Algorithms for the Reconstruction, Analysis, Repairing and Enhancement of 3D Urban Models from Multiple Data Sources

Doctoral Thesis

Marc Comino Trinidad

Doctoral dissertation submitted for
the International Doctorate Mention

Universitat Politècnica de Catalunya

Department of Computer Science (CS)
PhD in Computing

Advisor: Carlos Andújar Gran
Co-Advisor: Antonio Chica Calaf

Barcelona, November 2020

Mighty is geometry; joined with art, resistless.

— EURIPIDES

All that a city will ever allow you is an angle on it – an oblique, indirect sample of what it contains, or what passes through it; a point of view.

— PETER CONRAD

Los dos elementos que el viajero capta en la gran ciudad son: arquitectura extrahumana y ritmo furioso. Geometría y angustia.

— FEDERICO GARCIA LORCA

Abstract

Over the last few years, there has been a notorious growth in the field of digitization of 3D buildings and urban environments. The substantial improvement of both scanning hardware and reconstruction algorithms has led to the development of representations of buildings and cities that can be remotely transmitted and inspected in real-time. Applications that benefit from these technologies include several GPS navigators and virtual globes such as Google Earth [3] or the tools provided by the Institut Cartogràfic i Geològic de Catalunya [4; 5; 6].

In particular, in this thesis, we conceptualize cities as a collection of individual buildings. Hence, we focus on the individual processing of one structure at a time, rather than on the larger-scale processing of urban environments.

Nowadays, there is a wide diversity of digitization technologies, and the choice of the appropriate one is key for each particular application. Roughly, these techniques can be grouped around three main families:

- Time-of-flight (terrestrial and aerial LiDAR)
- Photogrammetry (street-level, satellite, and aerial imagery)
- Human-edited vector data (cadastre and other map sources)

Each of these has its advantages in terms of covered area, data quality, economic cost, and processing effort.

Plane and car-mounted LiDAR devices are optimal for sweeping huge areas, but acquiring and calibrating such devices is not a trivial task. Moreover, the capturing process is done by scan lines, which need to be registered using GPS and inertial data. As an alternative, terrestrial LiDAR devices are more accessible but cover smaller areas, and their sampling strategy usually produces massive point clouds with over-represented plain regions. A more inexpensive option is street-level imagery. A dense set of images captured with a commodity camera can be fed to state-of-the-art multi-view stereo algorithms to produce realistic-enough reconstructions. Another advantage of this approach is capturing high-quality color data, whereas the geometric information is usually lacking.

In this thesis, we analyze in-depth some of the shortcomings of these data-acquisition methods and propose new ways to overcome them. Mainly, we focus on the technologies that allow high-quality digitization of individual buildings. These technologies include terrestrial LiDAR for geometric information and street-level imagery for color information.

Our main goal is the processing and completion of detailed 3D urban representations. For this, we will work with multiple data sources and combine them when possible to produce models that can be inspected in real-time. Our research has focused on the following contributions:

- Effective and feature-preserving simplification of massive point clouds.
- Developing normal estimation algorithms explicitly designed for LiDAR data.
- Low-stretch panoramic representation for point clouds.
- Semantic analysis of street-level imagery for improved multi-view stereo reconstruction.
- Color improvement through heuristic techniques and the registration of LiDAR and imagery data.
- Efficient and faithful visualization of massive point clouds using image-based techniques.

Acknowledgments

The document you are about to read is already an essential piece of my life. It represents an old teenager's growth into a young adult, a maturing process filled with joy and tears. It has been a slow but steady process that I believe has reached a successful end.

This journey would not have been possible without the help of so many beautiful people. I want to start expressing my deepest gratitude for Carlos and Toni, my advisors. Whenever I reached out for help, whenever I needed guidance, whenever I needed learning, they were always present. I want to make this extensive to Pere Brunet, who we never needed to ask for help, as he ever reached us before we could!

I also want to thank all the ViRVIG group members, starting with Isabel Navazo, who retired earlier this year. I cannot imagine another person managing the group with the amount of love and care that Isabel showed. I also want to thank professors Pere-Pau Vázquez, Àlvar Vinacua, Marta Fairén, Nuria Pelechano, and Antonio Susin because they have always treated me as an equal member of the group. Finally, I want to thank Óscar Argudo, my senior Ph.D. who has always extended a helping hand. I feel these people have become a part of a little family of mine.

Finally, I want to thank various people who made it possible for me to be here. Thanks, Maria Serna, program coordinator for the Ph.D. in Computing, and Merce Juan, administrator of the program, for tirelessly fighting the bureaucracy.

Besides my academic family, I also want to thank the people who have tried to talk me into dropping my Ph.D. Starting with my parents Juana and Antonio, my brother Oscar and my grandmother Lucia. You are the reason I have been able to get to this point. You are my happiness.

Special thanks to my friends, who I consider my non-blood-related family. Thanks, Gerard, for always sharing a coffee. Thanks, Noel, for always sharing a laugh. Thanks, Ferran, for keeping our friendship through the distance. Thanks, Marc, for sharing your name with me. Thanks, Albert, for inviting us to your home when we travel to Madrid. Thanks, Nestor, for forcing us to ski. Thanks, Jaume, for talking so that we can understand you. Thanks, Maria José, for being

unique. Thanks, Eli, for being a public officer. Thanks, Isaac, for sharing your guest room. Thanks, David and Carlos, for being there, sometimes. And, finally, thanks, Clara, for encouraging us to join you at Google.

My time at Google has been a milestone that profoundly influenced my career and broadened my view of the world. I want to thank Ania Marszalek, who was my first manager. Thanks for your trust. I want to thank Eric Penner, who is probably responsible for most I know about photogrammetry. And, especially, I want to thank Janne Kontkanen, my two times host, who has given me infinite learning opportunities. I also want to thank the wonderful people I got to work with, including Florian Kainz, Michael Shantzis, Ricardo Martin-Brualla, Robert Gens, Lior Shapira, Noah Snavely, and many more.

Finally, I want to thank Dr. Montse Mateu and Dr. Laia Pujol for showing me that, even if life is imperfect, it is still fine to be happy.

I want to thank the *Museu Diocesà i Comarcal de Solsona*, Carles Freixes, Lídia Fàbregas, the *Museu Nacional d'Art Catalunya* and Gemma Ylla-Catalá for kindly allowing us to scan Pedret's mural paintings. I would also like to thank the *Ajuntament de la Garriga* and Enric Costa for kindly allowing us to scan the *Doma* church and Javier Rui-Wamba, Esteyco's president, who commissioned us to digitize the *Mercat de Sant Antoni*. Finally, I want to thank Imanol Muñoz, Xavier Pueyo, and Jordi Moyés, for helping us with the digitization, and Isaac Besora, for being our coauthor.

This thesis has been funded by the Spanish Ministry of Education, Culture and Sports Ph.D. Grant FPU14/00725, by the Spanish Ministry of Economy and Competitiveness and FEDER Grants TIN2014-52211-C2-1-R and TIN2017-88515-C2-1-R, the *Romanesque Pyrenees, Space of Artistic Confluences II (PRECA II)* project (HAR2017-84451-P, Universitat de Barcelona) and the JPICH-0127 EU project *Enhancement of Heritage Experiences: the Middle Ages. Digital Layered Models of Architecture and Mural Paintings over Time (EHEM).*

Contents

1 Introduction

Over the last few years, there has been a notorious growth in the field of digitization of 3D buildings and urban environments. The substantial improvement of both scanning hardware and reconstruction algorithms has led to the development of representations of buildings and cities that can be remotely transmitted and inspected in real-time.

(1) Source: Google Earth [3]

(2) Source: Institut Cartogràfic i Geològic de Catalunya [5]

Figure 1.1: Two 3D views of the Sagrada Familia in Barcelona. **(1)** is a mesh-based model whereas **(2)** is a point-based one.

Applications that benefit from these technologies include several GPS navigators and virtual globes such as Google Earth [3] or the tools provided by the Institut Cartogràfic i Geològic de Catalunya [4; 5; 6]. These viewers have been designed to provide compelling aerial inspection results, as shown in Figure **1.1**. However, when zoomed to a finer scale, the detail they provide is insufficient (see

Figure 1.2) for several use cases, such as:

- Visual simulation from a near-surface point of view.
- Cultural heritage and virtual tourism.
- Tracking public works: keeping a 3D record of work progress on improving urban infrastructure.
- Physical phenomena simulation (e.g., weathering or water evacuation).

(1) Source: Google Earth [3]

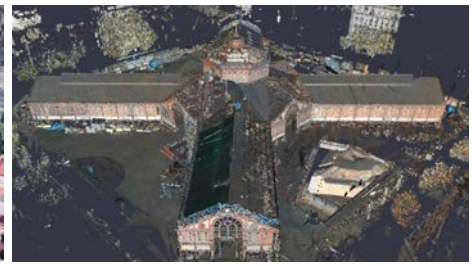
(2) Source: Google Earth [3]

(3) Source: Institut Cartogràfic i Geològic de Catalunya [5]

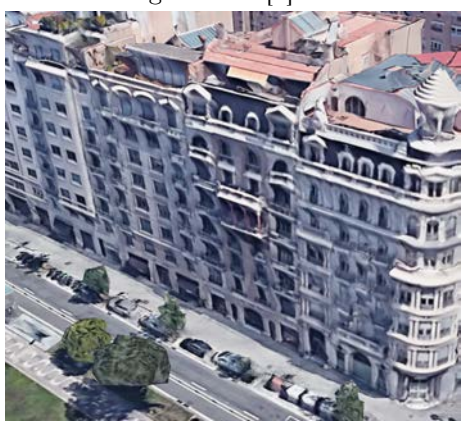
(4) Source: Institut Cartogràfic i Geològic de Catalunya [5]

Figure 1.2: Illustrating geometric problems. While these Sagrada Familia models are good enough for a bird's-eye view, they lack detail for closer inspection. (1) and (2) depict example artifacts for low-resolution mesh models. (3) and (4) show the same point cloud rendered with different point sizes. In (4) we can perceive the actual density, which is not enough to represent finer detail. These models were generated from airborne data (either LiDAR or multi-view stereo on aerial imagery, taken from an aircraft). These technologies usually yield low-resolution representations only suitable for coarse inspection.

(1) Mercat de Sant Antoni (Barcelona). Source: Google Earth [3]



(2) Mercat de Sant Antoni (Barcelona). Source: LiDAR scan.



(3) A facade in Barcelona. Source: Google Earth [3]

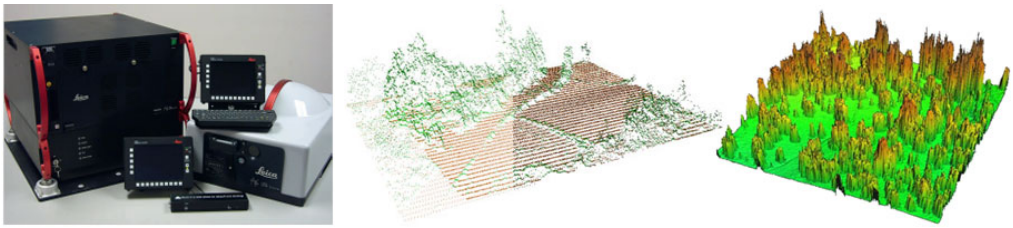
(4) A facade in Barcelona. Source: Multi-view Stereo Reconstruction.

(5) La Doma church (La Garriga). Source: Google Earth [3]

(6) La Doma church (La Garriga). Source: LiDAR scan.

Figure 1.3: Comparison of some Google Earth models against some of the models obtained during this thesis.

Particularly, in this thesis, we conceptualize cities as a collection of individual buildings. Hence, we focus on the separated processing of one structure at a time, rather than on the larger-scale processing of urban environments. In Figure **1.3**, we can see some of the high-detail models obtained during this thesis compared against the same ones in Google Earth [3].



(1) Leica ALS50-II Sensor Source: Institut Cartogràfic i Geològic de Catalunya [7]

(2) Sample point cloud obtained from aerial LiDAR Source: Institut Cartogràfic i Geològic de Catalunya [7]

Figure 1.4: Aerial LiDAR sensor and aerial LiDAR point cloud.

Nowadays, there is a wide diversity of digitization technologies, and the choice of the appropriate one is essential for each particular application. Roughly, these techniques can be grouped around three main families:

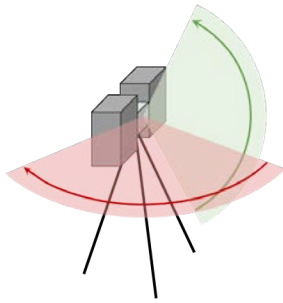
- Time-of-flight (terrestrial and aerial LiDAR)
- Photogrammetry (cameras and street-level, satellite, and aerial imagery)
- Human-edited vector data (cadastre and other map sources)

Each of these has its advantages in terms of covered area, data quality, economic cost, and processing effort.

Plane and car-mounted LiDAR devices are optimal for sweeping huge areas, but acquiring and calibrating such devices is not a trivial task. Namely, aside from the LiDAR sensor (Figure **1.41**), you need access to an aircraft or an automobile to mount it. Moreover, the capturing process is done by scan lines, which need to be registered using GPS and inertial data. This approach is suitable to digitize a vast region, such as a city, at low resolution, which may be useful to generate bird's-eye views while lacking near-surface detail (Figure **1.42**). For instance, the Institut Cartogràfic i Geològic de Catalunya [4; 7] has LiDAR data at 0.5 points/m^2 for the whole Catalonia territory.

