented the map using a 3D octree which solves part of the

issues occurring in the work of Moravec et al. (1985). The octree defined hierarchical space discretization in which each node represented the volume contained inside it. Each node was recursively subdivided into eight sub-volumes until a certain voxel size is reached. The usage of octrees solved the issue of knowing the map size beforehand by delaying the initialization of the map volumes until measurements needed to be integrated. Also, by storing free space volume only at a very coarse resolution, a more memory efficient representation than the uniform grid was achieved.

2.1.2 Signed Distance Field

Instead of representing the cell occupancy, another approach is to represent the distance of each cell to the surface. This implicit representation of the surface is the idea behind the Signed Distance Field. One of the first approaches that deals with this is the work of Curless et al. (1996) which integrated range images into a cumulative weighted signed distance field, discretized as a uniform voxel grid. The final manifold was extracted as the zero-crossing of the implicit representation (where voxels change from being inside to being outside the volume) using the Marching Cubes algorithm.

More recently, the work of Richard A Newcombe et al. (2011) demonstrated that the Signed Distance Field can be used in a real-time fashion on GPU to reconstruct highly detailed surfaces. They employed a Kinect sensor for obtaining depth images of the scene and integrated them into a Truncated Signed Distance Function (TSDF). The truncated version of the SDF stores distance values only near the surface thus gaining memory efficiency. However the volume to be reconstructed was still limited to room sized scenarios due to the fixed size of the volume grid. This limitation was addressed in the subsequent work of Whelan; Kaess, et al. (2012) which streamed regions of space, that are not mapped anymore, out of the GPU and converts them to a triangular mesh. J. Chen et al. (2013) provided a lossless method for streaming inactive regions to CPU and Richard A. Newcombe et al. (2015) focused on reconstructing dynamic surfaces, like people moving in the scene, by warping a canonical static model into the deforming live frame.

Steinbrücker et al. (2014) also created a volumetric map using TSDF stored on an octree but they provided a fast CPU implementation. Nießner et al. (2013) described a volumetric integration method which does not use an octree but voxel hashing which reduced the voxel lookup and insertion operations from $O(\log n)$ to $O(1)$ (Oleynikova et al. 2017).

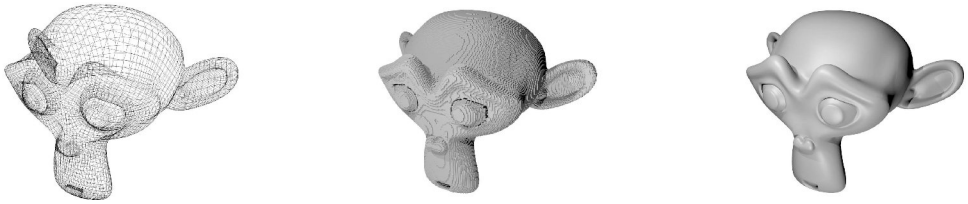


Figure 2.1: **Surface represetations**: Comparison of the same surface represented as mesh(left), occupancy grid(middle) and distance function(right). Taken from Peasley (2013)

2.1.3 Surfel

The space can also be discretized in "surface elements" called surfels which store the mean and covariance of a set of 3D-points. Data can be probabilistically fused inside a surfel map as shown in the work of Stückler et al. (2012) which describes a full SLAM pipeline including loop closure and color reconstruction with a multi resolution surfel map. One of the drawbacks of the approach was that they require an intermediate octree representation which limits scalability and adds computational complexity.

The drawback of using the octree representation was solved in the subsequent work of Keller et al. (2013) which treated the map as a flat array of surfel in which new measurements were fused and denoised over many observations. Their system was capable of real-time execution using a GPU but didn't provide loop-closure capabilities.

Whelan; Leutenegger, et al. (2015) built on top of Keller et al. (2013) and deal with global loop closure by storing a deformation graph of the scene to perform non-rigid deformation.

2.1.4 Mesh

As opposed to surfel or voxel grids, another popular space representation are meshes. A mesh consists of a list of vertices with faces between them. They have the advantage of naturally filling holes and being fast to render using the already established graphics pipeline. Building meshes from a point cloud can be a challenging problem depending on how noisy the sampled points are, the availability of normals for the points, if they densely sample the surface, etc. There have been various works that attempt to solve this problem, the most relevant for this thesis will be mentioned here.

Hoppe et al. (1992) reconstructed a complete mesh from points devoid of normal information by first estimating a tangent plane for each point and creating a Rie-

mannian graph of the points to establish a consistent normal orientation. Finally a Signed Distance Field was built from the points with normals from which a mesh is extracted as the zero crossing of the isosurface.

Marton et al. (2009) proposed a data re-sampling approach which dealt with non uniformly sampled and noisy point clouds. Afterwards, a greedy triangulation was performed in which each point's nearest neighbors were connected with edges.

Ling et al. (2017) built incremental maps from only sparse visual features with the objective of using the map for navigation. Robustly triangulated ORB features were used as vertices for 3D Delaunay triangulation. The tetrahedra generated were evaluated against a visibility constraint which pruned the triangulation, finally yielding a complete mesh of the surface of the mapped environment.

Romanoni; Fiorenti, et al. (2017) proposed a joint meshing and texturing pipeline using a Velodyne 64HD to provide laser range measurements. For meshing they employ the batch version of the space carving algorithm presented in Romanoni and Matteucci (2015). They also proposed a system for removing dynamic objects like cars and pedestrians and obtaining a consistent mesh without them.

2.2 Semantic Segmentation

Semantic segmentation is the task of densely classifying each pixel of the image as belonging to a certain class. It is used in systems which need the ability to recognize the objects in a scene and also the boundaries of each object.

Before deep learning approaches were popular, semantic segmentation was performed using TextonForests (Shotton; Johnson, et al. 2008) and Random Forest based classifiers (Shotton; Fitzgibbon, et al. 2011). This type of classifier applies techniques called ensemble learning, where multiple classifiers were trained and a combination of their hypotheses was used. They were trained for a classification task where each pixel was separately classified by using a small patch around it. The features used to represent each image patch were usually hand crafted.

Long et al. (2015) popularized the idea of using Convolutional Neural Networks (CNNs) for semantic segmentation. They provided a fast way of training by repurposing networks initially trained for the task of image classification. However the pooling layers used in typical classification networks, while they increase the receptive field, aggregating more context from the image, they also discard positional information which is highly necessary for a high resolution semantic segmentation. To obtain a final segmentation, the feature maps from a ImageNet pre-trained network still have to be upsampled because of the pooling layers. Long et al. (2015) solved this issue by proposing an upsampling layer that learns how to best upscale

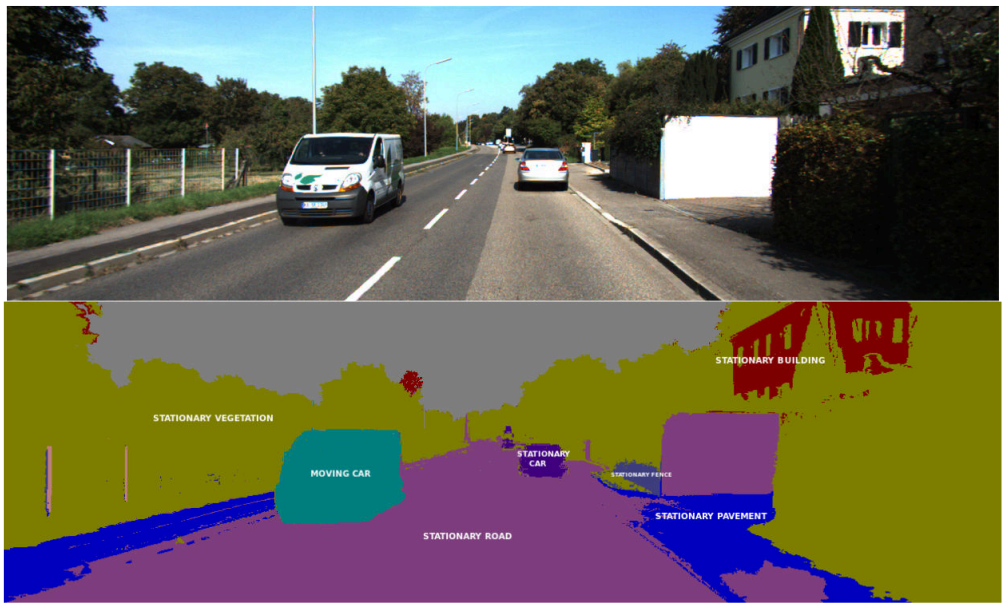


Figure 2.2: **Semantic segmentation**: RGB image and the corresponding semantic segmentation. Taken from Reddy et al. (2014)

the image. However, the segmentation is still coarse due to the loss in information during pooling.

L.-C. Chen et al. (2018) partially solved the problem in loss of information by introducing the atrous or dilated convolution which increases the receptive field without decreasing spatial dimensions and without an increase in parameters. Finally, they also trained a CRF as a post-processing step that smoothed and improved the final segmentation.

Lin et al. (2017) proposed a solution that solves some of the problems of atrous convolution like the increased computational expense and the memory requirements to store the high resolution feature maps. They proposed an encoder-decoder architecture based on pre-trained ResNet (He et al. 2016) which concatenates high-level semantic features from deeper layers of the network with low level ones from earlier convolutions in order to obtain a high precision semantic segmentation. This provided a state of the art segmentation at a low memory cost with high speed.

2.3 Semantic Mapping

With the rise of autonomous robots and cars, the task of creating a semantic map has gained more and more attention in the robotics community. Most approaches use one of the geometrical map representations while storing inside of it the labels

obtained from one of the semantic segmentation approaches. The most relevant works will be shown in this section.

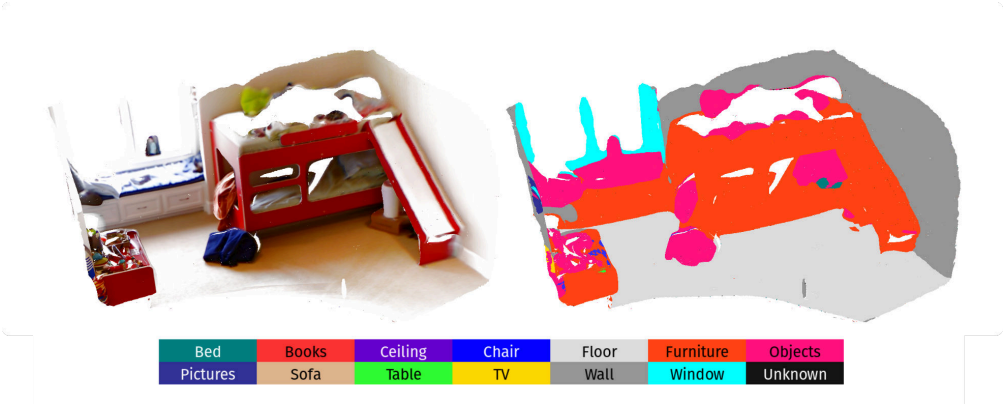


Figure 2.3: **Example semantic map**: Semantic map created by our approach. Color coding of semantic labels correspond to NYUv2 dataset (Silberman et al. 2012)

Civera et al. (2011) paved the way towards a semantic SLAM system by presenting an object reasoning system, able to learn object models using feature descriptors in an offline step and then recognizing and registering them to the map at runtime. However, their system was limited to a small number of objects and, apart from the recognized objects, the map was represented only as a sparse point cloud.

Similarly, Hermans et al. (2014) fused semantic information obtained from segmenting RGB-D frames using Random Forests but represented the map as a point cloud. Their main contribution was an efficient spatial regularizing Conditional Random Field (CRF), which smooths semantic labels throughout the point cloud. Li et al. (2016) extended this approach to monocular video while using the semi-dense map of LSD-SLAM (Engel et al. (2014)). Here, a convolutional neural network (CNN) called DeepLab-CNN (L.-C. Chen et al. 2018) was used instead of a Random Forest for segmentation. Vineet et al. (2015) achieve a virtually unbounded scene reconstruction through the use of an efficient voxel hashed data structure for the map. This further allows them to incrementally reconstruct the scene. Instead of RGB-D cameras, stereo cameras were employed and depth was estimated by stereo disparity. Semantic segmentation was performed through Random Forest. The requirement for dense depth estimates is lifted in the approach of Kundu et al. (2014). They use only sparsely triangulated points obtained through monocular Visual SLAM and recover a dense volumetric map through a CRF that jointly infers semantic category and occupancy for each voxel. A different approach is used in the keyframe-based monocular SLAM system by Tateno et al.

(2017) where a CNN predicts per keyframe the pixel-wise monocular depth and semantic label.

Recently, McCormac et al. (2017) integrated semantics into ElasticFusion (Whelan; Leutenegger, et al. 2015) which represents the environment as a dense surfel map. ElasticFusion is able to reconstruct the environment in real-time on a GPU given RGB-D images and can handle local as well as global loop closure. Semantic labels are stored on a per-surfel basis. Inference is done by a CNN before fusing estimates probabilistically.

Closely related to our approach is the work of Valentin et al. (2013). Their map is represented as a triangular mesh. They aggregate depth images in a Truncated Signed Distance Function (TSDF) and obtain the explicit mesh representation via the marching cubes algorithm. Afterwards, semantic inference is performed for each triangle independently using an aggregation of photometric (color dependent) and geometric (mesh related) features. Spatial regularization is ensured through a CRF over the mesh faces.

2.4 Mesh creation

As mentioned previously, this thesis attempts to decouple the semantic information from the geometric one by means of a textured mesh. The mesh, representing the geometry of the environment, is reconstructed from a series of registered depth sensor point clouds. The choice of meshing algorithm will greatly influence the prior depth processing, therefore we will briefly introduce some of the state of the art approaches in mesh reconstruction, their advantages and disadvantages, and finally our algorithm of choice.

2.4.1 Delaunay triangulation

The Delaunay triangulation of a series $\mathbf{X} = \mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^2$ of points in 3D is a triangulation $DT(\mathbf{X})$ such that no point in $\mathbf{X}$ is inside the circumcircle of any triangle in $DT(\mathbf{X})$. This definition extends naturally to a d-dimensional space, however, we are only interested in the three-dimensional case. In 3D the empty-circle criteria is replaced with the empty circumsphere and the triangulation doesn't produce triangles but rather tetrahedra, hence the 3D Delaunay triangulation is often referred to as tetrahedralization.

A very attractive attribute of the Delaunay triangulation is that it comes with provable guarantees for geometric and topological quality provided the input surface is sampled densely enough. However one big downside of the Delaunay triangulation is it's lack of robustness to noise as it performs only interpolatory

reconstruction in the sense that the computed mesh contains as vertices the same set of points that are used to sample the surface. This poses a problem in the case of imperfect data since the reconstruction doesn't naturally deal with noise or missing data, rendering it impractical for real-world scenarios.

2.4.2 Volumetric integration

Volumetric integration methods represent the surface implicitly as the zero crossing of a Signed Distance Function (SDF). They naturally deal with noise by discretizing the space into a voxel grid and fusing the SDF computed for each individual depth map into a global volume using a weighted average. The update for the weighted average is given by Equation (2.1) where D_i is the SDF integrated up to frame i, with associated weight W_i and the new SDF measurement is given by d_{i+1} with weight w_{i+1}. After the integration of all the depth maps, an explicit mesh can be extracted as the zero crossing of the SDF by a Marching Cubes algorithm.

$$\begin{aligned} D_{i+1}(\mathbf{x}) &= \frac{W_i(\mathbf{x})D_i(\mathbf{x}) + w_{i+1}(\mathbf{x})d_{i+1}(\mathbf{x})}{W_i(\mathbf{x}) + w_{i+1}(\mathbf{x})} \\ W_{i+1}(\mathbf{x}) &= W_i(\mathbf{x}) + w_{i+1}(\mathbf{x}) \end{aligned} \tag{2.1}$$

The whole pipeline has been demonstrated in the work of Richard A Newcombe et al. (2011) where they perform both tracking of the depth sensor and volumetric integration in real-time on a GPU. Despite the maturity of volumetric integration methods, one of its downsides is the inability to cope with missing data or varying sampling density. Therefore, this method proves inadequate for laser scanner sensors which have an inherit low vertical resolution which will result in the undersampling of some surfaces.