As an alternative, terrestrial LiDAR devices (Figures **1.51** and **1.53**) are more accessible. However, they cover smaller areas, and their sampling strategy

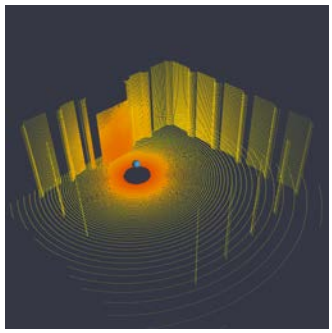
(1) Leica P20
Source: Leica Geosystems [9]



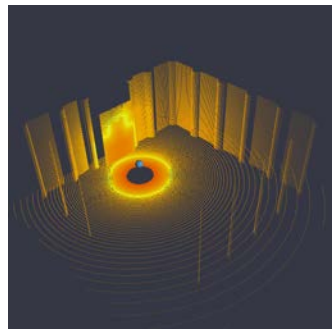
(2) Illustrating the capturing process.

(3) Leica RTC 360
Source: Leica Geosystems [9]

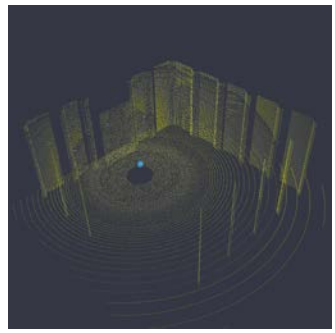
Figure 1.5: Terrestrial LiDAR scanning devices. These are able to capture very high-resolution point clouds with very irregular sample densities. The rotating mirror and base allow for hemispherical scanning.



(1) Raw simulated LiDAR point cloud.



(2) Raw simulated LiDAR point cloud.

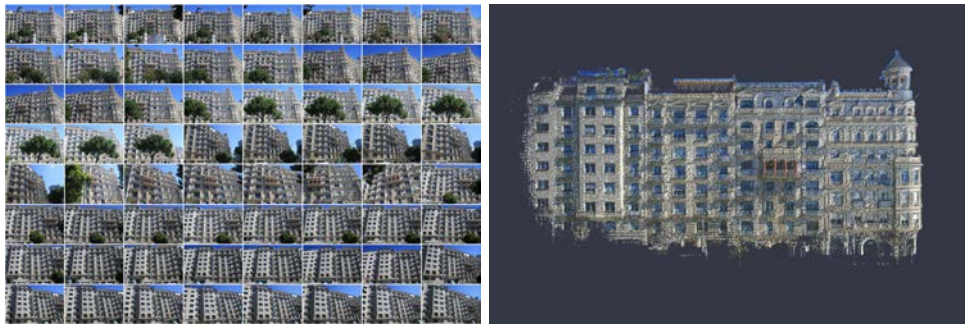


(3) Simplification of Figures **1.61** and **1.62**.

Figure 1.6: **(1)** and **(2)** are the same cloud whereas **(3)** is a simplification of these. These heat maps display how the point densities change across the clouds (in a logarithmic scale). Colors are obtained through linear interpolation between 0 (encoded in *white*), $10e3$ (encoded in yellow) and $10e6$ pts/m^3 (encoded in *red*).
In **(1)** we can see the actual density values and how they change across the cloud. There are huge differences in densities resulting from the scanning process. Surfaces which are orthogonal to the ray directions are over-sampled (red and orange regions) as opposed to planes which are very tangent to the ray directions.
In **(2)** and **(3)** we display the distances to the mean density value of the cloud. This is a better metric for comparing the original cloud against its simplification. If we directly displayed the density values, we would see that they are smaller for the simplified one, but this would give us less information about how dissimilar they are. This way, in **(3)** we can see that the density values are more similar across the cloud and this is because redundant samples have been removed while preserving most of the points in other regions.

usually produces massive point clouds where some plain regions may be over-represented (Figure 1.6). These devices are mounted on a rotating support, allowing horizontal sweeping while using a rotating mirror to deflect the infrared beam allows for vertical sweeping (Figure 1.52). The distance to a given surface is estimated by measuring the time it takes for the beam to return to the sensor after bouncing on it (time-of-flight). The produced samples are concentric (i.e., each sample has unique polar coordinates), hence to completely capture a complex scene, these devices must perform scans at multiple locations.

Similar to aerial LiDAR, to obtain aerial imagery, one would require a flying vehicle. Moreover, the resolution of the resulting models is directly related to the resolution of the captured image, which, at building level, is not very high. Its main advantage is that cameras are usually cheaper than a LiDAR device.



(1) 64 images of a facade in Barcelona. **(2)** Point Cloud generated using Colmap [SF16; SZP+16].

Figure 1.7: Multi-view stereo reconstruction example on a dense set of 64 images. The original image resolution was 3888×2592 pixels although for the reconstruction they were downscaled to approximately 3000×2000 pixels. The resulting reconstruction had about 10 million points which pales in comparison to terrestrial LiDAR scans. Notice the lack of points on windows (specular surfaces).

A more inexpensive option is street-level imagery. A dense set of images captured with a commodity camera can be fed to state-of-the-art multi-view stereo algorithms to produce realistic-enough reconstructions (Figure 1.7). Another advantage of this approach is capturing high-quality color data, whereas the geometric information usually lacks detail and robustness. This process usually consists of two main steps: structure-from-motion and dense stereo reconstruction. For the first one, a set of sparse features are computed for each input image. These are used to calibrate the camera intrinsics (e.g., focal length, principal point) and the extrinsics (relative positions of the images). The second one usually consists of a plane-sweeping algorithm; for a given image, we start by reprojecting others to its point of view at different depths. The depth for one pixel

is determined as the reprojection depth that minimizes a given energy function (usually, color similarity within a neighborhood). Because of this, these algorithms usually fail to estimate any depth values in both textureless and specular surfaces.

In this thesis, we analyze some of the shortcomings of these data-acquisition methods and propose new ways to overcome them. Mainly, we focus on the technologies that allow high-quality digitization of individual buildings. These are terrestrial LiDAR for geometric information and street-level imagery for color information.

1.1 CONTRIBUTIONS

Our main goal is the processing and completion of detailed 3D urban representations. For this, we will work with multiple data sources and combine them when possible to produce models that can be inspected in real-time. In particular, we focus on the processing of individual buildings. We propose new methods to convert raw inputs into practical representations. We test them on synthetic datasets for quantitative valuation and on our datasets and other public ones for qualitative evaluation.

Here we summarize our contributions. Figure 1.8 provides a visual overview of these and the relations between them.

1 **Effective Simplification of Point Clouds.**

LiDAR point clouds captured using terrestrial devices are usually massive and very unevenly distributed, i.e., over-sampled and redundant in some areas and under-sampled in others (see Figure 1.6). More light-weight representations are needed for rendering, streaming, or learning on these point clouds. Hence smart techniques for point decimation are needed. Ideally, we would need an algorithm that removes points from over-sampled regions while preserving those in regions with smaller densities. More formally, we want to obtain a Poisson disk sample with maximal radius by increasingly removing points while maximizing the minimum radius between samples.

We propose a greedy out-of-core method to approach this problem. Given a point cloud and a target number of points, our algorithm iteratively removes points based on specific cost criteria. The need to work out-of-core arises from the fact that most of these point clouds do not fit in memory. Hence individual parts (in our case, voxels) must be processed independently. Nevertheless, we can ensure that locally optimal results aggregate

to a nearly-optimal global result. We test our algorithm on smaller synthetic clouds and real huge ones, and we prove we can produce compelling results. This contribution is developed mainly throughout Section 4.1.

2 Sensor-aware Normal Estimation.

Normal vectors are essential information for multiple applications related to point clouds, such as rendering, surface reconstruction, 3D printing, segmentation, or simplification. Robustly estimating these normals is a difficult task that has been approached by multiple authors. However, for the particular case of LiDAR data, it is even more challenging due to the presence of noise, outliers, under-sampling, and also over-sampling (depending on the region). Moreover, noise is not always spatially smooth for this particular case due to points scanned from different locations. This fact led us to design a novel algorithm that first computes multiple normal candidates (one for each scan location from which the query point has neighbors) and then combines them into a single normal estimate.

Our normal-estimation algorithm for LiDAR data is based on the traditional PCA approximation. However, whereas this approach uses a fixed user-provided radius, we instead derive mathematical bounds for this neighborhood based on each point's estimated noise level. This results in a virtually parameter-free method. We work under the assumption that the noise level is constant within the query point neighborhood, and this only holds if we only consider points captured from the same scan location. Hence we compute a different normal for each location and then combine them using different strategies. We test our method on synthetic data for quantitative evaluation and real data for qualitative evaluation, and we show that we achieve competitive results compared with previous methods. This contribution is developed mainly throughout Section 4.2, and related results were published in [CAC+17; CAC+18].

3 Low-stretch Panoramic Representation.

Finding the right representation for data is a challenging and application-dependent task. Particularly, geometry is usually represented using polygonal meshes or point clouds. The former has the advantage of having a well-defined notion of connectivity and visibility; the latter is a more flexible representation and is usually the output of scanning technologies. Properties on a mesh are usually represented on a per-vertex or per-face basis and can be extended using textures. However, for point-clouds, textures are used less frequently.

We propose encoding some of the properties of a point cloud into textures. A panoramic image is always a valid and complete representation of a cloud

from a single scan location. Particularly, we propose using stretch-invariant polar-capped maps to encode these properties. These have the advantage of minimizing the distortion across every region of the unit sphere, yielding a more uniform sample distribution. We design a projection algorithm that maps points into texels to encode properties such as color, normals, and depth. These textures can be used for visualization but also have the advantage of enabling direct manual editing. This contribution is developed mainly throughout Section **4.3**, and the related results were published in [CAC+17; CCA19].

4 Effective Visualization of Sparse Image-to-Image Matches.

Robust feature matching between images is a critical task for different Computer Vision applications. Algorithms can easily fail in the presence of repetitive patterns/structures, reflective and mirror-like surfaces, and moving and occluding objects. These mismatches can negatively affect the optimization process for the registration and alignment of these images, motivating the urge to quickly detecting and removing them.

The easiest way of detecting feature mismatches could be a quick visual inspection. Nevertheless, state-of-the-art photogrammetry systems often produce cluttered caused by the representation of dozens of matches. To quickly spot mismatched features, we propose a hierarchical clustering strategy to group together similar matches. Isolated matches usually correspond to outliers; therefore, they become quick to spot. Moreover, we ease identifying two matched features by joining them using bent segments, using a carefully-chosen color palette, and drawing corresponding glyphs next to the features. This contribution is developed mainly throughout Section **5.2**, and the related results were published in [ACC20].

5 Semantic Analysis of Street-level Imagery for Improved Multi-view Stereo Reconstruction.

Multi-view stereo reconstruction is a widely-studied task that processes a set of input images and outputs a 3D representation of the content represented on those. The traditional pipeline starts by estimating the relative position of the images (camera extrinsics) and then densely reconstructs the scene using a plane-sweep algorithm. However, lately, an increasing number of authors propose using neural networks to estimate a coherent depth-map for each image.

Our approach uses semantic information computed on the images using neural networks to improve the reconstruction process. Particularly, we use this information in several steps of the traditional pipeline, for instance:

- To minimize the influence of non-static objects on the registration process.
- To repair depth and normal maps using priors that depend on the type of object which a given surface represents.
- To remove from the final reconstruction the contribution of unwanted occluding objects (e.g., trees).

This contribution is developed mainly throughout Section **5.3**, and the related results were published in [AAB+18].

6 Color Improvement through Heuristic Techniques and Photography-to-LiDAR Registration and Texturing.

A camera is usually used to take most of the color information on a scene. The most common photogrammetry artifacts are directly associated with the choice of camera settings. If the shutter speed is too slow, we will get motion-blur, if the exposition time is too long, we will get clipped highlights, if the aperture is too big, we will get shallower depth-of-field, and if not enough light reaches the sensor, we will get high photon-shot noise. Moreover, if we want to use this information for rendering, we need something as close as possible to the materials' physical properties. This is even more challenging because it means accounting for the multiple light sources that cause shadows and specular highlights in our recorded scene.

For LiDAR clouds, color information, if available, comes from a panoramic image generated from a mosaic of multiple images (each photography on the mosaic tries to capture the color for a spherical sector). This introduces further artifacts since the lighting conditions and camera settings may differ within a single cloud. We propose a simple color correction scheme that uses the LiDAR infrared (IR) intensity to obtain color information as coherently as possible within a cloud to address these issues.

For multi-view stereo, color information comes from the set of images from which the scene is reconstructed. Each image may present color variations due to the difference in lighting conditions and camera settings at the moment it was taken. However, even further coloring artifacts may occur if the depth estimation algorithm fails, and some foreground objects are pasted into the background (e.g., trees into facades). We propose using semantic labels and smart weights to improve the process of generating color information for these meshes. The same algorithm can project these images' colors into a LiDAR point cloud if there is an available registration between the two representations. This contribution is developed mainly throughout Subsection **4.3.3** and section **5.4**, and some of the related results were published in [CAC+17; CCA19].

7 Efficient and faithful visualization of massive point clouds using image-based techniques.

The visualization of massive geometric representations is a recurrent case of study in computer graphics literature. The algorithm that approaches these problems needs to leverage interactive inspection techniques with precise and complete data visualization. This usually is addressed by using multi-resolution techniques, i.e., when the geometry primitive maps to less than a screen pixel, then a lower resolution representative is used instead.

We propose two novel and efficient real-time interactive rendering techniques for point clouds. The first one is based on the image-based rendering paradigm. For this, we encode our point clouds into high-resolution panoramic textures (one for color, one for depth, and one for normals), which, at render time, are smartly tessellated in order to produce geometry. Our second approach is based on textured splats. We start by voxelizing a point cloud and producing multiple resolution levels for each voxel by iteratively decimating it. Then, we interactively select a different resolution for each voxel at render time together with splats of different sizes. Instead of using flat-colored splats, we take their color details from textures for enhanced visual results. This contribution is developed mainly throughout Chapter 6, and some of the related results were published in [CAC+17; CCA19].

1.2 DOCUMENT OUTLINE

The rest of this thesis manuscript has been structured as follows. In Chapter 2, we will introduce the notation used throughout the document and an overview of the datasets we will be using to test the proposed algorithms.

In Chapter 3, we present a survey on the various topics covered, such as normal estimation, surface reconstruction, point cloud simplification, point cloud visualization, multi-view stereo, and image segmentation.

Following, we explain our contributions, starting with our work with LiDAR data (Chapter 4), continuing with our work on imagery (Chapter 5), and finalizing with our work on the visualization of the point cloud (Chapter 6).

We conclude this document with Chapter 7, where we summarize our contributions and provide some ideas for future work.

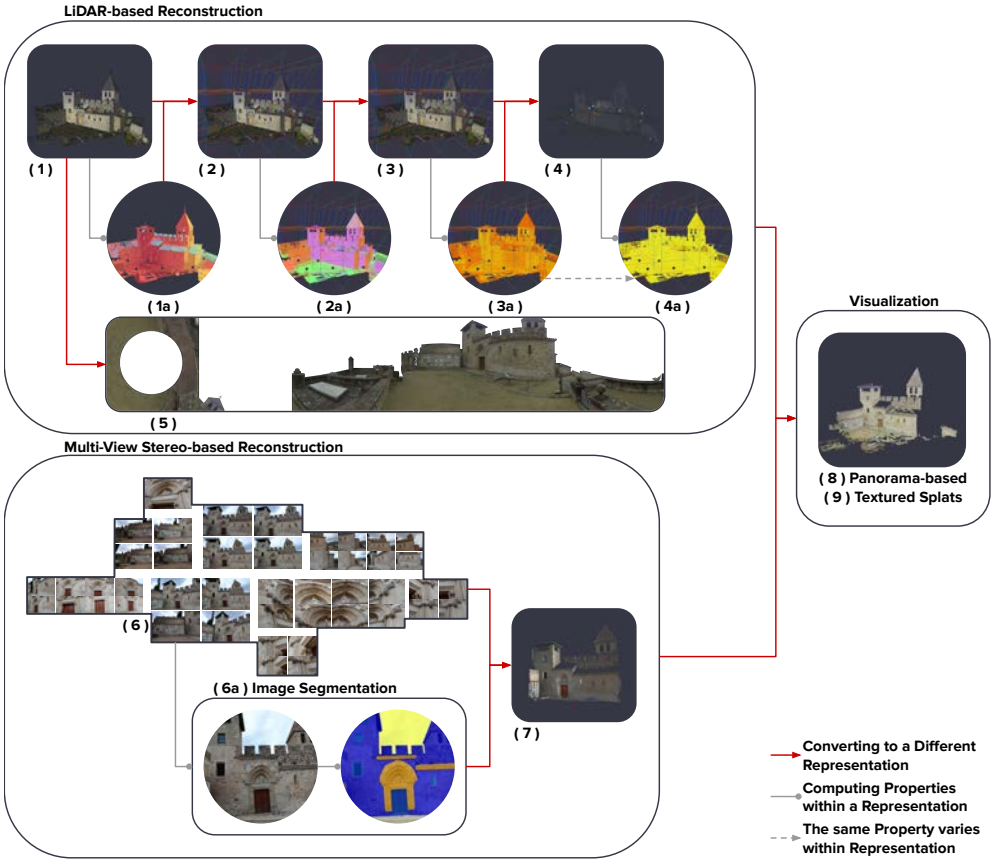


Figure 1.8: Overview of our contributions and relationships between them. We deal with two information sources: LiDAR data **(1)** and street-level imagery **(6)**. A LiDAR point cloud **(1)** is composed of several scans, and we assume we know a registration transformation for each of them. We can have multiple properties for each point, e.g., in **(1a)**, we can see the color-coded ID of the scan to which it belongs. Using the scanner specifications, we can also estimate its noise level.
To process a massive point cloud, we need to voxelize it **(2)**. Using this, we propose a new method for noise-aware normal estimation on LiDAR data **(2a)**.
Raw clouds have non-uniform point densities **(3a)** and we propose a new simplification method that removes redundant samples **(4)** yielding uniform densities **(4a)**. Alternatively, we propose encoding point properties (e.g., colors, normals) into a set of panoramas **(5)**. These can be used for fast image-based visualization **(8)** or, in combination with simplified clouds, for textured splat visualization **(9)**.
A set of images **(6)** can be converted into point clouds **(7)** using multi-view stereo software such as Colmap [SF16; SZP+16]. We propose improving these algorithms using semantic information **(6a)**. In particular, a combination of the Cityscapes [COR+16] and a novel neural network. These clouds can also be visualized using our approaches **(8)** and **(9)** but lack finer geometry detail. Instead, these and other techniques can be used to improve the color quality of LiDAR clouds.

1.3 PUBLICATION LIST

The contributions from this thesis have led to the following articles, two of them in indexed journals:

- M. Comino, C. Andujar, A. Chica, and P. Brunet. "Error-aware construction and rendering of multi-scan panoramas from massive point clouds". In: *Computer Vision and Image Understanding* 157 (2017). Large-Scale 3D Modeling of Urban Indoor or Outdoor Scenes from Images and Range Scans, pp. 43–54
- M. Comino, C. Andujar, A. Chica, and P. Brunet. "Sensor-aware Normal Estimation for Point Clouds from 3D Range Scans". In: *Computer Graphics Forum* 37.5 (2018), pp. 233–243
- C. Andújar, O. Argudo, I. Besora, P. Brunet, A. Chica, and M. Comino. "Depth Map Repairing for Building Reconstruction". In: *Spanish Computer Graphics Conference (CEIG)*. The Eurographics Association, 2018
- M. Comino Trinidad, A. Chica Calaf, and C. Andújar Gran. "View-dependent Hierarchical Rendering of Massive Point Clouds through Textured Splats". In: *Spanish Computer Graphics Conference (CEIG)*. The Eurographics Association, 2019
- C. Andujar, A. Chica, and M. Comino. "Effective Visualization of Sparse Image-to-Image Correspondences". In: *EuroVis 2020 - Short Papers*. The Eurographics Association, 2020
- M. Comino, A. Chica, and C. Andujar. "Easy Authoring of Image-Supported Short Stories for 3D Scanned Cultural Heritage". In: *Eurographics Workshop on Graphics and Cultural Heritage*. The Eurographics Association, 2020

An overview of this thesis was presented as a short talk and poster at the Eurographics 2019 Doctoral Consortium.

2

Preliminaries

In this Chapter, we will briefly introduce some of the notation used throughout the document and present the different datasets used for the experiments.

2.1 NOTATION

- $Q \subset \mathbb{R}^3$ denotes a set of 3D scan locations.
- $\mathbf{q} \in Q$ denotes an arbitrary scan location.
- C denotes an arbitrary point cloud. A point cloud is a list of points.
- $C_{\mathbf{q}}$ denotes a point cloud scanned from position $\mathbf{q}$.
- $C_Q = \bigcup_{\mathbf{q} \in Q} C_{\mathbf{q}}$ denotes the cloud resulting from the union of point clouds $C_{\mathbf{q}} | \mathbf{q} \in Q$.
- $T_{\mathbf{q}}$ denotes a texture associated to point cloud $C_{\mathbf{q}}$.
- $T_Q = \bigcup_{\mathbf{q} \in Q} T_{\mathbf{q}}$ denotes the set of textures for the point clouds in C_Q.
- $\mathbf{p}$ denotes an arbitrary point. Typically, each $\mathbf{p}$ will belong to a cloud C.
- $\tilde{\mathbf{p}}$ denotes the noisy counterpart of $\mathbf{p}$.
- $\mathbf{n_p}$ denotes the normal vector associated to $\mathbf{p}$.
- $\mathbf{d}$ denotes the laser beam direction.

- $d(\mathbf{p}, \mathbf{p}')$ denotes the Euclidean distance between points $\mathbf{p}$ and $\mathbf{p}'$.
- $\mathcal{N}_k(\mathbf{p})$ denotes the k-neighborhood of $\mathbf{p}$ in its corresponding cloud C.
- $\mathcal{N}_r(\mathbf{p})$ denotes the neighbors of $\mathbf{p}$ within search radius r in its corresponding cloud C.
- $\mathcal{N}_r^{\mathbf{q}}(\tilde{\mathbf{p}})$ denotes the neighbors of $\mathbf{p}$ within search radius r in its corresponding cloud $C_{\mathbf{q}}$.
- $\mathsf{N}_r(\tilde{\mathbf{p}}) = \bigcup_{\mathbf{q} \in Q} \mathcal{N}_r^{\mathbf{q}}(\tilde{\mathbf{p}})$ denotes the union of neighborhoods for $\mathbf{p}$ in the clouds $C_{\mathbf{q}} | \mathbf{q} \in Q$.
- $\mathbf{c}_{\mathbf{p}}$ denotes the centroid of some neighborhood of point $\mathbf{p}$.
- $\mathbf{p}^{\mathbf{c}} = \mathbf{p} - \mathbf{c}_{\mathbf{p}}$ denotes the centered $\mathbf{p}$.
- $\mathbf{CV}(\mathcal{N}_k(\mathbf{p}))$ denotes the covariance matrix of the points in $\mathcal{N}_k(\mathbf{p})$.
- $\lambda_{\mathbf{p}}^0 \leq \lambda_{\mathbf{p}}^1 \leq \lambda_{\mathbf{p}}^2$ denote the three eigenvalues of $\mathbf{CV}(\mathcal{N}_k(\mathbf{p}))$.
- $\rho_{\mathbf{p}}$ denotes the local density of C at $\mathbf{p}$.
- $i_{\mathbf{p}}$ denotes the reflected intensity by the material at point $\mathbf{p}$.
- $n_{\mathbf{p}}$ denotes the noise value associated to point $\mathbf{p}$.
- $\sigma_{\mathbf{p}}$ denotes the standard deviation of the noise associated with point $\mathbf{p}$.
- $s_{\mathbf{p}}^2$ denotes the sample variance associated with point $\mathbf{p}$.
- $\mathbf{s}(n_{\mathbf{p}}, \mathbf{p}^{\mathbf{c}})$ denotes the vector of sample covariances between the (x, y, z) components of the centered point $\mathbf{p}^{\mathbf{c}}$ and its associated noise $n_{\mathbf{p}}$.
- $\mathbf{R}$ denotes an arbitrary rotation matrix.
- $\mathbf{t}$ denotes an arbitrary translation vector.

2.2 DATASETS

In this thesis, we mostly work with two kinds of datasets: LiDAR point clouds and sets of facade photographs. The availability of point cloud data has increased in recent years, but it is still scarce compared to photogrammetric data availability. Thanks to the collaboration in different projects, we performed multiple captures of different cultural heritage sites. For these models, we have both LiDAR and photogrammetric data. Moreover, we also captured several dense sets of photographs from multiple facades.

2.3 LIDAR-BASED MODELS

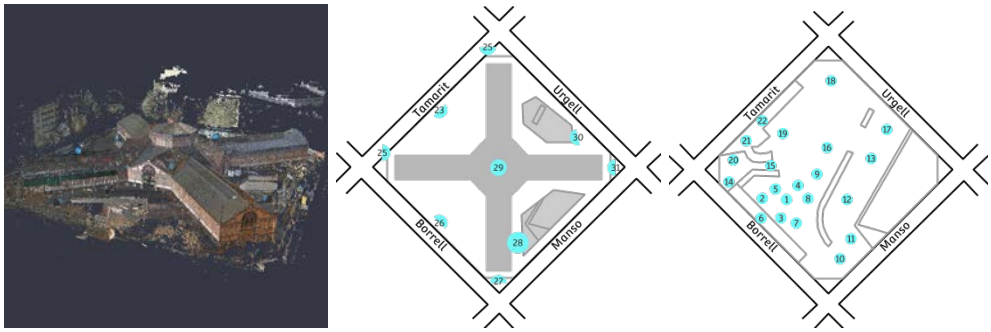
Nowadays, an increasing number of LiDAR point clouds are becoming freely available. Some examples are:

- The Open Heritage project [11], which provides photogrammetry, terrestrial LiDAR and aerial LiDAR point clouds from different cultural heritage sites.
- The Robotic 3D Scan Repository [12] from the Universität Osnabrück and Universität Wurzbürg, which provides multiple terrestrial LiDAR point clouds.
- The New York University Spatial Data Repository [10], which provides a huge aerial LiDAR scan of Dublin.

Nevertheless, we were very fortunate and could capture multiple cultural heritage sites, which we introduce next.

Mercat de Sant Antoni

The *Mercat de Sant Antoni* is a modernist building in Barcelona, with a steel structure composed of four big arms that converge in a large octagonal dome 28 meters high. Eight tall steel columns support this dome. The reshuffling works started in 2009 uncovered some archaeological remains of Barcelona's medieval wall.