2.4.3 Poisson reconstruction

One of the state of the art approaches that create a mesh from orientated points is Poisson reconstruction presented by Kazhdan and Hoppe (2013).

Poisson reconstruction approaches the problem of surface reconstruction by computing a 3D indicator function $\mathcal{X}$ which is 1 inside the model and 0 outside (see Fig. 2.4).

The gradient of this indicator function is zero almost everywhere (since the indicator function is constant almost everywhere) except for points near the surface where the gradient corresponds to the normal of the surface. Thus, the orientated points can be seen as samples of the model's indicator function. Therefore, the problem of finding the indicator function can be formulated as finding the scalar

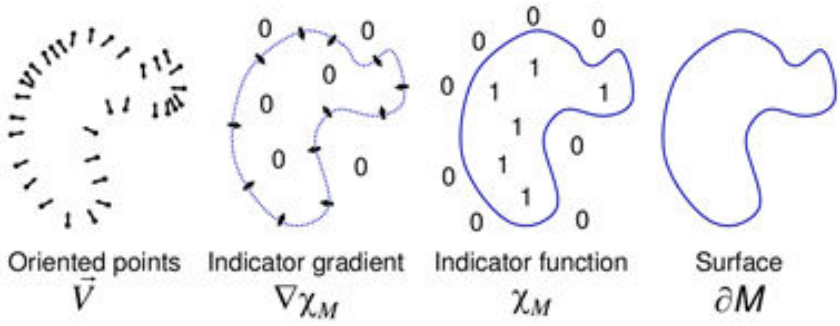


Figure 2.4: **Indicator function**: Poisson reconstruction uses an indicator function to recover the surface of the model as the zero crossing of it. Taken from Kazhdan; Bolitho, et al. (2006).

function $\mathcal{X}$ whose gradients best match the vector field $\vec{V}$ defined by the normals of the points from the point cloud, i.e. $min_{\mathcal{X}} \left\| \nabla\mathcal{X} - \vec{V} \right\|$. By applying the divergence operator we can cast this as a Poisson problem: Find the scalar function $\mathcal{X}$ whose Laplacian (the divergence of the gradient) equals the divergence of the vector field $\vec{V}$,

$$\Delta\mathcal{X} \equiv \nabla \cdot \nabla\mathcal{X} = \nabla \cdot \vec{V} \tag{2.2}$$

To solve this problem the space is discretized into voxels using a hierarchical octree $\mathcal{O}$ allowing for a high resolution representation of the indicator function only near the surface of the model. Now Equation (2.2) can be solved across this discretized space so that each leaf finally stores the values of $\mathcal{X}$ across the reconstructed surface.

Finally, a mesh is extracted from the implicit representation of the indicator function by using a marching cubes algorithm over the leafs of the octree. Triangular faces are created for leafs which experience a sign change (parts of the voxels are inside the surface while other parts are outside). After a full marching cube traversal a watertight and smooth mesh is recovered.

Indicator function methods are an instance of gradient based methods. For surface reconstruction, such a gradient based method provides robustness to noise, non-uniform sampling, and to some extent outliers and missing data. Therefore this makes it a viable solution for meshing the point cloud obtained from a laser scanner.

The full pipeline of the Poisson surface reconstruction can be seen in Fig. 2.5.

One of the first steps in Poisson reconstruction is computing the normals $\vec{N}$ for each point. This is performed by computing a tangent plane $T(\mathbf{x}_i)$ around each point of the input. The tangent plane is represented by it's center $\mathbf{c}_i$ together

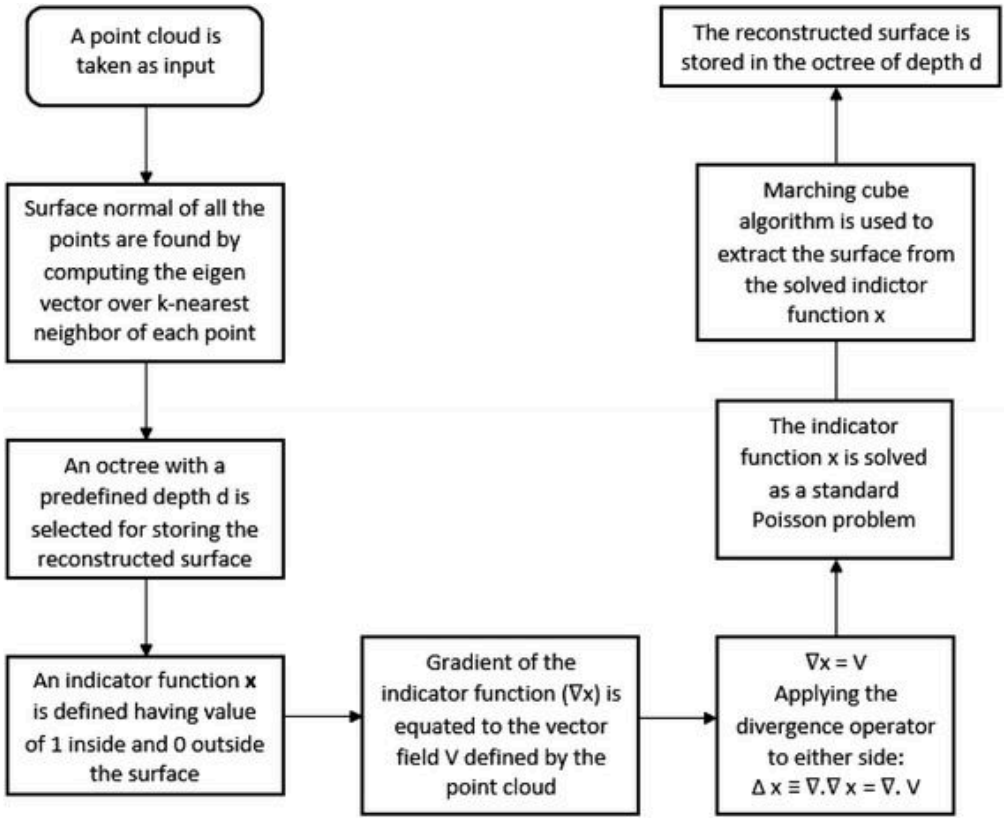


Figure 2.5: **Original Poisson pipeline**: Pipeline for computing a mesh from the orientated points using Poisson reconstruction (Kazhdan and Hoppe 2013). Taken from Maiti et al. (2016).

with a unit normal $\mathbf{n}_i$. The center and normal of $T(\mathbf{x}_i)$ are determined from the k nearest points from the set X to the current point $\mathbf{x}_i$. We denote the k nearest neighbors by $Nbhd(\mathbf{x}_i)$. The number of neighbors k is a user-specified parameter although in practice a value of around 5 works well. The center $\mathbf{c}_i$ of the tangent plane $T(\mathbf{x}_i)$ is taken as the centroid of $Nbhd(\mathbf{x}_i)$. The normal vector $\mathbf{n}_i$ is determined from principal component analysis. First the covariance matrix C of $Nbhd(\mathbf{x}_i)$ is computed, which is a 3×3 positive semi-definite matrix:

$$C = \frac{1}{k} \sum_{\mathbf{p} \in Nbhd(\mathbf{x}_i)} (\mathbf{p} - \mathbf{c}_i) \cdot (\mathbf{p} - \mathbf{c}_i)^T \tag{2.3}$$

If $\lambda_i^1 \geq \lambda_i^2, \geq \lambda_i^3$ denote the eigenvalues of C and $\mathbf{v}_i^1, \mathbf{v}_i^2, \mathbf{v}_i^3$ are the associated eigenvectors, we choose the normal $\mathbf{n}_i$ as either $\mathbf{v}_i^3$ or $-\mathbf{v}_i^3$. The orientation of the normals has to be chosen in such a way that it's consistent with nearby points. An approach to enforce this consistency was presented in Hoppe et al. (1992) by

modelling it as a graph optimization problem on a Riemannian graph.

An example of a mesh reconstructed from Poisson reconstruction is shown in Fig. 2.6.

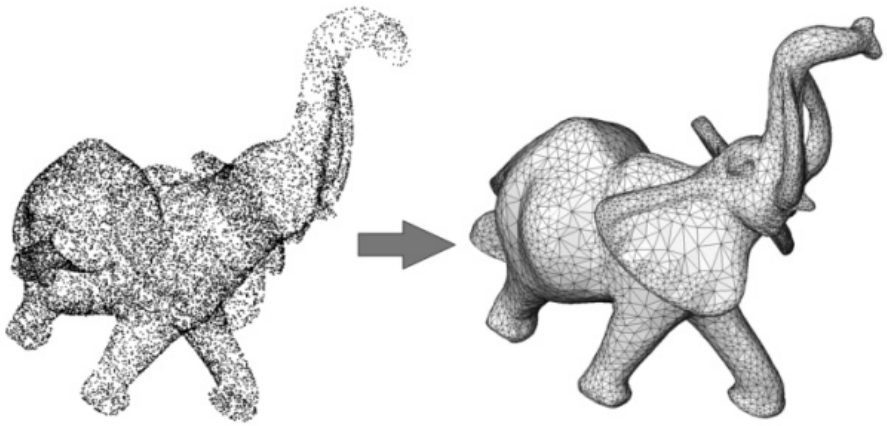


Figure 2.6: **Example mesh reconstruction**: Left: 17K points sampled on the statue of an elephant with a Minolta laser scanner. Right: reconstructed surface mesh. Taken from Alliez et al. (2018).

We choose Poisson reconstruction for our meshing pipeline due to it's natural ability to deal with noisy data and missing measurements while only requiring points with orientated normals as input.

3 Overview

3.1 Pipeline

In this thesis, we present a novel approach to building semantic maps by decoupling the geometry of the environment from its semantics by using semantic textured meshes. This decoupling allows us to store the geometry of the scene as a lightweight mesh which efficiently represents even city-sized environments.

Our method operates in three steps: mesh generation, semantic texturing and label propagation.

Mesh generation We create a mesh of the environment by aggregating the individual point clouds recorded by a laser scanner or a RGB-D camera. We assume that the scans are preregistered into a common reference frame using any off-the-shelf SLAM system. We calculate the normals for the points in each scan by estimating an edge-maintaining local mesh for the scan. Once the full point cloud, equipped with normals, is aggregated, we extract a mesh using Poisson reconstruction (Kazhdan and Hoppe 2013) and further simplify it using QSlim (Garland et al. 1998).

Semantic texturing We first prepare the mesh for texturing by parametrizing it into a 2D plane. Seams and cuts are added to the mesh in order to deform it into a planar domain. A semantic texture is created in which the number of channels corresponds to the number of semantic classes. The semantic segmentation of each individual RGB frame is inferred by RefineNet and fused probabilistically into the semantic texture. We ensure bounded memory usage on the GPU by dynamically allocating and deallocating parts of the semantic texture as needed. Additionally, the RGB information is fused in a RGB texture.

Label Propagation We project the stable semantics, stored in the textured mesh, back onto the camera frames and retrain the predictor in a semi-supervised manner using high confidence fused labels as ground truth, allowing the segmentation to learn from novel view points.

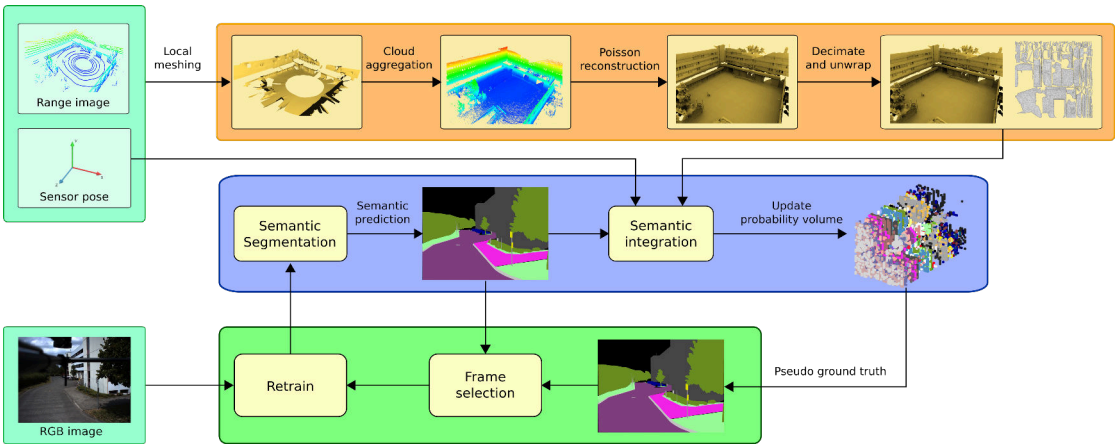


Figure 3.1: **System overview**: Individual range images are used to create local meshes for fast normal estimation and cloud simplification. The resulting points, equipped with normals, are aggregated into a global point cloud. Poisson reconstruction is employed to extract a mesh, which is simplified and unwrapped to obtain a lightweight scene representation. In the next step, we perform semantic integration. Individual RGB views are segmented using RefineNet (Lin et al. 2017). The semantic labels are then probabilistically fused into a semantic texture. Finally, label propagation is performed by inferring pseudo ground truth views, and using them to retrain the predictor. The retrained semantic segmentation is used to re-fuse the labels into the mesh yielding a more accurate semantic map.

Hence, the contribution presented in this thesis is fourfold:

- a scalable system for building accurate meshes from range measurements with decoupled geometry and semantics,
- an edge-maintaining local mesh generation from lidar scans,
- a label propagation that ensures temporal and spatial consistency of the semantic predictions, which helps the semantic segmentation to learn and perform segmentation from novel view points,
- fast integration of probability maps by leveraging the GPU with bounded memory usage.

3.2 Notation

In the following, we will denote matrices by bold uppercase letters and (column-)vectors with bold lowercase letters. The rigid transformation $\mathbf{T}_{F_2F_1}$ is represented as a 4×4 matrix and maps points from coordinate frame F_1 to coordinate frame F_2

by operating on homogeneous coordinates. When necessary, the frame in which a point is expressed is added as a subscript: e.g. $\mathbf{p}_w$ for points in world coordinates. A point $\mathbf{p}_w$ is projected into frame F with the pose $\mathbf{T}_F$ and the camera matrix $\mathbf{K}_F \in \mathbb{R}^{3\times3}$. For the camera matrix we assume a standard pinhole model with focal length f_x, f_y, and principal point c_x, c_y. The projection of $\mathbf{p}_w$ into image coordinates $\mathbf{u} = (u_x, u_y)^\intercal_F \in \mathbf{\Omega} \subset \mathbb{R}^2$ is given by the following mapping:

$$g_F(\mathbf{p}_w) : \mathbf{p}_w \rightarrow \mathbf{p}_F, \tag{3.1}$$

$$(\mathbf{p}_F, 1)^\intercal = \mathbf{T}_{F_w} \cdot (\mathbf{p}_w, 1)^\intercal, \tag{3.2}$$

$$\pi_F(\mathbf{p}_F) : \mathbf{p}_F \rightarrow \mathbf{u}_F, \tag{3.3}$$

$$(x, y, z)^\intercal_F = \mathbf{K}_F \cdot \mathbf{p}_F, \tag{3.4}$$

$$\mathbf{u}_F = (x/z, y/z)^\intercal. \tag{3.5}$$

An image or a texture is denoted by $I(\mathbf{u}) : \mathbf{\Omega} \rightarrow \mathbb{R}^n$, where $\mathbf{\Omega} \subset \mathbb{R}^2$ maps from pixel coordinates $\mathbf{u} = (u_x, u_y)^\intercal$ to n-channel values.