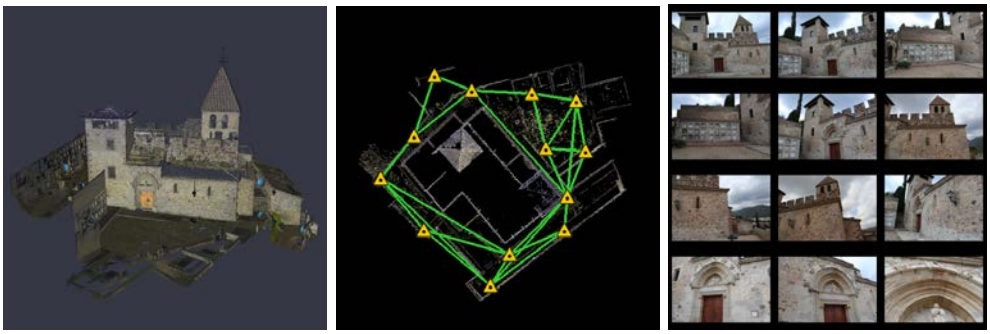
(1) **(2)** **(3)**

Figure 2.1: **(1)** Rendering of the captured point cloud for the street-level part of the *Mercat de Sant Antoni*. **(2)** Street-level scan locations. **(3)** Underground scan locations.

We performed a total of 31 scans (see Figure 2.1) using the Leica P20 scanner [8], most of them with 3mm spacing at 10 meters. The first 22 scans correspond to the building's underground (3115 Mpts), whereas the remaining ones correspond to the street-level part (372 Mpts). The point cloud containing the nine street-level scans of this building is referred to as the *market* model throughout the thesis.

Doma Church

The *Doma* church, located at La Garriga, is originally a Romanesque temple. However, only the central nave, built in the 12th century, remains from this period. In the 16th century, the building was enlarged by including the lateral nave (with the temple entrance), the deep chapel, the sacristy, the bell tower, the communal hall, and the choir. This church contains a magnificent Gothic altarpiece of San Esteban, the most relevant art piece of the complex, dating from the late 15th century.



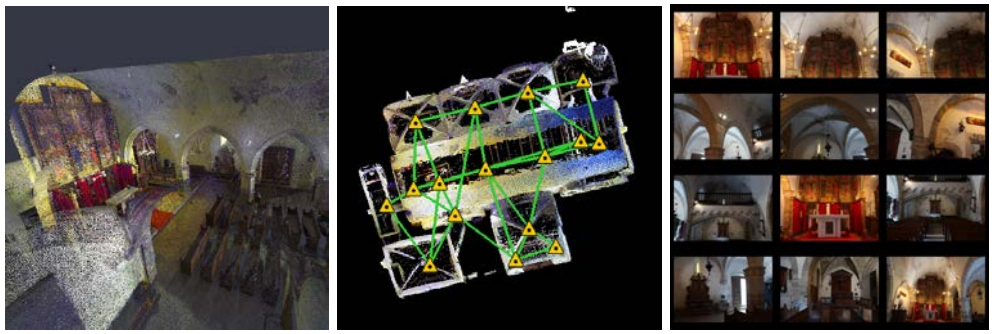
(1) **(2)** **(3)**

Figure 2.2: **(1)** Rendering of the captured point cloud, depicting the exterior of the *Doma*. **(2)** Map showing the different scan locations. **(3)** Sample photographs.

We performed a total of 30 scans (see Figures 2.2 and 2.3) using the Leica RTC360 scanner [9]. The first 14 scans correspond to the building's exterior (1026 Mpts), and the point cloud formed by these will be referred to as the *Doma interior* model throughout the thesis. The remaining 16 scans correspond to the building's interior (611 Mpts), and the point cloud formed by these will be referred to as the *Doma exterior* model throughout the thesis. We also captured several photographs of both interior and exterior using a Canon EOS M5.

Sant Quirze de Pedret

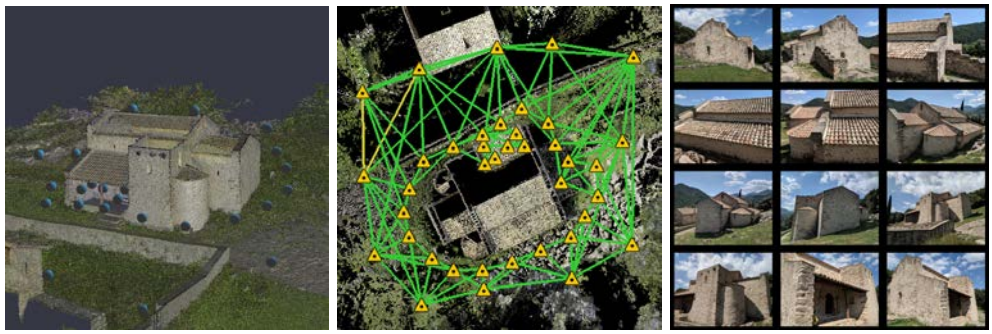
The original pre-Romanesque church dates from the 9th century. It was formed by a single nave (the current central one) with a trapezoidal apse, en-



(1) **(2)** **(3)**

Figure 2.3: **(1)** Rendering of the captured point cloud, depicting the interior of the *Doma*. **(2)** Map showing the different scan locations. **(3)** Sample photographs.

larged during the 10th century by adding two lateral naves and two apses. The current appearance resulted from successive smaller renovations and collapses. This church contained Romanesque mural paintings of a high historical value, which are currently preserved in the *Museu d'Art Nacional de Catalunya* in Barcelona (side apses) and in the *Museu Diocesà i Comarcal de Solsona* (central apse and fragments of the wall decorations).

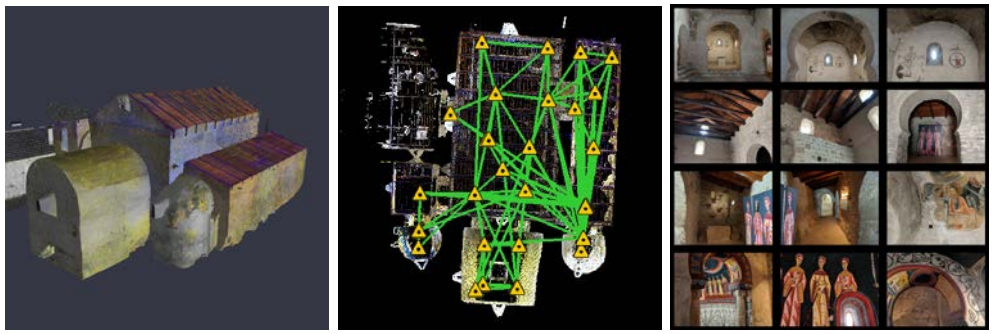


(1) **(2)** **(3)**

Figure 2.4: **(1)** Rendering of the captured point cloud, depicting the exterior of *Sant Quirze de Pedret*. **(2)** Map showing the different scan locations. **(3)** Sample photographs.

We performed a total of 68 scans (see Figures **2.4** and **2.5**) using the Leica RTC360 scanner [9]. The first 39 scans correspond to the building's exterior (1191 Mpts), and the point cloud formed by these will be referred to as the *Pedret interior* model throughout the thesis. The remaining 29 scans correspond to the building's interior (1213 Mpts), and the point cloud formed by these will be referred to as the *Pedret exterior* model throughout the thesis. We also captured

several photographs of both interior and exterior using a Canon EOS M5.

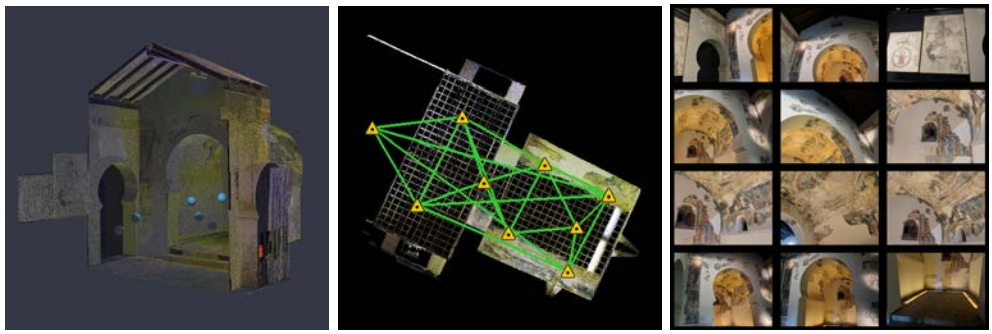


(1) **(2)** **(3)**

Figure 2.5: **(1)** Rendering of the captured point cloud, depicting the interior of *Sant Quirze de Pedret*. **(2)** Map showing the different scan locations. **(3)** Sample photographs.

Museu Diocesà i Comarcal de Solsona

Parts of the Sant Quirze de Pedret mural paintings are currently preserved at the *Museu Diocesà i Comarcal de Solsona*. We digitized the room where these paintings are located by performing a total of 9 scans (see Figure **2.6**) using the Leica RTC360 scanner [9]. The point cloud formed by these (1250 Mpts) will be referred to as the *Museum* model throughout the thesis. We also captured several photographs of the room using a Canon EOS M5.

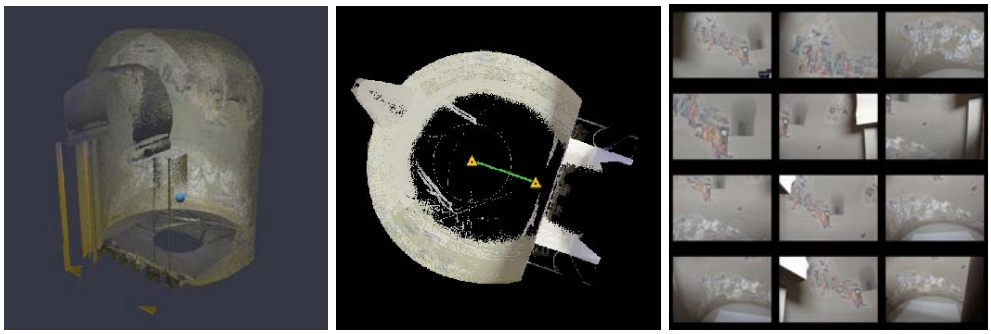


(1) **(2)** **(3)**

Figure 2.6: **(1)** Rendering of the captured point cloud, depicting the central apse mural paintings of *Sant Quirze de Pedret* preserved at the *Museu Diocesà i Comarcal de Solsona*. **(2)** Map showing the different scan locations. **(3)** Sample photographs.

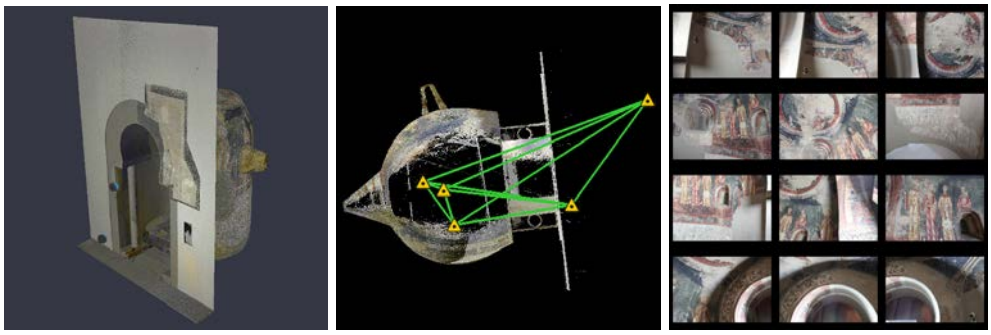
Museu d'Art Nacional de Catalunya

Parts of the Sant Quirze de Pedret mural paintings are currently preserved at the *Museu d'Art Nacional de Catalunya*. We digitized the rooms where these painting are located by performing a total of 7 scans (see Figures **2.7** and **2.8**) using the Leica RTC360 scanner [9]. The first 2 scans correspond to the paintings on the north apse (297 Mpts), and the point cloud formed by these will be referred to as the *Apse A* model throughout the thesis. The remaining 7 scans correspond to the paintings on the south apse (557 Mpts), and the point cloud formed by these will be referred to as the *Apse B* model throughout the thesis. We also captured several photographs of both apses using a Canon EOS M5.



(1) **(2)** **(3)**

Figure 2.7: **(1)** Rendering of the captured point cloud, depicting the north apse mural paintings of *Sant Quirze de Pedret* preserved at the *Museu d'Art Nacional de Catalunya*. **(2)** Map showing the different scan locations. **(3)** Sample photographs.



(1) **(2)** **(3)**

Figure 2.8: **(1)** Rendering of the captured point cloud, depicting the south apse mural paintings of *Sant Quirze de Pedret* preserved at the *Museu d'Art Nacional de Catalunya*. **(2)** Map showing the different scan locations. **(3)** Sample photographs.

2.4 PHOTOGRAPHS COLLECTIONS

Nowadays, there are numerous sources with abundant facade images:

- The CMP Facade dataset [16], which provides several facades segmented into different elements.
- The Cityscapes dataset [COR+16], which provides street-view imagery where the facades have been segmented.
- Ceylan et al. [CMZ+14] provide 9 collections of 20 to 70 photographs of buildings for multi-view stereo reconstruction.
- Schoeps et al. [15] provide several collections of photographs for multi-view stereo benchmarking, some of which contain buildings.

Nevertheless, we need high-resolution enough images to produce dense stereo reconstructions. Moreover, we also estimated that to obtain these, we needed around 30 images for each facade. Consequently we captured several collections (see Figure 2.9) of images for multiple facades. These images' resolution is approximately $4k \times 3k$ pixels, and they were captured with a Canon EOS M5.

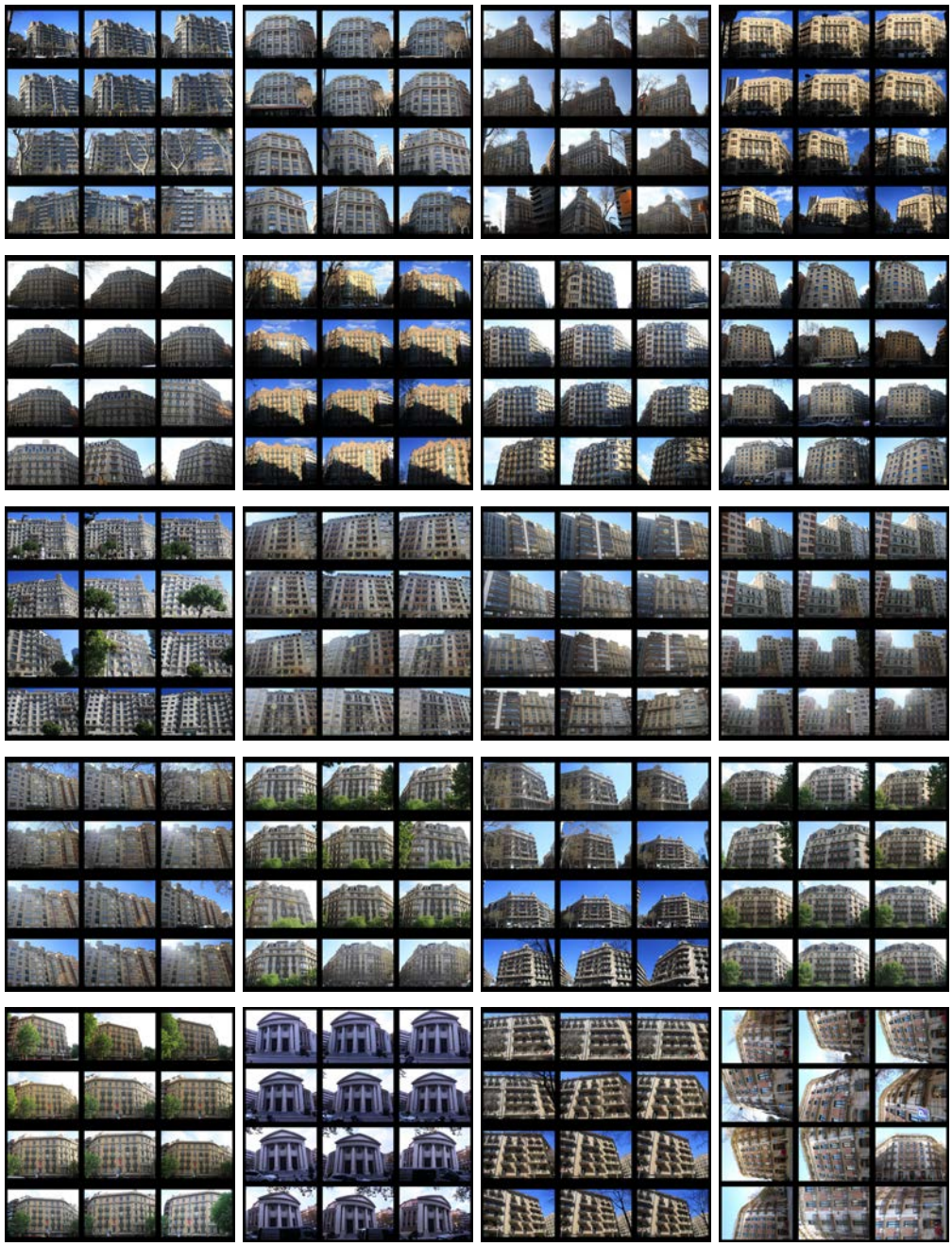


Figure 2.9: Several photographs collections of facades from European cities. Each collection contains about 25 to 70 images of the same facade. Images' resolution is approximately $4k \times 3k$ pixels, and they were captured with a Canon EOS M5.

3

State of the Art

In this Chapter, we will review the most relevant efforts related to our work and our field of study. We are mainly concerned with techniques for processing and rendering point-based representations. However, within the vast amount of techniques in this field, we will focus on those closely connected with our contributions. These could be summarized as normal estimation and simplification techniques for point clouds, multi-view stereo and semantic extraction techniques for images, and splat-based rendering techniques for points.

3.1 POINT-BASED REPRESENTATIONS

Point-based representations first became popular during the early 2000s because they posed a series of advantages compared to mesh representations. For instance, Kobbelt et al. [KB04] and Alexa et al. [ABC+03; AGP+04a] observe that, in very complex meshes, a triangle can map to less than one screen pixel. Hence, it is cheaper to use point primitives directly instead of more complex ones.

Another property of point representations is that they usually lack topological information (hence the name *unstructured point clouds*). Thus, different neighborhood definitions have been proposed in the literature. These include the Euclidean neighborhood, the k-neighborhood, the geodesic neighborhood, or the Voronoi neighborhood. Consequently, tasks such as normal estimation, simplification, edge recovering, or smoothing are less straightforward than for mesh-based representations. However, due to this lack of structure, point clouds

are also considered a more flexible representation [KB04], which is advantageous for tasks such as applying morphology operators [CB14].

A little afterward, point clouds left the spotlight. As the gaming and film industry became more prominent, the popularity of mesh representations got reinforced. Acceleration hardware required to run video games became more and more optimized for the processing of polygonal meshes (especially triangle meshes). Moreover, visual effects artists preferred the use of subdivision surfaces that work on top of quad meshes. However, in recent years, with the advent of Deep Learning (DL) and Neural Networks, point clouds regained notoriety.

The typical pipelines for processing acquired raw point cloud data usually include a subset of the following tasks: registration between different scans, feature estimation, fairing or simplification, smoothing and consolidation, edition, and surface reconstruction. On the one hand, there are many proposed traditional techniques for each of these tasks. Usually, each technique is built on top of a set of assumptions and priors, which are later evaluated. On the other hand, given the target task, Deep Learning aims to model these priors from the data automatically. Because raw point clouds have not undergone any processing and preserve the original geometric information, many authors prefer learning directly on this representation [GWH+19] than on more refined ones (such as meshes).

In our case, we focus on point-based representations because they are the natural output of digitization techniques. LiDAR devices [8; 9; Wal15] or multi-view stereo approaches [CVH+08; MK10; JP11; MMM12; SF16; SZP+16; BCM18; FK18; TGF+15; STM+19], their output are unstructured point clouds. Moreover, measurements between points are free of the errors possibly introduced during triangulation.

The remaining of this Chapter reviews the literature about the tasks in a typical point cloud processing pipeline. In Sections **3.2** and **3.3**, we review core problems in the acquisition of raw data. In Section **3.2**, we describe techniques for estimating point clouds from imagery using multi-view stereo, whereas, in Section **3.3**, we briefly describe the most relevant work on the registration of point clouds. In Section **3.4**, we review traditional approaches for the estimation of features on top of points, putting particular emphasis on normal estimation. Following, in Section **3.5**, we analyze existing techniques for the simplification and consolidation of point clouds, and in Section **3.6**, we briefly report existing tools for their edition. Section **3.7** briefly surveys the most common techniques for reconstructing meshes from these clouds and, finally, Section **3.8** focus on the existing work on the visualization techniques specially designed for point-based representations.

3.2 POINT CLOUD ACQUISITION THROUGH MULTI-VIEW STEREO

Point clouds acquired using multi-view stereo and other photogrammetric techniques suffer from different artifacts, which can also be found in those captured with LiDAR devices or RGB-D cameras. These include noise, misalignments, missing parts, and outliers. Addressing these artifacts and generating a faithful representation of reality is an open-ended problem, and, because of this, there exist different open source and commercial tools. State-of-the-art open source packages include Colmap [SF16; SZP+16] (which has been recently shown to provide the best average results on different datasets among open-source SfM implementations [BCM18]) OpenMVG [MMP+16] and AliceVision Meshroom [MMM12; JP11]. Some popular commercial solutions are Agisoft Metashape Pro [1], Reality Capture, Zephyr3D [STM+19; TGF+15] and Autodesk ReCap Photo 2020 [2].

Most photogrammetric pipelines start by computing a sparse reconstruction of the scene using Structure-from-Motion [PVV+04; AFM+06; PNF+08]. More specifically, SIFT [Low04]-like features are computed on each input image, and these are used to estimate both camera intrinsics (the camera model parameters) and the camera extrinsics (relative positions of the camera in the 3D scene). After this, a dense reconstruction of the scene is obtained by estimating one depth map for each image. These are later fused into a point cloud. Individual depth maps are usually computed using a plane-sweep algorithm [Col96]. Namely, the target image is reprojected into neighboring ones, assuming different depths, and the depth for each pixel is estimated as the one that minimizes the reprojection loss.

The major challenges in this type of reconstructions have been studied extensively for general objects [SCD+06; MWA+13; BCM18]. Concerning buildings, the most significant challenges are related to texture-less and specular regions, occluding objects, and thin objects. SIFT features cannot robustly be estimated on texture-less regions (e.g., homogeneous walls on facades). Moreover, it is tough to estimate a confident depth value on these regions since multiple reprojection depths can give relatively low loss values. Mirror-like surfaces (e.g., windows) are also quite tricky because they will display inconsistent content across different views. Occluding objects (e.g., street lights or trees) may partially hide relevant content and may also end-up being pasted into the background. Finally, thin objects (e.g., balcony rails) usually do not appear coherently across different images and are poorly reconstructed.

These challenges occur because traditional techniques make general assumptions [BTS+14; BTS+17] (such as surface smoothness) on scenes where a great variety of different objects coexisting. Hence, more recent works study how to apply domain knowledge to treat each case specifically [HZC+13; SYZ+17;

HMF+18].

Campbell et al. [CVH+08] extend the traditional dense reconstruction algorithm by storing multiple depths candidates (hypothesis) instead of just storing the depth the minimizes the reprojection loss. After this, an optimization process that considers spatial constraints determines the final depth for each pixel. This avoids the outliers introduced by registration mismatches due to repeated textures, occlusion, and texture-less regions, on the final depth maps.

Hane et al. [HZC+13] propose reconstructing scenes from images by jointly optimizing their segmentation and their 3D reconstruction. They adapt the reconstruction algorithm according to the class assigned to each surface. More specifically, they formulate class-specific smoothness assumptions using domain knowledge.

Instead of jointly performing reconstruction and labeling, Frohlich et al. [FK18] show how semantic information can be effectively applied to improve the reconstruction of digitized scenes. They study performing planar reconstruction of facades and buildings from a set of labeled images. The correspondences between segmented planar regions are used to simultaneously estimate the relative-pose of the images and the final geometry. Because the reconstruction process is constrained to a set of planes, the final meshes lack high-frequency detail. Furthermore, another limitation is that this algorithm requires labels and correspondences, which are not automatically generated.

Some authors exploit the fact that urban models can be coarsely approximated as a set of flat surfaces [MK10; BRV15; NW17; HMF+18]. These methods generally produce feasible reconstructions even for sparse sets of images but fail to provide finer detail when a large set of images are available. Mičušík et al. [MK10] and Bódis-Szomorú et al [BRV15] propose abstracting images as a set of superpixels and using these as the basic processing unit. In both cases, this highly speeds-up the algorithms. Mičušík et al. [MK10] constrain the set of possible plane orientations to the dominant directions in the scene. They argue that, by using larger groups of pixels than other algorithms, they improve overall image consistency and achieve more robust results in texture-less areas. Bódis-Szomorú et al [BRV15] extracts a 2D mesh by performing a Delauney Triangulation on the input image gradients. The obtained triangles (analogous to superpixels) are projected to 3D using the sparse point cloud that results from a previous Structure-from-Motion step.

With the advent of Deep Learning, high-accuracy automatically labeling scene objects has become a feasible task [CPS+17; COR+16; IZZ+17]. Holzman et al. [HMF+18] perform semantic segmentation of their scenes to distinguish be-

tween buildings and other objects. Scenes are reconstructed from an SfM-MVS pipeline, and for the segmentation step, they use fully-convolutional networks to segment their input images. Label images are then projected onto the resulting point clouds, allowing different reconstructing algorithms depending on the class label. However, similar to Nan et al. [NW17], they aim to create very simple and coarse representations of urban scenes.

Thus, rather than relying on the smoothness prior, it is better to exploit any available domain knowledge. One possibility is to use symmetries in our input models to repair them. Pauly et al. [PMW+08] observe that, after applying the appropriate transformation, repetitions of a part will appear as a uniform grid. This property can be used to detect these repetitions and repair incomplete instances. Zheng et al. [ZSW+10] extended this approach to process terrestrial LiDAR point clouds. Furthermore, Li et al. [LZS+11] also adapted this algorithm to use the available color information. Finally, Ceylan et al. [CMZ+14] perform symmetry detection and sparse scene reconstruction simultaneously as a global optimization process. Nevertheless, some common features in urban models cannot be handle by these techniques, and their performance would make them prohibitive for large datasets.