4 Mesh reconstruction

The input to our system is a sequence of organized point clouds $\{\mathcal{P}^t\}$[1], and RGB images $\{I^t\}$ where t indicates the time step. We assume that the point clouds are already registered into a common reference frame, and that the extrinsic calibration $\mathbf{T}_{cd}$ from depth sensor to camera, as well as camera matrices, are given. The depth sensor can be a RGB-D camera or a laser scanner. The output of our system is threefold:

- a triangular mesh of the scene geometry, defined as a tuple $\mathcal{M} = (\mathcal{V}, \mathcal{F})$ of vertices $\mathcal{V}$ and faces $\mathcal{F}$. Each vertex $\in (\mathbb{R}^3 \times \mathbb{R}^2)$ contains a 3D point and a UV texture coordinate, while the mesh face is represented by the indices $\in \mathbb{N}^3$ of the three spanning vertices within $\mathcal{V}$,
- a semantic texture S indicating the texels class probabilities,
- an RGB texture C representing the surface appearance.

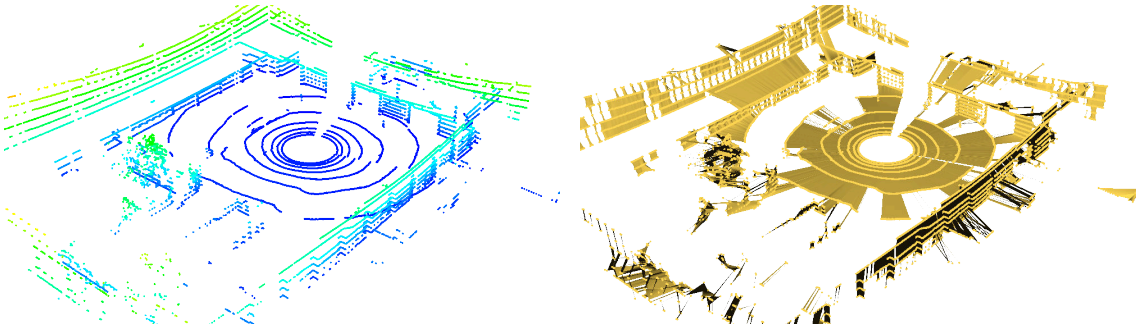
In the next sections of this chapter we will focus on the generation of a global mesh. We start first by describing the necessary depth preprocessing in Sec. 4.1, and continuing with the local mesh simplification (Sec. 4.2) in and refinement (Sec. 4.3); Lastly, we will elaborate on the global mesh generation and parametrization in Sec. 4.4.

4.1 Depth preprocessing

As mentioned previously, our system constructs a global mesh from the aggregation of a series of point clouds $\{\mathcal{P}^t\}$ recorded from a depth sensor. Since we use Poisson reconstruction for mesh extraction we require per-point normal vectors. One way to obtain these normals is by aggregating the full global point cloud, and using the k-nearest-neighbors to estimate the normals for each point. However, this would be prohibitively slow as it requires a spatial subdivision structure, like a Kd-tree, which can easily grow to a considerable size for large point clouds, limiting the scalability of the system. For fast normal estimation we take advantage of the

[1] An organized point cloud exhibits an image resembling structure, e.g. from commodity RGB-D sensors.

structure of the recorded point cloud. Since depth from a RGB-D sensor is typically structured as an image, we can easily query adjacent neighboring points. Similarly, rotating lidar sensors can produce organized scans. A complete revolution of e.g. a Velodyne VLP-16 produces a 2D array of size $16 \times N$ containing the measured range of each recorded point, where N is determined by the speed of revolution of the laser scanner.

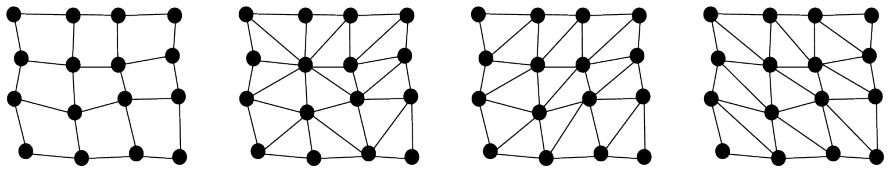


a) Point cloud from lidar scanner b) Reconstructed local mesh

Figure 4.1: **Local mesh:** We reconstruct an approximate local mesh from the given range measurements in order to estimate point normals needed for Poisson reconstruction.

Given the organized structure, we could create a triangular local mesh with approximate normals as introduced by Holz et al. 2014b. They create the mesh by traversing the cloud $\mathcal{P}$ and building a simple quad mesh by connecting every point $p = \mathcal{P}(u, v)$ (v-th point on the u-th scanline) to his neighbors $\mathcal{P}(u, v + 1)$, $\mathcal{P}(u+1, v+1)$ and $\mathcal{P}(u+1, v)$ from the same scan line and the next one (see Fig. 4.2). Afterwards each quad is divided into two triangles using a left cut or a right cut depending on the shortest edge length. After creating the initial triangular mesh, low quality triangles are removed based on the following criteria:

- Triangles, which vertices have point with invalid depth measurements, are



(a) Quad (b) Adaptive (c) Left cut (d) Right cut

Figure 4.2: **Organized cloud connections**: Meshing of an organized cloud and the types of cut that can be applied to a quad. Adaptive chooses the best cut depending on a distance criteria. Taken from Holz et al. (2014a).

removed.

- Triangles, which have a grazing angle with respect to the sensor origin $\mathbf{o} = 0$, are removed.
- Triangles, which have an edge longer than a certain threshold, are removed.

The conditions are formalized in Equation (4.1) where λ_ϕ and λ_d denote the thresholds for grazing angle and edge length respectively.

$$valid = \left(|\cos\phi_{i,j}| \leq \lambda_\phi \wedge (d_{i,j} \leq \lambda_d^2)\right) \tag{4.1}$$

$$\text{with} \quad \phi_{i,j} = \frac{(\mathbf{p}_i - \mathbf{o}) \cdot (\mathbf{p}_i - \mathbf{p}_j)}{\|\mathbf{p}_i - \mathbf{o}\| \, \|\mathbf{p}_i - \mathbf{p}_j\|} \tag{4.2}$$

$$\text{and} \quad d_{i,j} = \|\mathbf{p}_i - \mathbf{p}_j\|^2 \tag{4.3}$$

As it can be observed in Fig. 4.1, meshing provides a rough approximation of the scene geometry. The local mesh will help us in calculating fast normals for each vertex by using the connectivity information of each face. The first step for calculating vertex normals is the estimation of face normals.

Face normals The normals for each face of the mesh are calculated using the cross product between two of its edges. The choice of edges is important as it dictates the orientation of the normals (front-side or back-side). In our approach we take care that the triangles have an anti-clockwise winding order of its vertices, i.e. consecutive vertices of the face rotate anticlockwise around the triangle's center (see Fig. 4.3). Given this convention we take as edges $\mathbf{e}_{1,2} = \mathbf{v}_2 - \mathbf{v}_1$ and $\mathbf{e}_{3,2} = \mathbf{v}_2 - \mathbf{v}_3$. The cross product between them yields the normal vector for that particular face f:

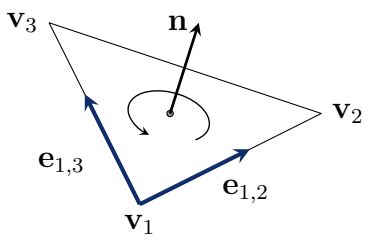


Figure 4.3: **Triangle winding order**: Normal of a triangular face calculated from the cross product of two of it's edges. We assume an anti-clockwise winding order.

$$\mathbf{n}_f = \frac{\mathbf{e}_{1,2} \times \mathbf{e}_{3,2}}{\|\mathbf{e}_{1,2} \times \mathbf{e}_{3,2}\|} \tag{4.4}$$

Vertex normals In computational geometry literature there have been a plethora of methods to compute vertex normals from the associated face normals. However all of them have an easy formula which casts the problem of estimating vertex normals as a weighted average over the adjacent face normals, the difference between

them being in the choice of weights:

$$\begin{aligned} \mathbf{n}_v &= \sum_{f \in AdjF(v)}^{n} \alpha_f \cdot \mathbf{n}_f, \\ \mathbf{n}_v &= \mathbf{n}_v / \left\| \mathbf{n}_v \right\|. \end{aligned} \tag{4.5}$$

where $\mathbf{n}_v$ denotes the normal for vertex $\mathbf{v}$ and the sum runs over the faces that are connected to the vertex. λ is the weight associated with face f. Some of the most common choices of weights are gathered in the work of Jin et al. (2005) which surveys the performance and speed of different algorithms. The conclusion from the survey is that the fastest algorithm is the one which considers each face equally, i.e. uniformly weighted, an algorithm named *Mean Weighted Equally* (MWE):

$$\mathbf{n}_{MWE} = \sum_{f \in AdjF(\mathbf{v})}^{n} \frac{1}{n} \cdot \mathbf{n}_f \tag{4.6}$$

The most accurate one with respect to the estimated normal directions and also the slowest in terms of speed, is the *Mean Weighted by Angle* (MWA) introduced by Thürrner et al. (1998) which weighs each triangles contribution by the angle under which it's incident to the vertex:

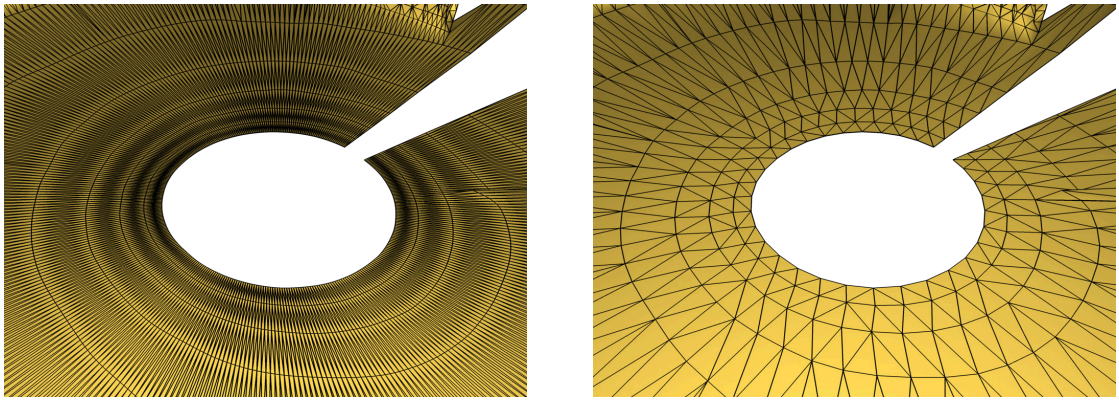
$$\mathbf{n}_{MWA} = \sum_{f \in AdjF(\mathbf{v})}^{n} \alpha_f \cdot \mathbf{n}_f \tag{4.7}$$

where α_f is the angle between the two edge vectors $\mathbf{e}_1$ and $\mathbf{e}_2$ that share the vertex. The angle between the two edge vectors can be computed using the dot product between them: $\alpha_f = \arccos\left(\frac{\mathbf{e}_1 \cdot \mathbf{e}_2}{\|\mathbf{e}_1\| \|\mathbf{e}_2\|}\right)$ Due to the increase in normal accuracy and the fact that the slightly higher computational time is not noticeable in practice, we choose to use the MWA algorithm for vertex normal estimation.

Now that we calculated normals for each vertex we can apply Poisson reconstruction and obtain a global mesh reconstruction. However, the local mesh created from each scan still has some issues which will be discussed in the next sections.

4.2 Local mesh simplification

One of the problems that was observed regarding the creation of the local mesh is that by using the approach of Holz et al. 2014b, the mesh results to be overly dense as they use all the points of the point cloud $\mathcal{P}$ for building the mesh. However, not all the points are necessary for describing the geometry, as most of them lie



(a) Dense mesh (b) Simplified mesh

Figure 4.4: **Mesh simplification**: Overly dense mesh reconstructed from all the points of the range image containing more than 20k vertices (left) and our simplified mesh which contains $\approx$ 3k vertices while still retaining the rough geometrical shape (right).

on common planes and can therefore be simplified. This issue can be especially observed in Fig. 4.4 where a series of scan rings lying on the ground plane result in a unnecessarily dense mesh. Simplifying the local mesh to yield a similar representation but with a lower number of vertices is important for fast and scalable reconstruction.

In order to perform fast simplification and take advantage of the structure of the point cloud and the anisotropy in the sampling we decide to cast the 2D mesh simplification problem to a 1D one by simplifying each scan ring individually. Simplifying each scan ring implies reducing the number of segments which represent it with a subset of it while still maintaining a good approximation of the geometry. For this we choose the Ramer-Douglas-Peucker algorithm for line simplification (Douglas et al. 1973) which takes as input a curve C composed by an ordered list of vertices $\mathbf{v}_0, \mathbf{v}_1, ...\mathbf{v}_n \in \mathbb{R}^n$ and the output is curve $\widetilde{C}$, which contains only a subset of the initial vertices and still represents C well. The methods is best described recursively: to approximate curve spanning from $\mathbf{v}_0$ to $\mathbf{v}_n$, start with line segment $\overline{\mathbf{v}_0\mathbf{v}_n}$. If the farthest vertex from this segment has distance at most ϵ then take that segment as the final one, otherwise split the segment in two around the point which is the furthest. Finally we simplify the two resulting segments recursively. The resulting curve $\widetilde{C}$ is ensured to be within ϵ error of the original one. Algorithm 1 describes it formally and Figure 4.5 shows an example of a dense and simplified curve.

Applying the simplification algorithm to each individual laser ring allows us to severely reduce the total number of vertices in the scan without sacrificing much

Algorithm 1 Simplify a subchain from i to j of an array of vertices V

```
1: procedure SIMPLIFY(V, i, j)
2:     Find the vertex v_k farthest from line v_i v_j
3:         Let dist be it's distance
4:     if dist > ε then
5:         Simplify(V, i, k)
6:         Simplify(V, k, j)
7:     else
8:         Return (i, j)
```

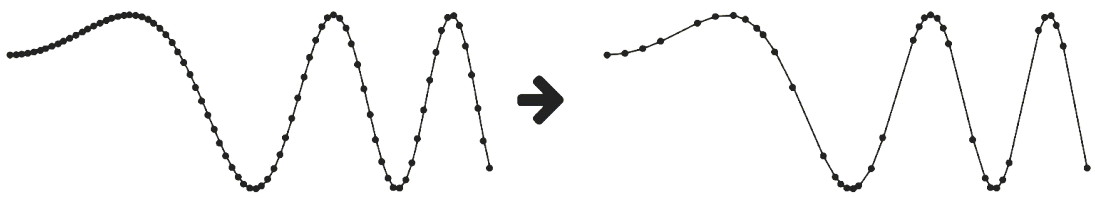


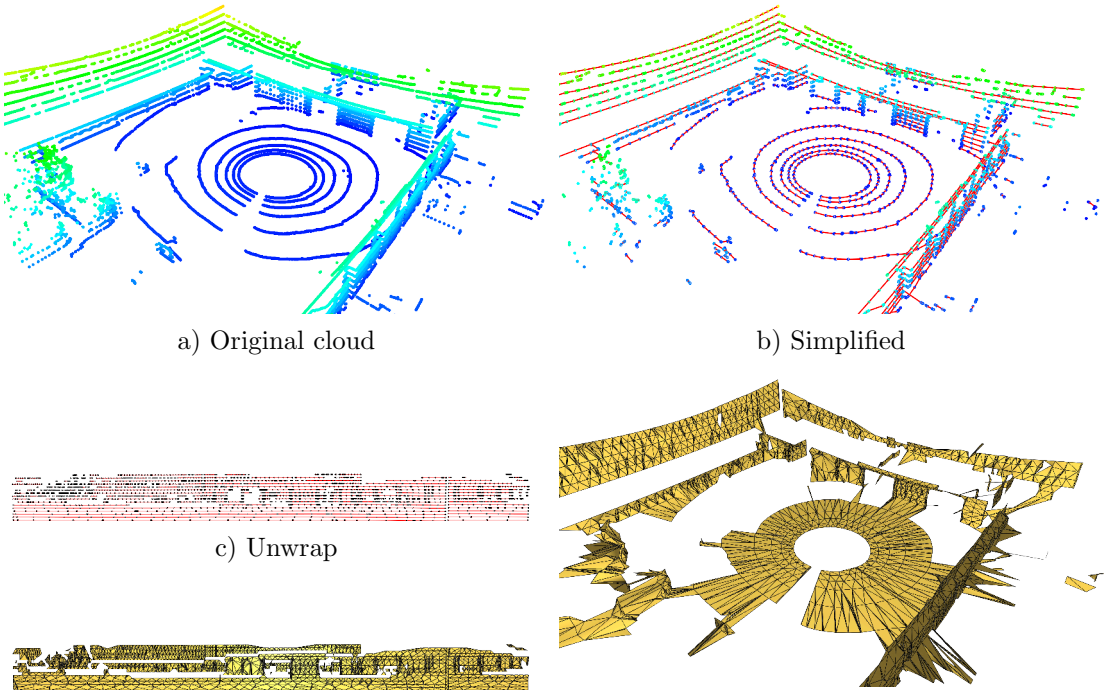
Figure 4.5: **Line simplification**: Curve with a dense number of samples and the corresponding simplified curve after applying the Ramer-Douglas-Peucker algorithm.

of the geometrical accuracy (in our experiment we choose ϵ to be 5cm).