Another approach by Przemyslaw et al. [PPM+09] detects symmetries in facades using Monte Carlo sampling and then propagates these symmetries while removing unwanted objects from the images. The difference between the repaired image and the original one could yield a mask containing occluding objects. However, this would be limited to regions with strong symmetries. Another shortcoming is that the repaired images can not be used for photogrammetry reconstruction since they are processed individually. Hence, there are no coherence guarantees between multiple views.

Symmetry allows establishing relations for similar content in a scene but does not provide any information about the type of content itself. Song et al [SYZ+17] present an end-to-end 3D convolutional network that estimates occupancy and object labels for a complete 3D voxelization. The input to their system is just 2D depth maps. It would be possible to use these labels to reconstruct each typical component of a building with algorithms specially designed to deal with each case.

GPU memory consumption significantly constrains the size of the voxelizations that can be processed (about a maximum of 128^3). Since depth maps are a 2.5D representation, it is generally more efficient to learn in the image space than on a 3D grid. Semantic segmentation (or pixel-wise classification) is a widely studied topic in Computer Vision. We need to correctly identify objects in a scene to apply domain knowledge correctly. There exist specific works tar-

geting the case of facades. For instance, Teboul et al. [TSK+10] observe that the disparity between pixels belonging to the same facade element is challenging for traditional ML-based classifiers. Therefore, they propose using shape grammars to improve the result of learning methods.

Nevertheless, most recent methods rely on deep neural networks for this task [COR+16; SM16; LZZ+17; FPM+18]. Cordts et al. [COR+16] introduce the Cityscapes dataset. Their main objective is generating coarse segmentation of urban environments, discerning between classes such as facades, trees, vehicles, pedestrians, and roads. Another example by Femiani et al. [FPM+18] adapts Segnet [BKC17] to produce a finer and robust segmentation of facade images in different formats (e.g., panoramic, rectified, cropped).

3.3 POINT CLOUD REGISTRATION

After scanning a scene from multiple locations, different point clouds are produced, but they are usually in their own coordinate system. Hence, there is a need for an automatic method that can determine the relative positions and rotations between them. Although newer scanning technology can estimate a coarse registration during the scanning process [9], this result is only approximate and needs to be refined.

The most classical algorithm for this task is called Iterative Closest Points [BM92]. Given a source cloud C_s and a target cloud C_t, the ICP method tries to align C_s to C_t using only rigid motion. Namely, it tries to compute the rotation $\mathbf{R}$ and the translation $\mathbf{t}$ that minimize the following energy:

$$E = \sum_{\mathbf{p} \in C_s} \|\mathbf{R}\mathbf{p} + \mathbf{t} - \mathbf{p}_t\|^2 \tag{3.1}$$

where $\mathbf{p}_t$ is the correspondence of $\mathbf{p}$ in C_t.

The algorithm tries to minimize this energy by iterating two steps:

1. Given a rotation $\mathbf{R}$ and the translations $\mathbf{t}$ find the correspondences $\mathbf{p}_t \in C_t$ to each point $\mathbf{R}\mathbf{p} + \mathbf{t} | \mathbf{p} \in C_s$.
2. Given fixed correspondences $\mathbf{p}_t \in C_t$, estimate $\underset{\mathbf{R},\mathbf{t}}{\operatorname{argmin}} \sum_{\mathbf{p} \in C_s} \|\mathbf{R}\mathbf{p} + \mathbf{t} - \mathbf{p}_t\|^2$

3 Iterate (1) and (2) until E becomes smaller than a given threshold.

Delving into the different proposed variants of this algorithm is out of this thesis's scope, and the reader can refer to existing surveys [RL01; TCL+13; BSB+14; MGG17; CCL+18] for more details.

Nevertheless, ICP (and its variants) work well when consecutive scans are nearly aligned. For the case of point clouds captured using LiDAR devices, this does not usually happen. Hence, an initial coarse alignment is needed. Meshlab [CCC+08] has a tool that allows the user to select homologous points if the source and target clouds and uses these correspondences to generate a coarse alignment.

Most automatic algorithms for the coarse alignment of point clouds are based on RANSAC alignment. The typical approach randomly selects a base formed by three points from the source cloud C_s and another base from C_t, computes the rigid transformation ($\mathbf{R}$, $\mathbf{t}$) that aligns them and evaluates their quality. This is repeated several times, and the transformation with the best quality is selected. Aiger et al. [AMC08] improve this approach and propose a robust and automatic way for globally aligning two point sets. Their key observation is that the problem becomes easier when considering bases of 4 co-planar points. For a 4-point base in C_s, they show they can find all possible candidate alignment bases in C_s in quadratic time with respect to $|C_s|$. Mellado et al. [MAM14] further improves this method and achieve linear time performance with respect to $|C_s|$.

3.4 FEATURE ESTIMATION ON POINT CLOUDS

The notion of *feature* for point clouds refers to values that locally describe the geometry at any 3D point in the space. However, features are usually computed and associated with the individual points composing the cloud. The feature information is especially essential for simplification or surface reconstruction since it allows applying domain knowledge to treat differently distinct parts of the cloud. For instance, feature information can be used to preserve sharp edges effectively.

The most commonly used feature is the surface normal, namely a unit 3D vector which describes the orientation of the underlying surface at each point. Hoppe et al. [HDD+92] were the first authors that proposed estimating this vector using the Principal Component Analysis (PCA) on unstructured point clouds.

This method inherently assumes that the underlying surface is smooth. Moreover it works on the assumption that points and randomly distributed around this surface and that their distances to it, in the tangential direction, follow a certain standard normal distribution. More formally, given points $\mathbf{p}$ and its k-neighborhood $\mathcal{N}_k(\mathbf{p})$, their method tries to compute the plane $\mathbf{n}_\mathbf{p}^T x - d = 0$ for point $\mathbf{p}$ that minimizes the squared distances to the points $\mathbf{p}_i \in \mathcal{N}_k(\mathbf{p})$:

$$\begin{aligned} &\underset{\mathbf{n_p},d}{\operatorname{argmin}} \left(\frac{1}{2k} \sum_{\mathbf{p}_i \in \mathcal{N}_k(\mathbf{p})} (\mathbf{n}_\mathbf{p}^T \mathbf{p}_i - d)^2 \right) \\ &\text{subject to } ||\mathbf{n_p}|| = 1 \end{aligned} \tag{3.2}$$

where $k = |\mathcal{N}_r(\mathbf{p})|$. Given the centroid $\mathbf{c_p}$ of $\mathcal{N}_k(\mathbf{p})$, the vector $\mathbf{n_p}$ that minimized the above function can be estimated through the eigenvalue analysis of the matrix $\mathbf{CV}(\mathcal{N}_k(\mathbf{p}))$:

$$\begin{aligned} \mathbf{c_p} &= \frac{1}{k} \sum_{\mathbf{p}_i \in \mathcal{N}_k(\mathbf{p})} \mathbf{p}_i \\ \mathbf{CV}(\mathcal{N}_k(\mathbf{p})) &= \sum_{\mathbf{p}_i \in \mathcal{N}_k(\mathbf{p})} (\mathbf{p}_i - \mathbf{c_p})(\mathbf{p}_i - \mathbf{c_p})^T \end{aligned} \tag{3.3}$$

Let $\lambda_\mathbf{p}^0, \lambda_\mathbf{p}^1, \lambda_\mathbf{p}^2$ be the smallest, second smallest and largest eigenvalues of $\mathbf{CV}(\mathcal{N}_k(\mathbf{p}))$, respectively. As we formulated a minimization problem, we should pick the eigenvector with the smallest eigenvalue ($\lambda_\mathbf{p}^0$) as the normal vector estimate.

Most traditional algorithms in the literature build on top of the PCA approach. Newer approaches based on Neural Networks and Deep Learning fall out of this thesis's scope and will be briefly discussed later (Section **3.9**). At inference time, these approaches usually can perform a smaller number of queries per second than traditional methods (in some cases, their performance is even prohibitive). For normal estimation, PCA-based approaches already perform reasonably well (in a qualitative sense). We believe DL-based strategies are more useful when the nature of the problem involves a probabilistic estimation, such as for object detection or segmentation (Section **3.9**).

Before delving into normals, we will review few works that study other descriptors used to characterize points. Gumhold et al. [GUM01] propose a series of

descriptors used to determine whether a point belongs to a flat surface, a crease, a border or a corner. For instance, they propose estimating the curvature at $\mathbf{p}$ as:

$$k_{\mathbf{p}} = \frac{2 * |\mathbf{n_p}(\mathbf{p} - \mathbf{c_p})|}{\left(\frac{1}{k}\sum_{\mathbf{p}_i \in \mathcal{N}_k(\mathbf{p})} ||\mathbf{p} - \mathbf{p}_i||\right)^2} \tag{3.4}$$

And also other weights such as the corner penalty $\omega^{co}_{\mathbf{p}}$, the border penalty $\omega^{b2}_{\mathbf{p}}$, the border direction $\vec{\omega}^{b1}_{\mathbf{p}}$ penalty and the border crease penalty $\vec{\omega}^{cr}_{\mathbf{p}}$:

$$\begin{aligned}
\omega^{co}_{\mathbf{p}} &= \frac{\lambda^2_{\mathbf{p}} - \lambda^0_{\mathbf{p}}}{\lambda^2_{\mathbf{p}}} \\
\omega^{b2}_{\mathbf{p}} &= 1 - \frac{\beta_{\mathbf{p}}}{2\pi} \\
\vec{\omega}^{b1}_{\mathbf{p}} &= \frac{|\lambda^2_{\mathbf{p}} - 2\lambda^1_{\mathbf{p}}|}{\lambda^2_{\mathbf{p}}} e^2_{\mathbf{p}} \\
\vec{\omega}^{c2}_{\mathbf{p}} &= \frac{\max(\lambda^1_{\mathbf{p}} - \lambda^0_{\mathbf{p}}, \lambda^2_{\mathbf{p}} - \lambda^0_{\mathbf{p}} - \lambda^1_{\mathbf{p}})}{\lambda^2_{\mathbf{p}}} e^2_{\mathbf{p}}
\end{aligned} \tag{3.5}$$

where e^2 is the eigenvector corresponding to the largest eigenvalue ($\lambda_{\mathbf{p}}$) of the matrix $\mathbf{CV}(\mathcal{N}_k(\mathbf{p}))$ and β is the maximum angle interval between the vectors $\mathbf{p} - \mathbf{p}_i | \mathbf{p}_i \in \mathcal{N}_k(\mathbf{p})$ projected onto the plane $\mathbf{n}^T_{\mathbf{p}} x - d = 0$ that does not contain any other point.

Later, Pauly et al. [PGK02] presented a method for point cloud simplification that uses a measure of *surface variation* to determine which regions contain high-frequency geometry, in order to preserve them. The surface variation measure is defined as:

$$sv^k_{\mathbf{p}} = \frac{\lambda^0_{\mathbf{p}}}{\lambda^2_{\mathbf{p}} + \lambda^1_{\mathbf{p}} + \lambda^0_{\mathbf{p}}} \tag{3.6}$$

k again is the number of neighbors. This measure is 0 if the surface is flat and 1/3 when points are isotropically distributed.

Pauly et al. [PKG03] also extend surface variation to a multi-resolutional feature ω^{sv}. They compute this value for different neighborhood sizes and count how many times $sv^k_{\mathbf{p}}$ exceeds a threshold sv_{max}:

$$\begin{aligned} \Omega^k_{\mathbf{p}} &= \begin{cases} 1, & sv^k_{\mathbf{p}} > sv_{max} \\ 0, & sv^k_{\mathbf{p}} \leq sv_{max} \end{cases} \\ \omega^{sv}_{\mathbf{p}} &= \sum_k \Omega^k_{\mathbf{p}} \end{aligned} \tag{3.7}$$

While these very concise and specific descriptors try to model very defined properties, for learning, we would usually need to provide a broader set of unspecific values and let the learning algorithm decide which of them are valuable. In this sense, Frome et al. [FHK+04] extended shape context descriptors to 3D. These are a radial histogram, centered at each point, which counts how many points fall on each bin. More recently, Khoury et al. [KZK17] proposed learning Compact Geometry Features (CGF) by embedding shape context descriptors into a lower-dimensional space.

Moving into normal estimation, we could differentiate the many works that tackle this task on those that perform a covariance analysis on the local k-neighborhood around each point [HDD+92; ABC+01; PGK02; MN03; HLZ+09; HWG+13; CLZ13; ZCL+13; NBW14; LY15; LZC+15; KL17; CCZ+18; SDC+20], Voronoi-based methods [AB99; DLS05; DS06; ACT+07; MOG11], those that do plane probing through RANSAC [LSK+10; BM12; BM16; ZPL+19], those that fit more complex surfaces [CP05; GG07; CGA+13], and on those that rely on Deep Learning [BM16; QSM+17; QYS+17; AML18; HRV+18; GKO+18].

Each of these works was designed to be robust in a specific task. For instance, some of them focus on being adaptive to the noise level [MN03; DS06; MOG11; CLZ13; ZCL+13], robust against outliers [LCL+07; HLZ+09; NBW14; LY15; KL17; SDC+20] or keen on edge preservation [LSK+10; MOG11; BM12; CLZ13; ZCL+13; LZC+15; HWG+13; BM16; CCZ+18].

This shows that normal estimation is a widely studied topic. Several rendering and surface reconstruction [BTS+14; BTS+17] techniques rely on accurate normals. It is hard to find an algorithm that will estimate them robustly for each of the vast diversity of types of point clouds coming from different data sources.

Alexa et al. [ABC+01; ABC+03] were the first to introduce a weighted version of the least squares fitting problem. Their moving least squares (MLS) surfaces give more importance to points that are closer to the query point:

$$\begin{aligned} &\underset{\mathbf{n_p},d}{\operatorname{argmin}} \left(\frac{1}{2k} \sum_{\mathbf{p}_i \in \mathcal{N}_k(\mathbf{p})} (\mathbf{n_p}^T \mathbf{p}_i - d)^2 f_a\left(||\mathbf{p}_i - \mathbf{p}||\right) \right) \\ &f_a(x) = e^{\frac{-x^2}{k_a^2}} \\ &\text{subject to } ||\mathbf{n_p}|| = 1 \end{aligned} \tag{3.8}$$

for some user-defined parameter k_a.

Lipman et al. [LCL06] present a data-independent tight error bound and a data-dependent error function approximation for the MLS approach. This error analysis can guide the selection of a local support size that ensures minimal error.

Mitra et al. [MN03] noted that when choosing the neighborhood size there is trade-off between filtering the noise in the cloud and preserving the local curvature. They find that, under the same assumptions as Hoppe et al. [HDD+92], the optimal neighborhood search radius, with probability $1 - epsilon$, should be:

$$r = \left(\frac{1}{k_\mathbf{p}} \left(c_1 \frac{\sigma_\mathbf{p}}{\sqrt{\epsilon \rho_\mathbf{p}}} + c_2 \sigma_\mathbf{p}^2 \right) \right) \tag{3.9}$$

for some user-defined parameters c_1 and c_2 and where $\|_\mathbf{p}$ is the local curvature at $\mathbf{p}$, $\sigma_\mathbf{p}$ is the standard deviation of the noise level at $\mathbf{p}$ and $\rho_\mathbf{p}$ is the local density at $\mathbf{p}$.

Andersson et al. [AGP+04b] study the lower and higher bounds on the number of k-neighbors that can guarantee that all Delaunay neighbors are captured, which allows faithfully approximating the local geometric properties.

Alexa et al. [ABC+01; ABC+03]'s method was first devised as a projection operator to remove the noise and consolidate a point cloud. Following this philosophy, Lipman et al. [LCL+07] introduced a new method or point cloud consolidation requiring no normals. Their parameterization-free Local Projection Operator (LOP) aims to project a set of points onto their local multivariate median while ensuring that the resulting points are as uniformly distributed as possible. Huang et al. [HLZ+09] extend this method to deal with non-uniform sampled input clouds. After consolidation, an initial estimate for normals is computed using traditional weighted PCA. These are improved using a corrector loop that alternates a consistent normal orientation step and one orientation-aware PCA step. Later, the authors presented improvements [HWG+13] on top of their

LOP operator to preserve sharp edges. Using their new Edge Aware Re-sampling (EAR) method, point clouds are smartly re-sampled away from edges, where robust normals can be estimated. Then, these normals are propagated while the original point set is re-sampled towards the edges. However, these three methods are re-sampling methods, as discussed in Section **3.5**, which means they compute normals on new point clouds that are different from the original ones.

Another sharp feature-preserving algorithm is given by Castillo et al. [CLZ13]. They reformulated the least squares fitting problem by introducing weights that deflate the contributing of points which lay far away from the estimated planes. They simultaneously optimized for the position of $\tilde{\mathbf{p}}$ allowing it to move a certain distance t along the normal direction:

$$\begin{aligned} &\underset{\mathbf{n_p},t}{\operatorname{argmin}} \left(\frac{1}{2} \sum_{\mathbf{p}_i \in \mathcal{N}_k(\mathbf{p})} \left(\exp\left(-\lambda_{cast}(\mathbf{n}_\mathbf{p}^T(\mathbf{p}_i - \mathbf{p}) - t)^2\right)(\mathbf{n}_\mathbf{p}^T(\mathbf{p}_i - \mathbf{p}) - t)\right)^2 \right) \\ &\text{subject to } ||\mathbf{n_p}|| = 1 \\ &\qquad\qquad |t| \leq T \end{aligned} \tag{3.10}$$

where λ_{cast} is a user-defined parameter, and T is a bound on the distance from $\mathbf{p}$ to the underlying surface. Notice this formulation can no longer be solved using PCA, and they need to use a non-linear solver.

Zhang et al. [ZCL+13] estimated a confidence value for the PCA normals so that those under a certain threshold are considered to belong to a feature potentially. Afterward, the neighborhood of each feature point is segmented into planar regions using a low-rank subspace clustering. Finally, they report each point's normal as the estimate computed using PCA on the samples belonging to the subspace with minimum fitting residual. Liu et al. [LZC+15] improve their algorithm by reformulating the subspace segmentation model using a least-squares representation. They achieve similar quality as their previous method while improving their performance.

Nurunnabi et al. [NBW14] used the Minimum Covariance Determinant to compute a robust estimation of the covariance matrix for a point neighborhood, on which they applied PCA to obtain the normal. Larter, in [NWB15], they propose selecting the maximum consistent subset of the neighborhood's points $\mathcal{N}_r(\mathbf{p})$, for each point $\mathbf{p}$ in the point cloud. The consistency is evaluated by computing robust versions of the z-score and the Mahalanobis distance. This

subset is then used to estimate a best-fit-plane robust to noise and outliers. A more straightforward method to deal with outliers by Liying et al. [LY15] randomly draws triplets of points and computes the residual distances from the plane defined by these and the rest of the points in $\mathcal{N}_r(\mathbf{p})$. The process is repeated multiple times, and the residual distances are accumulated for each point. In the end, points with the highest accumulated values are discarded, and the normal is computed using PCA on the remaining ones.

Continuing with another method designed to be robust against outliers, Khaloo and Lattanzi [KL17] proposed using a deterministic MM-estimator to obtain robust estimations for the local mean and the covariance matrix on the k-neighborhood. Then, the squared Mahalanobis distance is used to classify points into inliers and outliers. The normal for each point is computed using PCA on the inliers, and later they used this to segment the cloud.

Cao et al. [CCZ+18] introduced the idea of shifting the neighborhoods, namely centering the neighborhood $\mathcal{N}_r(\mathbf{p})$ slightly away from $\mathbf{p}$ to avoid considering points from across an edge. For this, they use the surface variance $sv^k_{\mathbf{p}}$ to determine whether a point is on an edge, a corner, or a flat surface and apply a different shift accordingly.

The method presented by Sanchez et al. [SDC+20] is closely related to Castillo et al. [CLZ13], however instead of performing a global non-linear optimization, they choose to perform a piece-wise linear optimization which can be solved using traditional PCA. More specifically they define the minimization problem:

$$\begin{aligned} &\underset{\mathbf{n_p}}{\operatorname{argmin}} \left(\sum_{\mathbf{p}_i \in \mathcal{N}_k(\mathbf{p})} \left(w_{\mathbf{p}_i} (\mathbf{n}_{\mathbf{p}}^T (\mathbf{p}_i - \mathbf{p})) \right)^2 \right) \\ &w_{\mathbf{p}_i} = \frac{\mu}{\mu + \mathbf{n}_{\mathbf{p}}^T (\mathbf{p}_i - \mathbf{p})} \\ &\text{subject to } ||\mathbf{n_p}|| = 1 \end{aligned} \tag{3.11}$$

where μ is a constant. Weights and normals are updated alternatively during the optimization step until convergence conditions are met.

Another family of approaches works using a Voronoi partition of the space instead of numerical approximation. First introduced by Amenta et al. [AB99], these methods use the Voronoi diagram to estimate the normals of a point cloud by finding the *pole* of each point. This is the farthest Voronoi vertex that belongs

to the cell of the point. The normal $\mathbf{n_p}$ can then be estimated using the direction of the line that joins $\mathbf{p}$ with its *pole*. Dey et al. [DLS05; DS06] argue that this method does not work when points are noisy and study using *big* Delaunay balls instead of polar balls in these cases.

Alliez et al. [ACT+07] combine the good qualities of the Voronoi-based methods with the PCA approach's robustness against noise. They also construct the Voronoi diagram of the input cloud but add auxiliary points around a large bounding sphere to prevent cells from extending to infinity. Then, the normal estimate of each point is computed by PCA on its Voronoi cell's covariance matrix. However, instead of picking the eigenvector with the smallest eigenvalue, they pick the one with the largest eigenvalue. This is because Voronoi cells are elongated along the normal directions. For noisy samples, they compute the covariance of the union of multiple Voronoi cells, which increases robustness. Mérigot et al. [MOG11] reformulate this method by changing the integration domain. Their new method follows a convolutional approach to estimate the Voronoi Covariance Measure (VCM). However, even if they use this to detect sharp edges through eigenanalysis, the estimated normals are smoothed. Another shortcoming is that this method requires careful tuning of its parameters.

The major disadvantage of Voronoi-based methods is that the Voronoi diagram construction consumes many resources. Namely, they take a long time to compute compared to PCA-based methods. Next, we introduce another family of approaches that uses RANSAC-like schemes. By repeatedly randomly drawing triplets of points from a point's neighborhood, they can detect the predominant orientations.

Li et al. [LSK+10] propose a two-step approach to filter outliers. First, the noise scale is estimated by randomly drawing triplets of points and finding the plane that minimizes the residual distances to the neighborhood's points. Second, the noise scale constrains the candidate planes' search space to those close to the query point. Using this scale also allows classifying points as inliers and outliers, and, finally, the plane with the smallest residual distance to the inliers is selected.

Boulch et al. [BM12] introduce a voting scheme based on a Hough accumulator to preserve sharp edges. Triplets of points randomly draw from a given neighborhood, compute the plane going through them and cast a vote using the raw normal. Still, this technique requires high point density near sharp features to produce reliable results, and it highly depends on the correct choice of parameters in order to be able to smooth out the noise. Later they modify this method [BM16] by substituting the voting scheme with a Neural Network. Instead of picking the normal with most votes, they feed a two dimensional Hough

accumulator into a Convolutional Neural Network that is tasked to predict the final normal vector. However, CNNs tend to learn to predict the average results that minimize their loss functions. Due to this nature, predicting sharp normals is a difficult task for this method. Moreover, they strongly rely on the availability of ground truth data, which is scarce for LiDAR normals.

Zhao et al. [ZPL+19] present a method closely related to Li et al. [LSK+10]. They also randomly draw triplets and select the plane that maximizes the inliers. However, instead of using the noise scale, they use a user-defined parameter as a threshold. They also propose a top-down refinement where points are inserted into an octree, which is processed recursively. For each cell, they consider all the points from their leaves and try to find a plane that fits them. If more than half of the points are found to be inliers, they are assigned to the selected plane, and the leaves are processed recursively, only considering the remaining unassigned points.

We have seen how most methods perform some kind of plane-fitting in order to estimate the normal vectors. Some authors explore fitting more complex surfaces to the local neighborhood to better adapt to the underlying surface. Cazals et al. [CP05] propose using quadric jets (truncated Taylor expansion), Guennebaud et al. [GG07] propose fitting spheres and Campos et al. [CGA+13] propose fitting d-dimensional splats (d-jets). Notice that normal vectors can be estimated using the normals of the fitted surfaces. However, all these surfaces are smooth and cannot capture sharp edges.

3.5 POINT CLOUD SIMPLIFICATION

Strategies for point cloud simplification in the literature can mostly be grouped into two classes. The first one consists of *re-sampling* [PGK02; WK04; DYK07; MPF09]. Namely, computing a set of different points representing the same underlying surface as the original one but with a smaller number of points. Methods in the second group perform *sub-sampling* [Lin01; ABC+01; SF09; CCS12; Yuk15; Sch16; QHG19]. Namely, finding a criterion to select a given number of points from the original set.

Re-sampling Wu et al. [WK04] introduced a method that optimizes the output cloud explicitly for splat rendering. Points, represented as splats, are greedily selected to ensure maximum coverage of the surface and, then, a global optimization step refines their position. Instead, Miao et al. [MPF09] studied how to produce non-uniform samplings so that more samples are placed near

high curvature regions. This is achieved through adaptive hierarchical mean-shift clustering.

Pauly et al. [PGK02] study how to transfer several mesh-based simplification methods to point clouds. Specifically, they propose using clustering, iterative decimation through quadric error metrics, and particle simulation. Du et al. [DYK07] also proposed a method based on quadric error metrics. They first divide the cloud into cells, and, for each one, they compute the average representative and PCA normal. After this, they use surface variation [PGK02] to find the boundaries of the model while using point-pair contraction with quadric error to simplify planar regions further.

Lipman et al. [LCL+07] introduced their parameterization-free Local Projection Operator (LOP), which can project a new cloud onto the local multivariate median of a noisy one. The projected point can be considered a re-sampling of the noisy cloud. The process does not require normals and tries to ensure uniformly distributed points. Later, Huang et al. [HLZ+09] extended this method to make it aware of non-evenly distributed input clouds. These algorithms are also known as *point cloud consolidation* methods. Han et al. [HJW+17] give an extensive review of these and other filtering methods for point clouds. One shortcoming of the LOP is that it is an isotropic operator. Hence it assumes that the underlying surface is smooth and will not preserve sharp features [HWG+13].

In general, re-sampling methods usually produce results smoother than the original clouds. These may be useful in order to reduce the noise level but may also remove geometric features. Notice that most methods rely on normals, which have been computed on the noisy raw data. Which makes this family of methods less reliable to preserve finer geometric detail.