Simplifying each line individually, while fast, has the drawback of making the point cloud lose it's organized structure, meaning we cannot create a mesh just by connecting the immediate neighbors of each point. To solve this issue we propose to cast the meshing process as a Constrained Delaunay triangulation in the planar domain. For this we first "unwrap" the point cloud into the planar domain by transforming each point using spherical coordinates. Each position in space will be represented as three numbers: the radial distance from the fixed origin r, the inclination with respect to the horizontal plane *theta* and the azimuthal angle around the origin *phi*. The transformation to spherical coordinates is described by Equation (4.8) where x, y, z denote the cartesian coordinates of the point.

$$\begin{aligned} r &= \sqrt{x^2 + y^2 + z^2}, \\ \phi &= \arctan(x/-z), \\ \theta &= \arcsin(y/r). \end{aligned} \tag{4.8}$$

Once in the spherical domain, a 2D map of the points is created by taking ϕ and θ as x and y coordinates respectively. The r is set to 0, therefore disregarding the z depth coordinate. In the planar domain we perform a 2D Constrained Delaunay



a) Original cloud b) Simplified

c) Unwrap

d) Delaunay e) Lifted to 3D

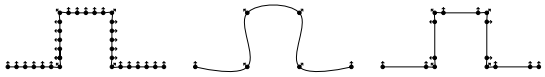
Figure 4.6: **Local mesh pipeline**: Individual point clouds from range images are line simplified using our edge-aware extension of Ramer-Douglas-Peucker (top right). The point cloud is unwrapped using spherical coordinates and a 2D Constrained Delaunay triangulation is performed (bottom left). The resulting mesh is lifted back to 3D and low quality triangles are removed. Pruning of triangles based on their quality follows the method of Holz et al. (2014b) by which triangles which are too grazing with respect to the view direction or too long, are removed.

triangulation of the points, setting the constraint edges as the one obtained from the previous line simplification. The constrained triangulation is a generalization of the standard triangulation in which certain predefined segments are enforced to be present. The enforced segments may cause the triangulation to lose the Delaunay property, however this is not an issue in our case since it will get lifted and further refined in 3D. The constraint edges merely ensure that points in the same scan ring will be connected together by triangles as they are likely to be spatially close in 3D and should contribute to each others normal vector estimation. An illustration of the process is depicted in Fig. 4.6.

A comparison between a dense mesh in which every point of the range image is used and our simplified mesh can be seen in Fig. 4.4.

4.2.1 Ramer-Douglas-Peucker extensions

Despite the obvious benefit of line simplification applied to laser rings, there are still some issues that need to be addressed in order to ensure accurate reconstruction when using an indicator function method like Poisson reconstruction. The search space of indicator functions is restricted to smooth functions with respect to the input points and the given normals. However, during line simplification hard edges like L shaped corners are approximated with only a point at the 90° edge which leaves a normal vector for that point to be tangent to both surfaces incident into it. This enforces the indicator function to create an overly smooth surface around the hard edge which is an undesirable effect. The solution comes by adding offset points at each side of the corner to ensure that the indicator function has normal information from both edges. The effect of the offset points is shown in Fig. 4.7.



(a) Original scanline (b) Simplified scanline (c) Simplified with offset points

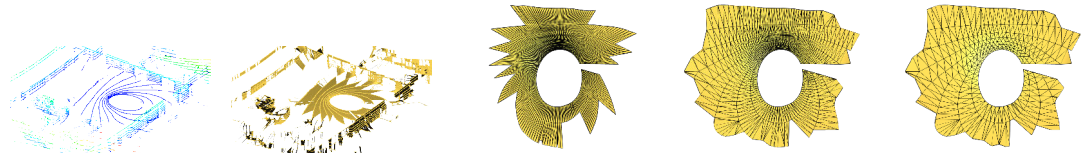
Figure 4.7: **Line simplification:** The original scan line (left) is overly dense in planar areas. The original simplification greatly reduces the number of points but creating a global surface using a method like Poisson reconstruction overly-smooths the edges(middle). Our extension simplifies the line and preserves hard edges by adding further constraints which allow Poisson reconstruction to maintain sharp features (right).

The second extension to the algorithm is to cap the maximum segment length by a certain threshold in order to disallow unreasonably large sub-chains that could possibly span meters. In our implementation we set the maximum length to 30 cm.

Algorithm 2 Simplify a sub-chain from i to j of an array of vertices V

```
1: procedure SIMPLIFY EXTENDED(V, i, j)
2:     length ← ‖v_i v_j‖
3:     Find the vertex v_k farthest from line v_i v_j
4:         Let dist be the distance to the line
5:     if dist > ε and length > λ then
6:         Simplify extended(V, i, k)
7:         Simplify extended(V, k, j)
8:     else
9:         Return (i, i + 1, j − 1, j)
```

The final algorithm for line simplification is shown in Algorithm 2.



a) Point cloud from lidar scanner | b) Local mesh | c) Top view of the ground plane | d) After edge-flipping | e) After edge-flipping and simplification

Figure 4.9: **Local mesh connections problem:** During sudden movements of the laser scanner, the scan rings are compressed behind and expanded in front of the sensor. This creates many small and steep triangles which negatively affects normal estimation. We perform iterative edge-flipping in order to connect each vertex with their closest neighboring vertex, hence, improving the likelihood for estimating correct normals. Furthermore, we apply line simplification to each scan ring independently for data reduction without sacrificing mesh fidelity.

4.3 Local mesh refinement

Until now, we explained how to obtain fast normals from the point cloud of a range image. However, in the case of a laser scanner, their sampling presents high anisotropy since the vertical resolution is order of a magnitude lower than the horizontal one (in our case we use the Velodyne VLP-16 which records 16 scan lines of around 1800 points each). This anisotropy in the sampling poses problems since the triangulation is made in 2D and the connections between the points may not be optimal when lifted back into 3D. This problem can be especially visible during sudden movements of the laser scanner where the scan rings become "compressed" in one direction and "expanded" in the other as shown in Fig. 4.9.

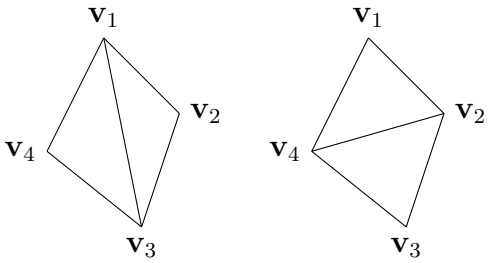


Figure 4.8: **Edge flip**: Example of edge flip in which the edge between $\mathbf{v}_1$ and $\mathbf{v}_3$ (left) gets flipped in order to connect vertices $\mathbf{v}_2$ and $\mathbf{v}_3$ (right).

The 2D triangulation will therefore connect points which are actually far in 3D which may cause the normal estimation for the vertex to be incorrect. Under the assumption that points should be connected to their closest neighbors for reliable and accurate vertex normal estimation, we try to tackle this problem.

One way to overcome this issue would be to use a 3D spatial subdivision structure like a Kd-tree or an octree and search for the closest neighbors of each point in

order to connect them together. However, this would prove slow as the subdivision structure would need to be rebuild for each new scan.

Instead, we propose a faster local approach based on edge flipping that refines triangulation and ensures that vertices are connected to the ones that lie spatially closer in 3D. For this we choose a quality measure for a pair of triangles which we choose to be monotonically increasing and to promote more equilateral triangles. We define it as:

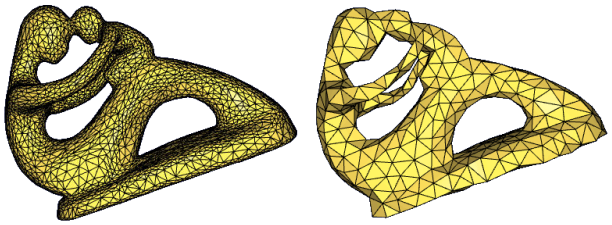
$$q = \frac{4a\sqrt{3}}{h_1^2 + h_2^2 + h_3^2}, \tag{4.9}$$

Bank 1998

where a denotes the area of the triangle and h is the length of the edge. We perform edge flipping in a greedy fashion by choosing first the triangle which will experience the biggest quality increase. After performing the flip for a triangle, the quality of adjacent triangles may change and is updated. We continue flipping edges until the quality measure can no longer be increased.

4.4 Global mesh generation

After the aggregation of the points with corresponding normals from all simplified scans, we perform Poisson reconstruction to recover a high quality mesh, despite having potentially noisy data and missing measurements. Poisson reconstruction has three important parameters to tune for a faithful reconstruction. We will explain some of them more in detail as they can have a big impact on the final resulting mesh.



(a) Dense mesh (b) Simplified mesh

Figure 4.10: **Mesh decimation**: Iterative edge collapsing using QSlim (Garland et al. 1998) greatly reduces the number of triangles while preserving the shape of the mesh.

- Tree depth: The indicator function $\mathcal{X}$ is discretized in an octree $\mathcal{O}$. The depth of the octree dictates how dense the resulting mesh will be. Lower tree depth will result in a lightweight but also overly smooth mesh while higher values will generate a higher resolution mesh that might however show the slight noise in the input cloud. Empirically we found out that values between 9 and 11 work well.

- Position confidence: The confidence that is applied globally for the position

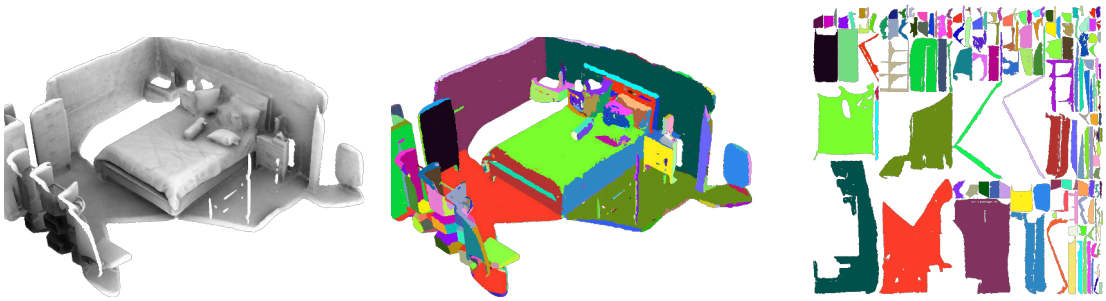
of the points. Higher values result in the mesh aligning more tightly to the input cloud while lower ones smooth the possibly noisy measurement. We found little change by modifying this parameter so we left it as the default one.

- Normal confidence: The confidence to which we trust the normals of the points to be correct. Since our normals are only an approximation, we prefer to lower this parameter more than the default value.

The resulting mesh after Poisson can still be overly dense in areas which are geometrically simple, like the ground. Hence, we apply a second global simplification step following the QSlim method (Garland et al. 1998). This approach iteratively collapses edges of the mesh until a certain error threshold, or a predetermined number of faces, is reached. The error used to determine if an edge will be collapsed or not is a quadratic energy which implicitly tracks how much error is committed by collapsing the edge by looking at how much the adjacent planes of vertex change. Each edge of the mesh is placed into a heap sorted by the cost, so the edge with the minimum cost is always on top. The algorithm iteratively removes the first element from the heap, performs the contractions and updates the cost of the adjacent edges. An example of a mesh decimated using QSlim is shown in Fig. 4.10.

The last step in the creation of the global mesh is to prepare it for texturing by parametrizing the mesh in the 2D domain in order to obtain UV coordinates for the vertices. For that, we make use of the *UV smart project* function provided within Blender[2]. The Blender unwrapping initially computes a set of planar regions (sometimes called charts or islands) by using a breadth-first search along the mesh. The planar regions are then individually parametrized in 2D by using an orthogonal projection. Finally all the charts are packed together into a final UV map by approximating their area by their corresponding bounding box and aligning the bounding boxes using a greedy algorithm. The steps of the UV unwrapping are shown in Fig. 4.11.

[2] https://www.blender.org



a) Mesh b) Planar segmentation c) UV parametrization

Figure 4.11: **UV parametrization:** Planar unwrapping is performed by segmenting the mesh into roughly planar regions and packing them into a 2D map.

As the parametrization is a critical step for creating a texture map, we also experimented with other tools for creating it.

The work of Poranne et al. (2017) computes an UV parametrization by jointly optimizing both UV positions to minimize distortion and seam positions. Since the optimization contains both a continuous (for the distortion) and a discrete term (for the positions of the seams) the problem is highly non-smooth and non-convex. They tackle the problem by using homotopy optimization which initially starts with a smoother energy function which is iteratively sharpened, helping with the convergence towards a minima. However, we found that their method has high memory requirements as they treat the mesh as a triangle soup (i.e. vertices are replicated as per the number of degrees). Also the method is not fully automatic and they require the intervention of a human to resolve possible charts that overlap in the UV space. For these reasons we discard this method as a possible algorithm for our purposes.

Thekla Atlas[3] is a tool mainly designed for lightmapping but can be used for the general case of UV mapping. While the tool works quite well, we found that it provides little control over the output and that the documentation is quite sparse. It also offers no guarantees that the charts won't overlap.

Zbrush UV Master[4] is a tool that seems to provide fully automatic UV parametrization with a guarantee of no overlaps. However it's licensed as a plug-in to a commercial tool which limits it's usage in our case.

For all of these reasons, we chose Blender for creating our UV maps as the tool is freely available and the parametrization can be created automatically with no intervention from the user. A number of batch scripts were also written for this thesis in order to fully automatize the process.

[3] https://github.com/Thekla/thekla_atlas
[4] http://pixologic.com/zbrush/features/UV-Master/

5 Semantic and Color Integration

In this chapter we will focus on the integration of the semantic and color information into the mesh and the creation of the semantic texture S and the RGB texture C.

We will start by detailing the semantic integration (Sec. 5.1) and the color integration (Sec. 5.2). We continue with the sparse semantic volume integration in Sec. 5.3 which helps alleviate the memory problems of the naive integration scheme. We elaborate on the label propagation algorithm in Sec. 5.4 and we finally conclude with implementation details in Sec. 5.5.

5.1 Semantic Integration

In this section we detail our approach on how to update the global semantic texture S using individual color images I^t. For semantic segmentation, we train RefineNet on the Mapillary dataset (Neuhold et al. 2017) for street level segmentation. The dataset contains 25 000 images densely labelled with 66 classes. Given the input image I_k, the output of the predictor can be interpreted as a per-pixel probability over all the class labels $P(O_{\mathbf{u}} = l_i | I_k)$, with $\mathbf{u}$ denoting pixel coordinates.

One common approach to integrate semantic information is to perform a Bayesian update over the classes probability, fusing new observations into the global belief for the semantic labels. However, this scheme of updating can quickly become slow for a large number of classes since the belief for all labels needs to be updated.

In our approach we choose to approximate the probability over the classes with only the *argmax* probability. Hence, a new observation will consist of only the label with the highest confidence and its corresponding probability instead of the full distribution. This approximation is valid as modern neural networks tend to be overconfident in their predictions as analysed in the work of Guo et al. (2017). They conclude that modern neural networks, due to a combination of increased depth, batch normalization and lack of regularization tend to output overly confident output despite the fact that the accuracy of the predictions may be lower than what the uncertainty measure may lead us to believe. Due to this, we argue that we can simplify the full Bayesian update by only the argmax class

significant loss of information. This enables us to use a fast integration scheme with run-time independent to the number of classes. We observe that in practice this approximation works well.

We define the best class L and its corresponding probability P^* using:

$$\begin{aligned} L &= \underset{c}{\operatorname{argmax}}\, P(O_u = l_i | I_k), \\ P^* &= \max_c P(O_u = l_i | I_k). \end{aligned} \tag{5.1}$$

We perform rendering of the mesh as it lies in texture space so as to effectively rasterize every point on it that is covered by a texel. This provides us with texel values which inherit the attributes of the containing faces by means of barycentric interpolation. Therefore we obtain a map of texels where each texel x lies at a certain coordinate $\mathbf{u}_x$ in UV space and inherits the position in world coordinates $\mathbf{p}_x$ from the vertices of the corresponding mesh face.

We perform a visibility check prior to updating the global semantic texture using individual segmentation results. Inspired by *shadow mapping* techniques in computer graphics, we first render a depth map D from the current camera view. Afterwards each texel's world coordinates $\mathbf{p}_x$ is projected into the current view, and discarded if it's depth d_x is larger than the stored value within the depth map $D(\pi(g_F(\mathbf{p}_x)))$, as it lies behind the visible part of the mesh. To indicate visibility, we use a per texel indicator variable $r_x \in \{0, 1\}$:

$$r_{x_i} = \begin{cases} 1, & \text{if } d_x \geq D(\pi(g_F(\mathbf{p}_x))) \\ 0, & \text{otherwise} \end{cases}. \tag{5.2}$$

All remaining texels ($r_x > 0$) are fused with the current segmentation result by increasing the probability of the obtained classes:

$$S(\mathbf{u}_x, l)^t = S(\mathbf{u}_x)^{t-1} + r_{x_i} \cdot w_{x_i} \cdot p_{x_i}, \tag{5.3}$$

$$W(\mathbf{u}_x)^t = W(\mathbf{u}_x)^{t-1} + r_{x_i} \cdot w_{x_i}, \tag{5.4}$$

$$l_{x_i} = L(\pi(g_F(\mathbf{p}_x))), \tag{5.5}$$

$$p_{x_i} = P^*(\pi(g_F(\mathbf{p}_x))). \tag{5.6}$$

Additionally, we weigh the fused probability by the face's distance from the camera under the assumption that pixels are more difficult to recognize from farther

away, due to the low resolution of semantic segmentation.

$$w_{x_i} = \begin{cases} 1, & \text{if } d_x \leq d_{min} \\ 1 - \frac{d_x - d_{min}}{d_{max} - d_{min}}, & \text{otherwise} \end{cases}, \tag{5.7}$$

where d_x denotes the depth of the current texel, and d_{min} and d_{max} are thresholds for the distance which define a linear fall-off for the weight. In our experiments we set them to 30 m and 100 m, respectively.

5.2 Color Integration

In addition to the semantics, we also fuse the raw images into a global color texture. The fusion is carried out by a weighted running average:

$$\begin{aligned} C(\mathbf{u}_x)^t &= \frac{W(\mathbf{u}_x)^{t-1} C(\mathbf{u}_x)^{t-1} + w_x I(\pi(g_F(\mathbf{p}_x)))}{W(\mathbf{u}_x)^{t-1} + w_x}, \\ W(\mathbf{u}_x)^t &= W(\mathbf{u}_x)^{t-1} + w_x. \end{aligned} \tag{5.8}$$

The weight w_x takes the distance, the radial intensity fall-off within an image, and the viewing angle into account:

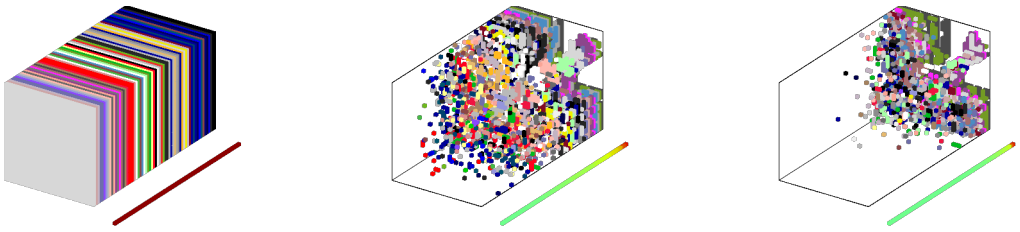
$$\begin{aligned} w_x &= w_{dist} \cdot w_{vign} \cdot w_\phi, \\ w_{dist} &= (\|g_F(\mathbf{p}_{w,x})\|_2^2)^{-1}, \\ w_{vign} &= \cos(\theta_x)^4, \\ w_{view} &= (\mathbf{o}_w - \mathbf{p}_{w,x}) \cdot \mathbf{n}_x. \end{aligned} \tag{5.9}$$

Here, w_{dist} is the inverse distance from the texel to the camera, which promotes frames that are spatially closer to the mesh, improving the resolution of the fused colors. The viewing angle θ_x between reprojection of the texel and the principal axis of the camera is used to account for the radial decrease in intensity by following the $\cos^4$ law (Goldman et al. 2005). The third term w_{view} increases the weight for texels that are viewed by the camera originating at $\mathbf{o}_w$ from a frontal perspective, further improving the quality of the fused texture.

We choose different update schemes for color and semantics as their behaviour is radically different. Firstly, the semantic segmentation is trained to be robust to illumination changes, hence the vignetting term is unnecessary. Secondly, it is not clear that the angle to the surface is a good indicator for confidence in semantic segmentation. In our experiments, we observed that the predictor learns to some extent the relative angle of surfaces with respect to camera view, thus

weighting based on relative angle may be adversarial. For example during sudden camera movements tilting toward the ground, the semantic segmentation decreases in accuracy as the view is unfamiliar to the predictor. Thirdly, the distance weight assumes that the accuracy of the semantics is more confident for closer surfaces. However, this is not accurate for large semantic entities like buildings for which the accuracy decreases as we go closer, due to large untextured areas, and increases as we take a step back and observe the bigger picture.

5.3 Sparse Semantic Volume



a) Dense volume b) Sparse, only committing c) Sparse

Figure 5.1: **Semantic probability volume for the courtyard dataset:** The memory consumption of the dense 3D semantic texture (left) is prohibitive for most modern GPUs. Committing memory pages with non-negligible probabilities results in a sparse volume (middle) with only 12.29 % allocated. Periodically removing low probability pages (right) further reduces the necessary memory (4.25 %) and ensures bounded memory usage that easily fits into GPU memory.

The semantic 3D texture S contains for each texel the probability distribution over all the classes. However, for reasonable sized resolutions and number of classes, this volume can occupy more memory than is typically available in modern GPUs, rendering this process infeasible. For a texture with 8.192×8.192 pixels and the 66 classes of the Mapillary dataset, we would need to allocate a volume of 16.5 GB (assuming we store each element as a floating point number of 4 bytes). This problem will only become worse as we add more class labels or increase texture size. In order to overcome this issue, we propose to store the semantics into a sparse 3D texture in which we allocate and deallocate dynamically the memory, ensuring bounded memory usage.

In the first step, we divide our global semantic volume into pages of size $128 \times 128 \times 1$. Each page[1] stores the probability for only one class and can be either

[1]Page size was chosen based on commonly supported values for multiple computers used during development.

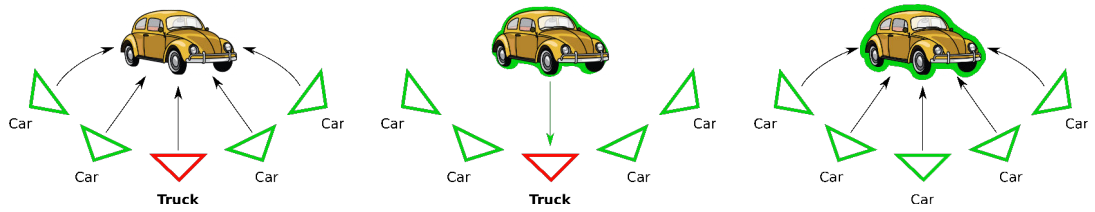
allocated in GPU memory or not. The volume starts initially with all pages in a deallocated state. Hence, it occupies no space on the GPU.

When fusing the semantic probability from the current frame into S, the corresponding pages that will be affected are computed and targeted for committing on the CPU. This ensures that we only add the parts that are actually relevant. After each frame, we also check for pages that have low probability and deallocate them from memory. The probability for a texel is computed as:

$$p_{x_l} = S(\mathbf{u}_x, l)/W(u_x). \tag{5.10}$$

If any texel inside a page is above a certain threshold, we will keep the corresponding page in memory, otherwise we target it for decommitting. This scheme of committing and decommitting portions of the memory can be seen as intrinsically *tracking* the modes of the distribution over the classes, ignoring parts with negligible probability.

5.4 Label Propagation



a) Fuse b) Retrain c) Re-fuse

Figure 5.2: **Label propagation:** semantic segmentations are probabilistically fused in the scene (left). Frames with the largest deviation (middle) from the stable mesh are used for retraining. Inference is repeated for all images and re-fused into the scene mesh to achieve a more accurate semantic map.

The fusion of semantic information from various view points into a global representation opens up possibilities to enforce temporal and spatial consistency of the semantic predictions by propagating the labels. The key insight is that if the majority of observations of the texture element predicts the correct class, the fused information S^* will be confident enough ($p_{x_l} \geq p_{min}$) to be used as ground truth. Hence, we can reproject the mesh into any camera frame, propagate the label $\widetilde{L}(\mathbf{u}_F)$, and retrain the semantic segmentation in a semi-supervised fashion to minimize the discrepancy between image predictions L and mesh label S^*.

Reprojecting the semantics into a camera frame is performed as follows:

$$\begin{aligned} \mathbf{u}_F &= \pi(g_F(\mathbf{p}_x)), \\ S^* &= \underset{c}{\operatorname{argmax}}\, S, \\ S_F^*(\mathbf{u}_F) &= S^*(\mathbf{u}_x), \\ \widetilde{L}(\mathbf{u}_F) &= \begin{cases} S_F^*(\mathbf{u}_F), & p_{x_l} \geq p_{min} \\ \text{Unlabeled}, & \text{otherwise} \end{cases}. \end{aligned} \tag{5.11}$$

The result consists of the image $\widetilde{L}(\mathbf{u}_F)$ which we denote as *pseudo ground truth*. This semantic labelling, together with the corresponding RGB view I, will be used to retrain the predictor.

The retraining ensures that the semantic segmentation will segment objects and surfaces more consistent as belonging to a certain class, regardless of different or extreme view points or even illumination changes. Furthermore, the label propagation and retraining stages can be applied iteratively in an Expectation Maximization scheme, e.g. for a certain number of iterations, or until all camera frames reach a consensus. This process is illustrated in Fig. 5.2. The obvious caveat are wrong predictions used for retraining since bootstrapping this has a self-reinforcing character. We have seen this behaviour only in some rare cases where the initial single-frame predictions were already incorrect, e.g. the table in the bottom row of Fig. 6.4.

Due to the significant number of camera frames contained in a large-scale dataset, we perform a conservative frame selection in order to choose only a subset of frames on which to reproject the semantics for retraining. The frame selection is based on an inconsistency coefficient, which rates frames higher the more the instantaneous semantic segmentations deviates from mesh semantics.

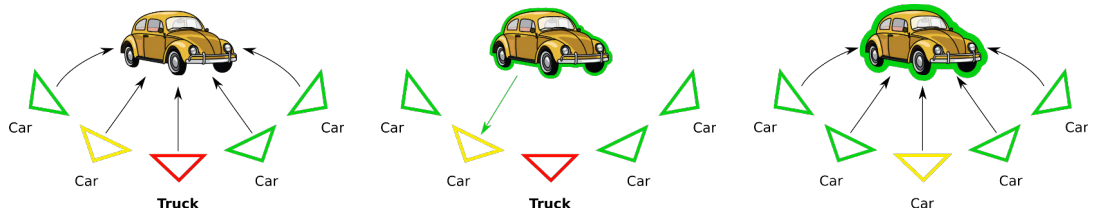
The inconsistency coefficient γ is calculated as:

$$\begin{aligned} \gamma &= \sum_i B(L_i, S_F^*(\mathbf{u}_F))P^*, \\ B(L_i, S_F^*(\mathbf{u}_F)) &= \begin{cases} 1, & L_i \neq S_F^*(\mathbf{u}_F) \\ 0, & L_i = S_F^*(\mathbf{u}_F) \end{cases} \end{aligned} \tag{5.12}$$

Given this coefficient for each frame, we select a restricted percentage of frames with the highest coefficient (in our experiments we choose 5 % of the frames for NYU and 15 % for the courtyard dataset) to be used for retraining. The intuition is that there is more information to gain by retraining on inconsistent frames as opposed to the ones which are already correct. However, in order to avoid

forgetting during retraining, we add the original training set (in our case all the images from the Mapillary dataset or the images from the NYU training set) to the new views. Moreover, we also restrict the selected views to be at least 10 frames apart from one another in order to avoid adding redundant views that are too similar.

Additionally, we also experiment with another type of label propagation in which we select for retraining not the frames with the highest inconsistency but rather the ones with the lowest. The intuition behind this alternative scheme is that by reinforcing the good frames (or the ones that are close to being good), the other frames that are "close" to them will also naturally be improved. Therefore performing this label propagation iteratively will eventually "lift" the less consistent frames to become better by reinforcing the ones that are already good or close to being good. An illustration of this process is depicted in Fig. 5.3. Also during this type of label propagation we ignore the frames which have too high of a consistency level as we saw that in the general case they provide little information. Rather we focus on the frames that are in the "middle zone", meaning that they are quite consistent but not fully. Therefore, we ignore the frames which have more than 98% of the pixels labelled as consistent with the mesh. From the rest of them we select the same percentages as mentioned earlier.



a) Fuse b) Retrain c) Re-fuse

Figure 5.3: **Label propagation with best frames:** semantic segmentations are probabilistically fused in the scene (left). Frames with *low* (but not too low) deviation from the stable mesh are used for retraining (middle). The frames that are close are naturally improved through the retraining. Inference is repeated for all images and the semantic labels are re-fused into the scene mesh to achieve a more accurate semantic map (right).

5.5 Implementation

Our pipeline consists of three modules, the mesh generator, the semantic texture integrator and the segmentation retraining. The mesh generator module is fully implemented in C++ and integrated into ROS (Quigley et al. 2009) to simplify interaction with other ROS packages. The texturer was also developed in C++

with the addition of OpenGL for rendering and semantic integration using GLSL compute shaders.

We will describe the optimization choices made for semantic integration in detail as they are the main focus of this work. The segmentation map is initially precalculated from the predictor and stored to disk. An asynchronous module reads the RGB images and the corresponding segmentation maps and stores them in ringbuffers ready to be processed by the texturing module. The texturing module receives the images and transfers them to the GPU using double buffered Pixel Buffer Objects (PBO) in a method commonly known as *ping ponging*. This ensures that the transfer can be done on the GPU side using Direct Memory Access (DMA), freeing the CPU to do other tasks in the meantime. The semantic integration is performed fully on the GPU by efficient compute shaders. However, the committing and decommitting of pages for the sparse semantic volume can only be performed from the CPU side, which requires synchronization and communication between CPU and GPU. We use two buffers for this communication, one for signalling to the CPU which pages require committing and one for confirmation to the GPU that they were committed. These buffers are updated asynchronously and are also double buffered in order to avoid stalling the pipeline. While double buffering allows maximum usage of the available resources, it also implies a delay of one frame between the CPU-GPU communication which in our case does not pose a problem due to the high frame rate at which the camera images arrive.

6 Experiments

All tests were performed on a Intel Core i7-940 2.93 GHz CPU with an NVIDIA Titan GPU.

6.1 NYUv2 Dataset

For comparison against SemanticFusion (McCormac et al. 2017) we utilize the NYUv2 dataset (Silberman et al. 2012) and use 108 out of all sequences where ElasticFusion did not exhibit significant drift. For comparability, we use the final surfel map for meshing and the segmentation module of Eigen et al. 2015 used within SemanticFusion that is trained on the 13 NYU classes. Furthermore, we store the semantics for Intersection-over-Union (IoU) calculation after the scene is completed and fused semantics are static. In contrast to McCormac et al. 2017 we only use the RGB-CNN since we want our method to work well even in outdoor scenarios where dense depth may not be obtainable.

The network is retrained using both methods of LP (retraining on worse frames or on the best frames). In the case of retraining with the worse frames we add from each NYU scene 5 % of the frames with the highest inconsistency coefficient. We chose this in order to have a number of pseudo ground truth frames comparable to the 795 originally used for training. The retraining then uses both the original training set and our pseudo ground truth.

In the case of retraining with the best frames we choose the 5 % frames with the *lowest* inconsistency coefficient, ignoring however those that have too few inconsistent pixels (in our case we choose those that have at least 2 % of the pixels labeled as inconsistent). In both cases we use a learning rate of 1×10^{-6} and the model is saved after each epoch. The training is stopped when the model begins to overfit and the IoU for the epoch starts to decrease.

6.2 Courtyard Dataset

The courtyard dataset (Droeschel et al. 2018) was captured using a DJI Matrice 600, with a horizontally attached Velodyne VLP-16 laser scanner. The lidar has

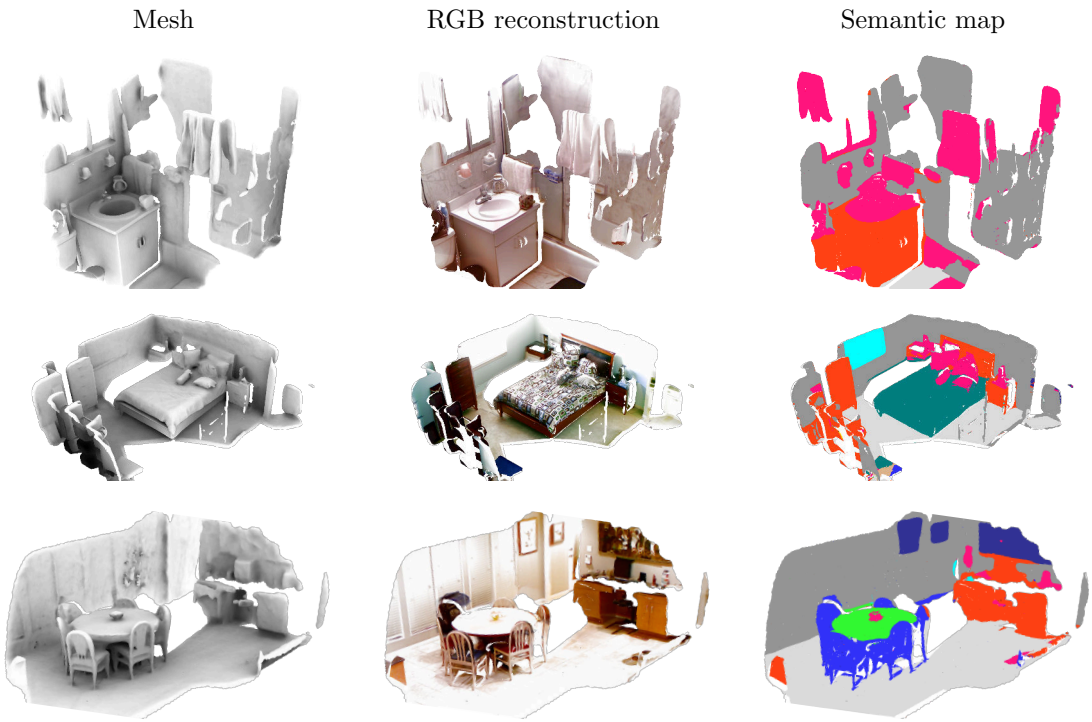


Figure 6.1: **Scenes from NYUv2**: The mesh (left) is reconstructed from the surfel map of ElasticFusion and textured with RGB appearance (middle) and semantic labels (right).

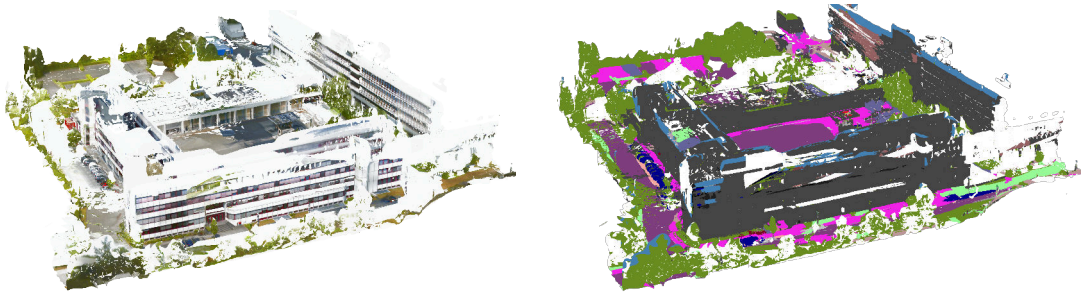
16 horizontal scan lines and a vertical field-of-view of 30° with a maximum range of 100 m. The color images were captured at 10 Hz using two synchronized global shutter Point Grey Blackfly-S U3-51S5C-C color cameras, with a resolution of 2448×2048 pixels. The MAV poses for this dataset were provided by Droeschel et al. (2018). The poses were represented as the transformation between the world coordinate system and the laser scanner. The poses for the two cameras were calculated from the provided extrinsic calibration between the laser scanner and the cameras. However, due to the delay in transferring the camera images from the sensor to the onboard PC, and the fact that the laser scanner poses may not exactly align time-wise with the time when the camera images arrive, the computed poses were not very reliable. To solve this issue, a continuous-time trajectory represented as a spline was computed that interpolates the poses in between the discrete ones provided for the laser scanner. Therefore, for a single camera image, together with its corresponding timestamp, we query the poses on the spline with a 40 ms delay in the past in order to account for said processing and transfer duration. Empirically we saw that this delay was the one that provides the most accurate poses. The two cameras are mounted outwards and pointing to the left and right side of the copter to improve visual coverage. In total 13 458 frames were captured during the experiment.

We densely annotated 48 images spread throughout the area to conduct accuracy experiments.

RefineNet with ResNeXt-101 (Xie et al. 2017) as a feature extractor was trained on the Mapillary dataset for 50 epochs with a learning rate of 10^{-5} and a batch size of 1 using Adam as a gradient descent optimizer. The retraining with pseudo ground truth has to be done with a larger batch size of 16 to account for the decreased signal-to-noise-ratio introduced by the pseudo ground truth. We perform only one iteration of LP consisting of five epochs using the worse frames as the improvement for more iterations would not be significant. We use the SGD optimizer as Adam proved to be too aggressive for the task of fine-tuning.

6.3 Accuracy Evaluation

We execute the two variants of Label Propagation for three iterations on the NYUv2 dataset. We observe that retraining on the worse frames yields a higher IoU hence we prefer this option for all further experiments. Furthermore the highest increase in IoU is experienced after the first iteration, while the subsequent ones yield a noticeably less improvement.



RGB reconstruction Semantic Map

Figure 6.2: **Courtyard semantic map**: The courtyard is reconstructed from a simplified point cloud recorded with a Velodyne VLP-16 laser scanner. The color coding of semantic labels corresponds to the Mapillary dataset (Neuhold et al. 2017).

We computed the IoU for different configurations on the NYUv2 dataset, including single-frame predictions and SemanticFusion. Label propagation was performed using our approach to retrain the predictor and executed for three iterations. We denote in Tab. 6.1 the use of the retrained semantic segmentation as *with LP*. Tab. 6.1 shows that our method outperforms single-frame as well as SemanticFusion. Using label propagation further improves the IoU. A visual comparison is provided in Fig. 6.4 for four different scenes. Already SemanticFusion improves single-frame predictions e.g. on the TV (yellow, first row), the window (blue, third row) and wall (gray, last row), but the result is noisy and partially inconsistent. We attribute this mostly to surfels of different scales that are not correctly fused. In comparison, our mesh is more consistent, for example on the bath tub (second row), but smoother around the edges. Yet, the bed (third row) is still mostly classified as a sofa. Through our label propagation and subsequent retraining, we were able to correct the classification. In the last row, we show a failure case in which the Label Propagation decreases the accuracy as the table (green) gets segmented as furniture. This decrease in accuracy is due to the fact that most single-frame predictions are wrongly labelling

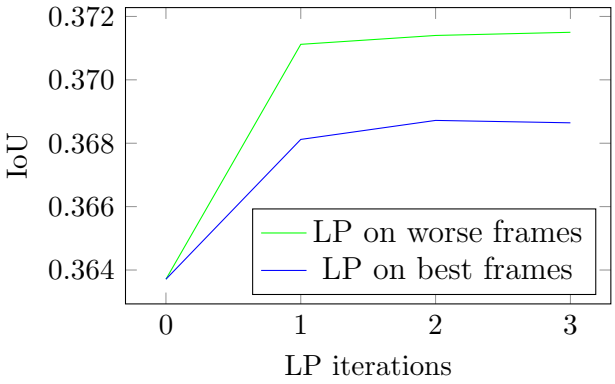


Figure 6.3: **LP variants**: We evaluate the IoU increase by performing LP on both the worse frames and the best frames. Propagating the labels towards the worse frames yields a higher IoU. Both LP variants converge quickly after the first iteration.

the object and establishing consistency through LP reinforces this wrong labelling. Further complete reconstructed scenes from NYUv2 are shown in Fig. 6.1.

Method	Bed	Bookshelf	Ceiling	Chair	Floor	Furniture	Objects	Picture	Sofa	Table	Tv	Wall	Window	Avg. IoU
Single frame	0.46	0.17	0.13	0.25	0.68	0.34	0.28	0.33	0.22	0.15	0.12	0.51	0.29	0.302
Single frame LP	0.52	0.20	0.14	0.27	0.68	0.35	0.30	0.35	0.26	0.14	0.14	0.53	0.31	0.322
SemanticFusion	0.47	0.15	0.18	0.30	0.65	0.36	0.30	0.35	0.24	0.15	0.20	0.53	0.33	0.324
SF with LP	0.52	0.18	0.21	0.31	0.65	0.38	0.31	0.38	0.28	0.16	0.20	0.54	0.36	0.343
Ours	0.54	0.17	0.23	0.35	**0.71**	0.40	0.33	0.39	0.28	**0.18**	0.21	0.56	0.37	0.363
Ours with LP	**0.56**	**0.20**	**0.25**	**0.35**	0.68	**0.40**	**0.34**	**0.41**	**0.31**	0.17	**0.23**	**0.57**	**0.38**	**0.372**

Table 6.1: **NYUv2 results**: We compare our method against single-frame predictions and SemanticFusion (McCormac et al. 2017). All cases are evaluated with and without Label Propagation (LP). For the case of single-frame, we exclude pixel without a valid depth measurement. All evaluations were performed at a 320×240 resolution.

We also conduct accuracy experiments on the courtyard dataset for which we densely labelled 48 frames around the scene. For fairness we labelled sky as background due to missing representation within the mesh. Fig. 6.5 shows that retraining using label propagation greatly improves the accuracy for most classes. However, an interesting observation from this experiment is that the single-frame predictions have on average higher accuracy than the fused semantics from the mesh. This is due to the fact that both the camera poses and the mesh are imperfect, hence fusing the information from various points of view may lead to discrepancies. This limitation is further reinforced by the fact that the classes which experience a higher drop in accuracy from the fusing process are those which are spatially small (lane-markings, poles, and street lights), while broader classes like building and vegetation remain largely unaffected by errors from the scene reconstruction.

For this reason we conduct further experiments to evaluate the impact of misalignments in the following Sec. 6.4. Nevertheless, we can conclude that label propagation grants a net improvement in the semantic accuracy, increasing the mean IoU for single-frame prediction by 7 % and for the fused information by 3 %. Fig. 6.6 shows a visual comparison of the semantics using various view points from the courtyard. The copters landing gear and rotor arm visible within camera images are masked out prior to evaluation. A partial failure case is shown in the first row. The thin lane-markings are actually degraded through fusion and LP.

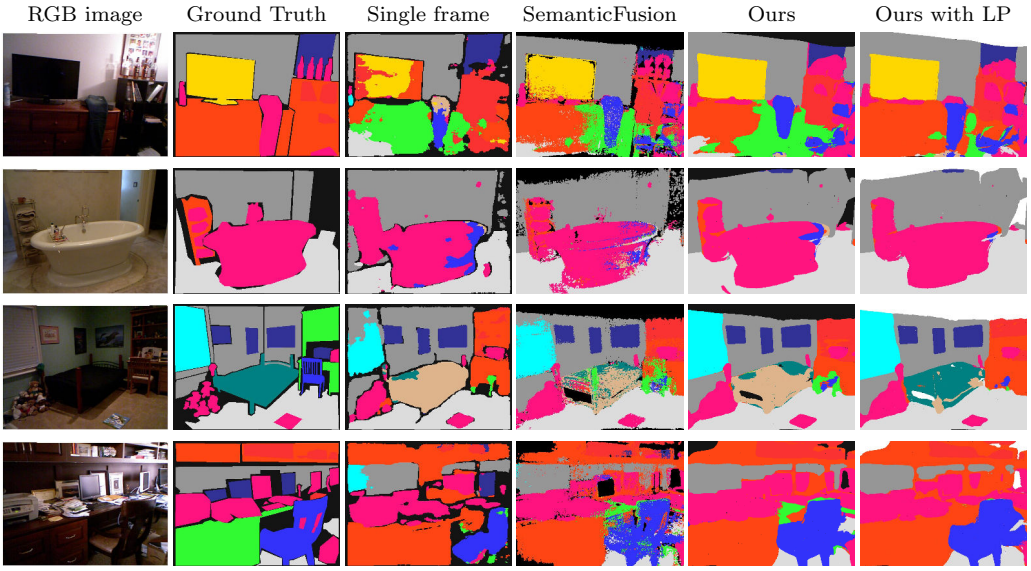


Figure 6.4: **NYUv2 qualitative results**: We compare our method, with and without Label Propagation, against single-frame predictions and SemanticFusion. The first three rows show a clear improvement achieved through Label Propagation, as the predictor learns to segment the table, bathtub and bed more accurately. The last row shows a failure case in which the Label Propagation decreases the accuracy as the table represented in green gets segmented as furniture. This decrease in accuracy is due to the fact that most single-frame predictions are wrongly labelling the object and establishing consistency through LP reinforces this wrong labelling.

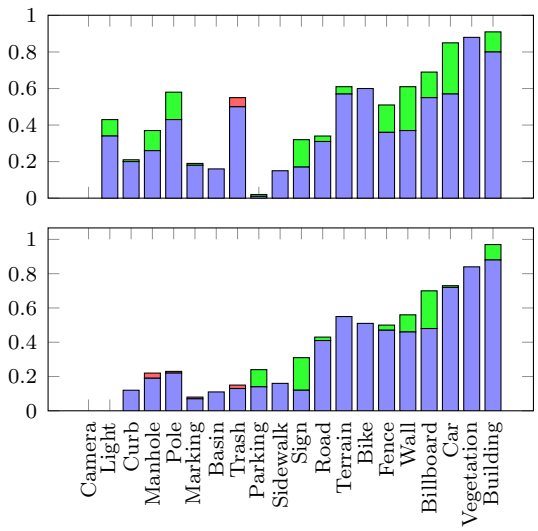


Figure 6.5: **Courtyard results**: We compare our method (bottom) against single-frame predictions (top). The per class IoU is denoted by blue bars. An increase in IoU due to Label Propagation is marked in green, a decrease in red. For single-frame we mask out the landing gear of the MAV and the areas which are not covered by the mesh.

We trace this back to inaccurate manual extrinsic and temporal calibration, since the corresponding single-frames continuously contained the lane-markings. Still, we observe improvements after LP on the container next to the service station. The second view was captured from behind the same service station. Insufficiency in the meshing process created only the top of the pole (front, left), which is correctly classified, but not connected to the ground plane reducing the overall IoU compared towards single-frame predictions. Further improvements through LP are especially visible in the last two rows. The left window is classified as a sign prior to retraining and a large portion of the sidewalk was incorrect. Also the building in the background is improved. The rooftop (third row) presents a unique novel view that is largely misclassified in the single-frame. Our mesh-based fusion improves the result as expected and allows successful retraining.

6.4 Registration Robustness

Mapping with known poses always raises the question of how robust the system is regarding misalignment. For this we perform experiments on the synthetic Synthia dataset (Ros et al. 2016) which provides ground truth poses, depth and semantics. We add random noise to the poses in order to observe the effect on the accuracy of the semantic map. We chose the *seq_4_summer* scene, due to the low number of dynamic objects. We aggregated the depth images and meshed the resulting point cloud (see Fig. 6.8). Synthia provides images from eight cameras arranged in groups of four to create an omnidirectional view-cone. For simplicity, we chose for the reconstruction only the front-facing camera of the left group. In order

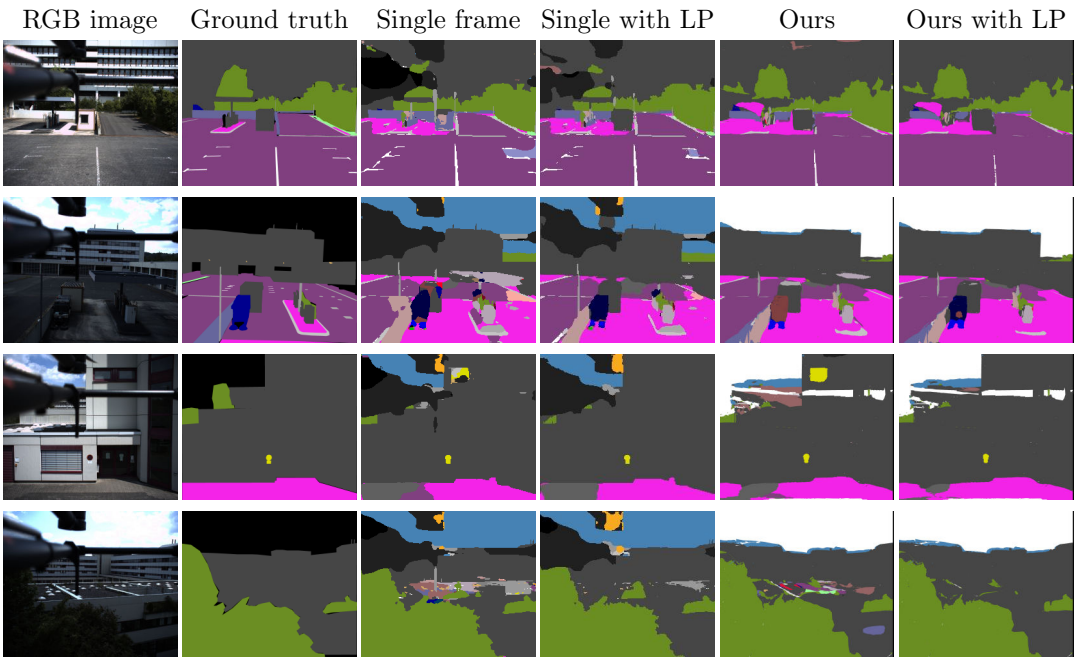


Figure 6.6: **Courtyard qualitative results**: We compare our method against single-frame with and without the label propagation.

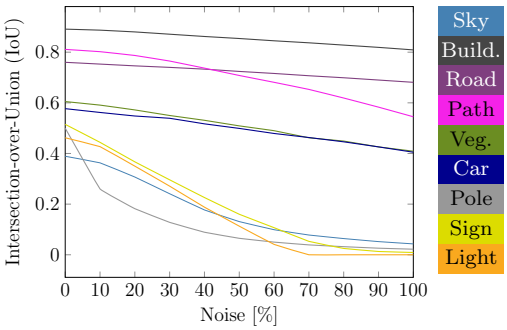


Figure 6.7: **Registration robustness:** Incorrect sensor poses for semantic map creation affects the accuracy as measured by IoU of larger object classes like buildings less than light posts or signs. Increasing amounts of noise are applied on the translation ($\leq 0.5\,\mathrm{m}$) and rotation ($\leq 5°$) of the sensor poses.

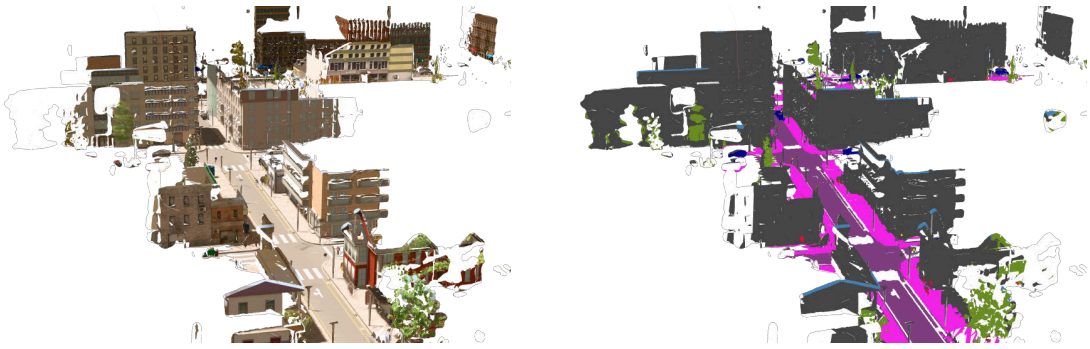


Figure 6.8: **Synthia semantic map:** We reconstruct a semantic map from a subset of frames from the Synthia dataset (Ros et al. 2016). We use the ground truth data and apply noise to evaluate the impact of camera misalignment on the semantic map accuracy.

to analyze the behaviour of the semantic map under incorrect poses or incorrect calibration between depth and color camera, the IoU is calculated for increasing amounts of noise on the translational ($\leq 0.5\,\mathrm{m}$) as well as rotational ($\leq 5°$) part of the camera poses. The IoU with increasing portion of noise is visualized per class in Fig. 6.7 with invisible or dynamic classes being disregarded. We choose to retain the cars as most of them were parked, and hence do not pose a problem for the reconstruction. As expected, the IoU decreases faster for smaller object classes, like poles, lights, and signs, than for buildings or the road. In conclusion, as the robotics community moves towards larger and bigger datasets with more semantic classes, the detail of the semantic maps will heavily depend on correct sensor poses.

6.5 Runtime Performance

We evaluate the runtime performance of our meshing and texturing modules separately, as they are performed sequentially with no overlap. Fig. 6.9 shows the resulting meshes after Poisson reconstruction for our simplified cloud, and the naïvely aggregated full point set recorded by the Velodyne scanner. It can be observed that the reconstruction quality does not suffer while the runtime and memory consumption is significantly lower (see Tab. 6.2).

The runtime of the semantic integration on the courtyard dataset is summarized in Fig. 6.10. We achieve real-time performance with an average texturing time of 27.1 ms per frame using an 8K texture. Decommitting the sparse volume is, however, a demanding functionality, and causes the average time per frame to

Cloud	#Points	#Verts	Time(s)	Mem(GB)
full	103M	4.5M	521.9	2.82
simple	21M	3.3M	193.8	1.99

Table 6.2: Poisson reconstruction using the naïvely aggregated cloud and our edge-aware simplified cloud. We report the number of points of the input cloud, the number of vertices of the reconstructed mesh, and the time and peak memory used by the reconstruction process.

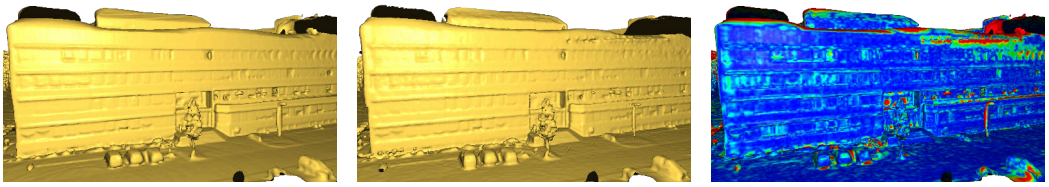


Figure 6.9: **Poisson reconstruction comparison:** Reconstruction from the edge-aware simplification (left), and its difference (right) toward the full reconstruction. The colormap denotes the deviation between the two meshes where red equals a difference of 15 cm and blue shows no difference. The deviation is minimal in areas of interest while reconstruction after simplification is faster (see Tab. 6.2).

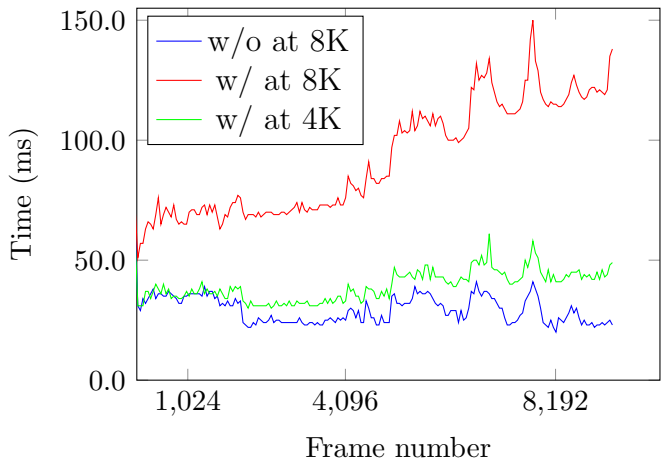


Figure 6.10: **Timing results for the courtyard dataset:** Semantic integration using a texture resolution of 8K without decommitting (blue line) can be performed in real-time. Enabling the decommiting at the same texture resolution (red line) proves to be too slow for real-time usage. Lowering the texture resolution to 4K allows the semantic integration with decommitment (green line) to be performed at real-time speeds.

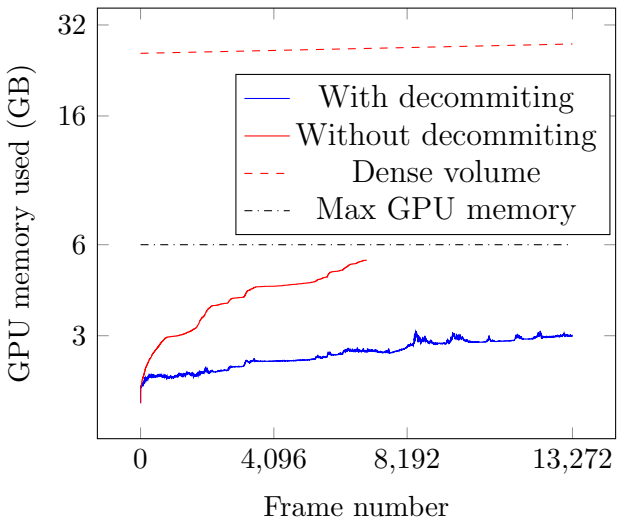


Figure 6.11: **GPU memory usage for a 10K texture on the courtyard dataset:** Dense allocation (red dashed line) would occupy more than 25.7 GB. Sparse allocation (red line) without decommitting quickly overburdens the available 6 GB causing a system failure. The memory usage drops with decommitting (blue line) below 3.1 GB at all times enabling the full reconstruction of the semantic map.

increase to 90.1 ms. Nevertheless, the semantic integration achieves 38.6 ms per frame for a smaller texture resolution of 4K.

6.6 Memory Consumption

Memory usage of the texturing system is also evaluated with and without decommitting of pages of the sparse texture. The results for the courtyard dataset are summarized in Fig. 6.11. We also analyse the relation between the decommitting threshold (the probability below which the pages in the sparse texture are deallocated) and the Intersection-over-Union (IoU) in Fig. 6.12. We evaluate this measure on the NYU dataset due to the presence of more labelled images than in the courtyard dataset which allows for a more accurate evaluation. We observe that while using low values for the threshold greatly reduces memory usage, higher values cause the IoU to degrade as more valuable information from the semantic texture is disregarded. However the decrease is still minor ($\leq 0.6\,\%$), as we restrain from decommitting pages that contain the argmax class. For further experiments, we choose a threshold value of 0.1.

6.7 Texture Resolution and Semantic Accuracy

We evaluate the impact of the semantic texture resolution and the accuracy of the semantic map. We perform the evaluation on the courtyard dataset as it spans a larger area than the NYU dataset and therefore the impact of the texture size becomes more noticeable. Semantic texture integration is performed for a series

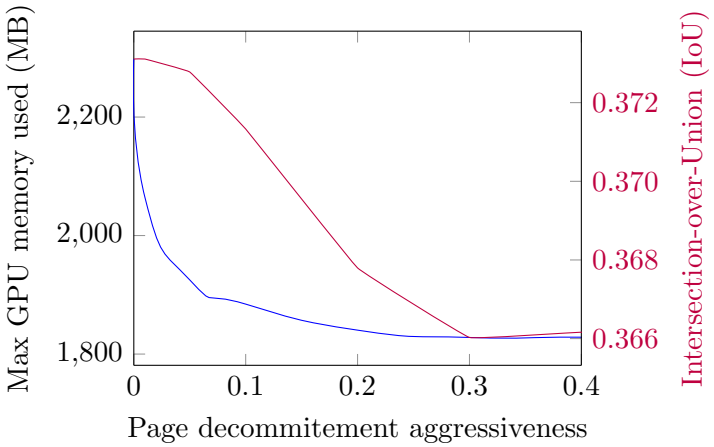


Figure 6.12: **Maximum GPU memory usage for different decommitting thresholds on NYUv2:** Increasing the decommitting threshold quickly reduces the memory consumption (blue line), while the IoU decreases slowly. As a consequence, we typically fix the threshold to 0.1.

of texture resolutions ranging from 512 to 12 288 and IoU is evaluated for each one. We observe that the IoU steadily increased and becomes stable at around a resolution of 4k.

However, there is some variation in the mean IoU as we increase the resolution, so we decide to further investigate this behaviour by analysing each class independently. For easier visualization we plot the IoU normalized between its minimum and maximum value: $N_{IoU} = (IoU - min)/(max - min)$. We observe that while most classes behave as expected; their accuracy increasing as we increase the texture resolution and eventually stabilizing, we observe a series of classes which behave more erratically. These "erratic classes" have their accuracies increasing and decreasing sporadically as the texture resolution is increased. The behaviour is depicted in figure Fig. 6.14 where the "stable" classes are depicted on the left and the erratic ones on the right. Note that the IoU accuracy shown is normalized individually for each class and therefore the relative differences between the different colored plots should not be taken into account.

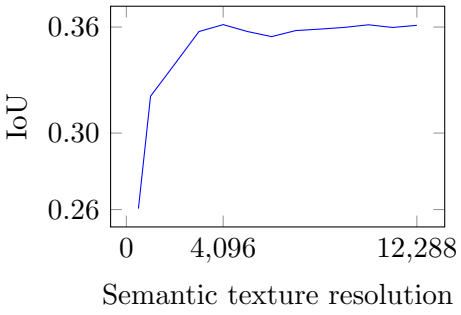
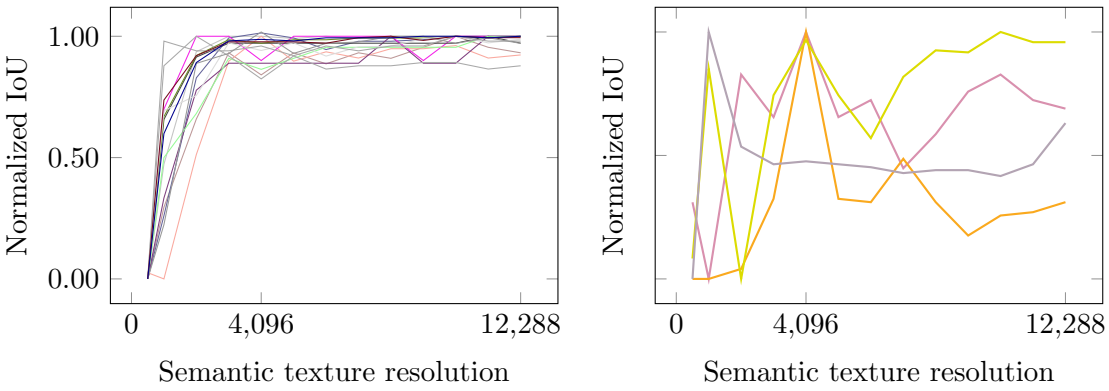


Figure 6.13: **Texture resolution and IoU**: We evaluate the mean IoU on the courtyard dataset as we increase the resolution of the semantic texture. The accuracy quickly rises and converges at a resolution of around 4K.

We observe that the classes that behave erratically correspond to very small objects, particularly the classes of street-light, lane-marking, traffic-sign and trash. This leads us to conclude that their sporadic behaviour is due to discretization artifacts when texture resolution is too low which makes their segmentation susceptible to aliasing effects. Due to this, some low resolution textures may actually



a) Stable classes a) Erratic classes

Figure 6.14: **Texture resolution and normalized IoU**: We plot the normalized IoU for each class and observe that some of them experience an accuracy increase as the resolution increases (left) while others behave more "erratically" fluctuating in accuracy due to aliasing issues (right). The erratically behaving classes correspond to small objects like lane-marking, street lights, traffic signs and trash bins.

have a higher measured accuracy as the texels that cover a bigger area may fit the small object completely. On the other side, with increased texel resolution, the inaccuracies due to imperfect poses become more apparent and the IoU for the small classes decreases drastically.

7 Conclusion

This thesis presents an end-to-end pipeline for the creation of semantic maps from depth and color images. Contributions were made in the areas of fast meshing from structured point clouds, in using semantic information to create a lightweight map which is decoupled from the underlying coarse geometry and finally, in a label propagation scheme which ensures temporal and spatial consistency and allows the predictor to learn from novel view points.

The whole system was demonstrated on the NYUv2 dataset and compared with the state of the art SemanticFusion(McCormac et al. 2017). Our mesh-based semantic mapping achieves higher accuracy than SemanticFusion and our Label Propagation scheme boosts the accuracy even further.

A possible extension to the work would be to incorporate a more scale aware fusion. At the moment the semantic labels are associated projectively with the texture on the mesh which disregards the fact that some objects are smaller than others. This causes issues for smaller semantic objects which can get blurred out by incorrect camera poses. Taking into account the scale of them may alleviate this issue.

Another interesting extension would be to investigate in a join map creation in which geometry and semantics are inferred simultaneously and both iteratively improve upon the estimate of the other. This would require however a change in the underlying data structure as meshes, while efficient in representation, are not suitable for large topology changes.

List of Figures

List of Tables

Bibliography

Alliez, Pierre; Saboret, Laurent, and Guennebaud, Gaël (2018). "Poisson surface reconstruction". In: *CGAL User and Reference Manual*. 4.12. CGAL Editorial Board. URL: https://doc.cgal.org/4.12/Manual/packages.html#PkgPoissonSurfaceReconstructionSummary.

Bank, Randolph E. (1998). *PLTMG, a Software Package for Solving Elliptic Partial Differential Equations: Users' Guide 8.0*. Vol. 5. Siam.

Chen, Jiawen; Bautembach, Dennis, and Izadi, Shahram (2013). "Scalable real-time volumetric surface reconstruction". In: *ACM Transactions on Graphics* 32.4, p. 113.

Chen, Liang-Chieh; Papandreou, George; Kokkinos, Iasonas; Murphy, Kevin, and Yuille, Alan L. (2018). "Deeplab: semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs". In: *IEEE transactions on pattern analysis and machine intelligence* 40.4, pp. 834–848.

Civera, Javier; Gálvez-López, Dorian; Riazuelo, Luis; Tardós, Juan D., and Montiel, Martínez (2011). "Towards semantic SLAM using a monocular camera". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 1277–1284.

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

Douglas, David H. and Peucker, Thomas K. (1973). "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature". In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 10.2, pp. 112–122.

Droeschel, David and Behnke, Sven (2018). "Efficient continuous-time SLAM for 3D lidar-based online mapping". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Eigen, David and Fergus, Rob (2015). "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 2650–2658.

Engel, Jakob; Schöps, Thomas, and Cremers, Daniel (2014). "LSD-SLAM: Large-scale direct monocular SLAM". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, pp. 834–849.

Garland, Michael and Heckbert, Paul S. (1998). "Simplifying surfaces with color and texture using quadric error metrics". In: *Visualization proceedings*. IEEE, pp. 263–269.

Goldman, Daniel B. and Chen, Jiun-Hung (2005). "Vignette and exposure calibration and compensation". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Guo, Chuan; Pleiss, Geoff; Sun, Yu, and Weinberger, Kilian Q. (2017). "On calibration of modern neural networks". In: *Arxiv preprint arxiv:1706.04599*.

He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing, and Sun, Jian (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.

Hermans, Alexander; Floros, Georgios, and Leibe, Bastian (2014). "Dense 3d semantic mapping of indoor scenes from rgb-d images". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 2631–2638.

Holz, Dirk and Behnke, Sven (2014a). "Approximate triangulation and region growing for efficient segmentation and smoothing of range images". In: *Robotics and Autonomous Systems* 62.9, pp. 1282–1293. ISSN: 0921-8890.

— (2014b). "Registration of non-uniform density 3D point clouds using approximate surface reconstruction". In: *ISR/Robotik 2014; 41st International Symposium on Robotics; Proceedings*. VDE, pp. 1–7.

Hoppe, Hugues; DeRose, Tony; Duchamp, Tom; McDonald, John, and Stuetzle, Werner (1992). *Surface reconstruction from unorganized points*. Vol. 26. 2. ACM.

Hornung, Armin; Wurm, Kai M; Bennewitz, Maren; Stachniss, Cyrill, and Burgard, Wolfram (2013). "Octomap: an efficient probabilistic 3D mapping framework based on octrees". In: *Autonomous robots* 34.3, pp. 189–206.

Jin, Shuangshuang; Lewis, Robert R., and West, David (2005). "A comparison of algorithms for vertex normal computation". In: *The visual computer* 21.1-2, pp. 71–82.

Kazhdan, Michael; Bolitho, Matthew, and Hoppe, Hugues (2006). "Poisson surface reconstruction". In: *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*. SGP '06. Cagliari, Sardinia, Italy: Eurographics Association, pp. 61–70. ISBN: 3-905673-36-3. URL: http://dl.acm.org/citation.cfm?id=1281957.1281965.

Kazhdan, Michael and Hoppe, Hugues (2013). "Screened poisson surface reconstruction". In: *ACM Transactions on Graphics* 32.3, p. 29.

Keller, Maik; Lefloch, Damien; Lambers, Martin; Izadi, Shahram; Weyrich, Tim, and Kolb, Andreas (2013). "Real-time 3D reconstruction in dynamic scenes using point-based fusion". In: *3D Vision-3DV 2013, 2013 International Conference*. IEEE, pp. 1–8.

Kostavelis, Ioannis and Gasteratos, Antonios (2015). "Semantic mapping for mobile robotics tasks: A survey". In: *Journal of Robotics and Autonomous Systems* 66, pp. 86–103.

Kundu, Abhijit; Li, Yin; Dellaert, Frank; Li, Fuxin, and Rehg, James M (2014). "Joint semantic segmentation and 3D reconstruction from monocular video". In: *European Conference on Computer Vision*. Springer, pp. 703–718.

Li, Xuanpeng and Belaroussi, Rachid (2016). "Semi-dense 3D semantic mapping from monocular SLAM". In: *Arxiv preprint arxiv:1611.04144*.

Lin, Guosheng; Milan, Anton; Shen, Chunhua, and Reid, Ian (2017). "Refinenet: Multi-path refinement networks for high-resolution semantic segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ling, Yonggen and Shen, Shaojie (2017). "Building maps for autonomous navigation using sparse visual SLAM features". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 1374–1381.

Long, Jonathan; Shelhamer, Evan, and Darrell, Trevor (2015). "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440.

Maiti, Abhik and Chakravarty, Debashish (2016). "Performance analysis of different surface reconstruction algorithms for 3D reconstruction of outdoor objects from their digital images". In: *Springerplus* 5.1, p. 932. ISSN: 2193-1801. URL: https://doi.org/10.1186/s40064-016-2425-9.

Marton, Zoltan Csaba; Rusu, Radu Bogdan, and Beetz, Michael (2009). "On fast surface reconstruction methods for large and noisy point clouds". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3218–3223.

McCormac, John; Handa, Ankur; Davison, Andrew, and Leutenegger, Stefan (2017). "SemanticFusion: Dense 3D semantic mapping with convolutional neural networks". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 4628–4635.

Moravec, Hans and Elfes, Alberto (1985). "High resolution maps from wide angle sonar". In: *Robotics and Automation. Proceedings. 1985 IEEE International Conference*. Vol. 2. IEEE, pp. 116–121.

Neuhold, Gerhard; Ollmann, Tobias; Bulo, S. Rota, and Kontschieder, Peter (2017). "The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes." In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 5000–5009.

Newcombe, Richard A.; Fox, Dieter, and Seitz, Steven M. (2015). "Dynamicfusion: reconstruction and tracking of non-rigid scenes in real-time". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 343–352.

Newcombe, Richard A; Izadi, Shahram; Hilliges, Otmar; Molyneaux, David; Kim, David; Davison, Andrew J.; Kohi, Pushmeet; Shotton, Jamie; Hodges, Steve, and Fitzgibbon, Andrew (2011). "KinectFusion: Real-time dense surface mapping and tracking". In: *Mixed and augmented reality (ISMAR), 10th IEEE international symposium*. IEEE, pp. 127–136.

Nießner, Matthias; Zollhöfer, Michael; Izadi, Shahram, and Stamminger, Marc (2013). "Real-time 3D reconstruction at scale using voxel hashing". In: *ACM Transactions on Graphics* 32.6, p. 169.

Oleynikova, Helen; Taylor, Zachary; Fehr, Marius; Siegwart, Roland, and Nieto, Juan (2017). "Voxblox: incremental 3D euclidean signed distance fields for on-board mav planning". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 1366–1373.

Peasley, Brian (2013). *Large scale 3d mapping of indoor environments using a handheld RGB-D camera*.

Poranne, Roi; Tarini, Marco; Huber, Sandro; Panozzo, Daniele, and Sorkine-Hornung, Olga (2017). "Autocuts: simultaneous distortion and cut optimization for UV mapping". In: *ACM Transactions on Graphics* 36.6, p. 215.

Quigley, Morgan; Conley, Ken; Gerkey, Brian; Faust, Josh; Foote, Tully; Leibs, Jeremy; Wheeler, Rob, and Ng, Andrew Y (2009). "ROS: an open-source Robot Operating System". In: *ICRA Workshop on Open Source Software*.

Reddy, N. Dinesh; Singhal, Prateek, and Krishna, K. Madhava (2014). "Semantic motion segmentation using dense CRF formulation". In: *Proceedings of the 2014 Indian Conference on Computer Vision Graphics and Image Processing*. ACM, p. 56.

Romanoni, Andrea; Fiorenti, Daniele, and Matteucci, Matteo (2017). "Mesh-based 3D Textured Urban Mapping". In: *Arxiv preprint arxiv:1708.05543*.

Romanoni, Andrea and Matteucci, Matteo (2015). "Incremental reconstruction of urban environments by edge-points delaunay triangulation". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 4473–4479.

Ros, German; Sellart, Laura; Materzynska, Joanna; Vazquez, David, and Lopez, Antonio M. (2016). "The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3234–3243.

Shotton, Jamie; Fitzgibbon, Andrew; Cook, Mat; Sharp, Toby; Finocchio, Mark; Moore, Richard; Kipman, Alex, and Blake, Andrew (2011). "Real-time human pose recognition in parts from single depth images". In: *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference*. Ieee, pp. 1297–1304.

Shotton, Jamie; Johnson, Matthew, and Cipolla, Roberto (2008). "Semantic texton forests for image categorization and segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 1–8.

Silberman, Nathan; Hoiem, Derek; Kohli, Pushmeet, and Fergus, Rob (2012). "Indoor Segmentation and Support Inference from RGBD Images". In: *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 746–760.

Steinbrücker, Frank; Sturm, Jürgen, and Cremers, Daniel (2014). "Volumetric 3D mapping in real-time on a CPU". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 2021–2028.

Stückler, Jörg and Behnke, Sven (2012). "Integrating depth and color cues for dense multi-resolution scene mapping using RGB-D cameras". In: *Multisensor Fusion and Integration for Intelligent Systems (MFI), 2012 IEEE Conference*. IEEE, pp. 162–167.

Tateno, Keisuke; Tombari, Federico; Laina, Iro, and Navab, Nassir (2017). "CNN-SLAM: Real-time dense monocular SLAM with learned depth prediction". In: *Arxiv preprint arxiv:1704.03489*.

Thürrner, Grit and Wüthrich, Charles A (1998). "Computing vertex normals from polygonal facets". In: *Journal of Graphics Tools* 3.1, pp. 43–46.

Valentin, Julien; Sengupta, Sunando; Warrell, Jonathan; Shahrokni, Ali, and Torr, Philip (2013). "Mesh based semantic modelling for indoor and outdoor scenes". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 2067–2074.

Vineet, Vibhav; Miksik, Ondrej; Lidegaard, Morten; Nießner, Matthias; Golodetz, Stuart; Prisacariu, Victor A.; Kähler, Olaf; Murray, David W; Izadi, Shahram; Pérez, Patrick, et al. (2015). "Incremental dense semantic stereo fusion for large-scale semantic scene reconstruction". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 75–82.

Whelan, Thomas; Kaess, Michael; Fallon, Maurice; Johannsson, Hordur; Leonard, John, and McDonald, John (2012). "Kintinuous: Spatially extended Kinectfusion". In:

Whelan, Thomas; Leutenegger, Stefan; Salas-Moreno, R.; Glocker, Ben, and Davison, Andrew (2015). "ElasticFusion: Dense SLAM without a pose graph". In: Robotics: Science and Systems.

Xie, Saining; Girshick, Ross; Dollár, Piotr; Tu, Zhuowen, and He, Kaiming (2017). "Aggregated residual transformations for deep neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 5987–5995.

Zollhöfer, Michael; Stotko, Patrick; Görlitz, Andreas; Theobalt, Christian; Nießner, Matthias; Klein, Reinhard, and Kolb, Andreas (2018). "State of the Art on 3D Reconstruction with RGB-D Cameras". In: *Computer graphics forum*. Vol. 37. Wiley Online Library, pp. 625–652.