Huang et al. [HWG+13] also extended its LOP operator to handle sharp edges. They first estimate normals using the traditional PCA approach [HDD+92] and use these to estimate the location of edges. Afterward, the cloud is re-sampled away from these edges. The re-sampled cloud has reliable normals but also has gaps near the edges. Hence, on a second step, the cloud is iteratively up-sampled, towards the edges, to fill these gaps.

The generation of randomized distributions by sampling a surface is a closely related problem. Most techniques are based on Blue-noise sampling to generate randomly placed points that are approximately evenly spaced. Yan et al. [YGW+15] survey Blue-noise sampling methods, including Poisson-disk sampling and relaxation-based approaches. Farthest-point optimization techniques [SHD11; YGJ+14] also attempt to maximize the minimal distance in a point cloud as we do, but in the context of surface re-sampling rather than cloud sub-

sampling. Yan et al. [YGJ+14] report that farthest-point optimization generates point sets with excellent blue-noise properties for surface sampling.

Sub-sampling Linsen [Lin01] proposes ranking points using multiple descriptors that account for non-planarity, non-uniformity of the surface, and normal variation. Points are inserted into a priority queue and are iteratively removed until the target number is reached. A similar and contemporary work by Alexa et al. [ABC+01] ranks points based on the distance from a point to its moving least squares (MLS) projection. Then, those with the largest distances are iteratively removed. These methods aim at removing noisy and redundant points from high-density areas. However, they rely on the assumption that the underlying surface is smooth.

Song et al. [SF09] present a feature-preserving simplification algorithm that first determines whether a point belongs to an edge or not. Then it removes non-edge points based on a sorting criterion similar to Linsen's [Lin01] non-planarity score.

A drawback of the previous methods is that they may produce uneven sampling distributions. Qi et al. [QHG19] present a method that leverages feature preservation and density uniformity on the simplified clouds. They approach the problem by applying graph signal processing to point clouds represented as graphs. For this, they encode the clouds into adjacency matrices using k-nearest neighborhood adjacency.

Instead of decimation, Moenning et al. [MD03] presented a point cloud simplification algorithm, which, starting from an initial random subset of points, iteratively adds new points until a target density is reached (refinement). The algorithm selects the point which is "farthest" from the rest, using intrinsic geodesic Voronoi diagrams.

Finally, Yuksel [Yuk15] proposed an algorithm for generating Poisson disk samples by iteratively removing points. Like other sub-sampling methods, they define ranking criteria based on a point's distance to its neighbors. Intuitively, by removing points with many close neighbors, they end up minimizing the maximum point density.

A common limitation of most of the methods above is that they are not designed to work out-of-core (except Du et al. [DYK07]). While implementing clustering in an out-of-core fashion is straightforward, it is less obvious how to do the same for those requiring a consistent global ordering (priority queue) or massive data structures (adjacency matrices). Hence, to the best of our knowledge, no method has been proposed so far for decimating clouds with billions of

points while preserving the discussed properties.

A related approach by Corsini et al. [CCS12] generates Blue-noise samplings of meshes and can be easily adapted to a point cloud sub-sampling algorithm. The method randomly selects samples while removing those within a given radius. This produces good uniform distributions, and it seems amenable for out-of-core implementation.

Potree [Sch16] is a web-based application that provides tools for interactively inspecting point clouds. It is based on an adapted modifiable nested octree stored out-of-core. Points are distributed across the octree nodes and, at each node, a specific point spacing is guaranteed. The root node is first assigned a user-defined spacing, and for each subsequent level, this spacing is halved. The construction algorithm inserts points one-by-one. A given node is divided upon reaching a certain number of points. From these points, an approximate Poisson disk sample is generated and stored at the node, and the remaining points are assigned to its children. Poisson disk samples are constructed by dividing the node into a grid where each cell can only contain one point. A point is assigned to a cell if the cell is empty and respects the node spacing with adjacent cells' points. Otherwise, this point is assigned to the node's children. The advantage of this approach is that it builds multiple Poisson disk samples with different spacings simultaneously. However, as noted by the author, the spacing between points in different nodes is not guaranteed, and sometimes point stripes and holes may appear.

Two common limitations of [CCS12] and [Sch16] are that they do not allow for direct control of the output amount of samples, and they do not implement any kind of feature-preserving strategy.

3.6 EDITING POINT CLOUDS

Editing 3D data is usually a much more complicated task than image edition. Meshes and point clouds require consistent parameterizations in order to ease this task. However, one advantage of point clouds over meshes is that the lack of topology allows their restructuring without needing to take care of manifold conditions [KB04]. Here we provide a summary of the most popular edition tools for point clouds.

Pauly et al. [PKK+03] presented a free-form shape modeling framework for point-based representations. To exploit the advantages of implicit and parametric surface models, they define a proxy geometry that mixes unstructured point

clouds with the implicit surface definition of the moving least-squares approximation. Notably, they can support Boolean operations and free-form deformations.

Zwicker et al. [ZPK+02] presented a tool called Pointshop 3D, probably the most relevant example of point edition software. This system allows for efficient point cloud 3D-appearance and shape edition. It works on top of a parameterization and a dynamic re-sampling scheme based on a continuous reconstruction of the model surface.

More recently, Calderon et al. [CB14] introduced a complete framework for the morphological analysis of point clouds. Their central idea is simulating dilations and erosions without the need for any topological information. To achieve this, they devise a new model for the structuring elements based on a signed scalar field representation and replace the Minkowski sum operator with a new projection procedure.

Finally, one popular tool for geometry processing is Meshlab [CCC+08]. Although most of its utilities are designed to work on meshes, many tasks can be performed on top of point clouds, such as trimming, registration, surface reconstruction, or projective texturing.

3.7 SURFACE RECONSTRUCTION ON POINT CLOUDS

Substantial progress has been made in surface reconstruction from point clouds as extensively presented by Berger [BTS+14; BTS+17]. We have broadly reviewed normal estimation methods, which are crucial to generate a robust reconstruction. Here we focus only on the most relevant methods in the literature.

A standard procedure for this task is estimating normals using some algorithm (e.g., Hoppe [HDD+92]), consistently orienting them, and estimating the signed geometric distance towards the unknown surface. Then, a variant of the marching cubes algorithm can build a mesh following this distance field. One of the principal challenges these algorithms face is noise present through the scanned point clouds, which may not always be uniform.

Curless et al. [CL96] introduced a surface reconstruction algorithm from range images, which factors the noise level. They propose doing this by associated with each point a confidence value depending on the scanning technology. In particular, they associate lower confidence values at higher scanning grazing angles. Next, the different range images with associated distance and confidence functions are integrated into a voxelization. Each range image is tessellated by

constructing triangles from its nearest neighbors on the sampled surface. Tessellation is avoided over abrupt discontinuities by discarding triangles with edge lengths that exceed a threshold. Then, for each voxel, a ray is cast from the sensor along the sensor-voxel direction. If an intersection with the triangle mesh is found, the grid is updated accordingly. Each voxel is classified as unseen, empty, or near the surface. Frontiers between unseen and empty regions delimit holes in the surfaces. Finally, marching cubes is used to reconstruct the geometry as the zero-level isosurface.

We have also discussed Alexa et al. [ABC+01; ABC+03] moving least squares (MLS) surfaces for point-based methods and Guennebaud [GG07] improvement by fitting spheres for shape approximation. These implicit surface definitions can also be used to generate the distance field needed to extract the isosurface.

As noise is always present and not necessarily uniform through the scanned point clouds, it is necessary to factor it into the reconstruction algorithm. Giraudot et al. [GCA13] tackled this problem by estimating a noise-adaptive robust distance function, which was used to reconstruct the underlying surface. However, they do not consider that some points might be more reliable than others (e.g., in a point cloud formed by a mixture of registered scans) and treat all the points within a region equally.

Fuhrmann et al. [FG14] introduced a new method for surface reconstruction, which main novelty relies on the use of scale cues associated with the oriented sample points. The scale of a point is the finite surface area the point represents. Their method estimates an implicit function using the weighted sum of a set of basis functions parameterized by each sample point. Finally, the underlying surface is recovered as the zero set of this implicit function.

Poisson Surface Reconstruction [KBH06] and Screened Poisson Surface Reconstruction [KH13] are probably the most popular and widely-known surface reconstruction methods. They compute an implicit function that tries to ensure that its gradient is aligned with the normal vector at each point. This function indicates whether an arbitrary 3D point is inside or outside the surface. The Screened version adds additional constraints to avoid over-smoothing.

Finally, some methods simplify reconstruction for urban models and human-made scenes by assuming a small set of planes can approximate the scene. These include the already discussed methods from Nan et al. [NW17], Holzman et al. [HMF+18], and Bodis et al. [BRV15]. Another method by Monszpart et al. [MMB+15] optimizes a small regular arrangement of planes to fit the raw points. This is achieved by minimizing a cost function that balances the fitting to the data and the arrangement's regularity. Regularity is measured by introducing

relations such as parallelism, coplanarity, orthogonality, and symmetry between planes.

3.8 VISUALIZATION OF POINT CLOUDS

Zwicker et al. [ZPB+01] proposed one of the first rendering approaches for point clouds. Their method renders one oriented surface splat for each point. The mutual overlap between splats in object-space guarantees a hole-free rendering in image-space; however, a naive approach might cause shading discontinuities. Therefore they introduced a high-quality anisotropic anti-aliasing method based on the Elliptical Weighted Average (EWA) filter. They assign each splat to a different radially symmetric Gaussian filter kernel. These functions allow reconstruction of a continuous surface signal in object-space by performing a weighted average of the splat data. The main shortcoming of this method is that it was implemented entirely in the CPU, limiting real-time interactivity.

In contrast, Rosenthal and Linsen [RL08] presented a method that performs the filtering point properties (color, depth, and normal) after rendering them. This process fills any holes in the final visualization and can potentially be implemented in GPU. They claim to obtain results that are robust enough to detect model edges and silhouettes.

Preiner et al. [PJW12] devised another point cloud visualization approach that dynamically generates splats on the GPU. This method's core is a quick GPU-based screen-space k-nearest-neighbors search, which allows computing local tangent planes. They claim they can visualize scenes with about 10 million points. However, current technology can easily capture point clouds with one or two orders of magnitude more points.

LiDAR point clouds cannot be visualized instantly since different algorithms assume specific point densities or the availability of normals associated with each point. As we have previously discussed, sub-sampling the original clouds is usually required to render them using surface splatting optimally. In Section **3.5** we have already seen that the method by Wu and Kobbelt [WK04] optimizes the distribution of circular and elliptical splats to cover the rendered surface completely. Another method by Wimmer and Scheiblauer [WS06] supports massive point model visualization by combining two data structures. Memory-optimized point trees allow rendering point sets sequentially on the GPU, while nested octrees are used to manage the out-of-core data. Thus, enabling the direct exploration of point clouds without the need for post-processing. Potree [Sch16] is a web-based interactive tool for point rendering based on a version of modifiable

nested octrees. This structure stores Poisson disk samples of the original cloud with different spacings at each octree level. Nodes are rendered in a screen-projected-size order.

Another way of tackling massive point clouds is through visibility culling. Katz et al. [KTB07] introduced a projection operator named "Hidden Point Removal" (HPR), which can approximate each point's visibility to avoid rendering the occluded ones.

We have seen how some works focus on improving the efficiency of the rendering process. However, most authors have also developed out-of-core hierarchical strategies to tackle the massiveness of these point sets. One example is QSplat [RL00], which uses an out-of-core bounding sphere tree. At run time, the tree can be traversed to perform visibility queries and to generate point representatives at different level-of-detail. Follow-up approaches focus on further exploiting GPU capabilities for hierarchical point rendering [GM04; WBB+07; GEM+13]. Gobbetti at al. [GM04] introduced the Layered point clouds method. This visualizes the clouds by dynamically refining a set of multi-resolution blocks, guided by each point's contribution to the final render (measured in projected screen size). Later, Goswami et al. [GEM+13] proposed using multi-way kd-trees as the hierarchical structure for managing the point cloud information. All these approaches build their hierarchies bottom-up through a simplification process. Hierarchical LoD approaches can render arbitrary point clouds at interactive rates but do not benefit from the implicit 2.5D structure of point scans acquired from static LiDAR equipment.

One work that differs from this philosophy is presented by Schutz et al. [SMO+20]. Instead of building an out-of-core hierarchical structure, their methods progressively renders random subsets of points throughout multiple frames. If there is camera movement, the previous frame is reprojected onto the new view to take advantage of the already rendered information. This process is iterated until convergences or until it reaches the memory capacity of the GPU.

So far, we have seen methods designed to accurately reproduce the geometric details of point clouds at render time. However, they do not take into account other properties, such as color or normals. When using splats, most methods display uniform properties across their surface (e.g., uniform color). Consequently, if the point cloud had been previously simplified, the flat splat misses the original signal's frequency.

Some recent approaches use textures to display high-frequency detail across the splats' surfaces [SSL+13; BLM+18]. Nevertheless, their primary focus is on generating high-quality views and are not designed to provide interactive

experiences.

Local features extractors (e.g., SIFT) only tolerate small perspective changes when matching two images. Therefore, to localize a query image, a similar view must exist within the reference model. Sibbing et al. [SSL+13] study point-based rendering techniques to generate novel views which better match the query image. Their approach combines terrestrial LiDAR scanned data with extensive collections of registered images. Their principal contribution is the use of image completion techniques to fill holes and better preserve color gradients.

The approach by Bui et al. [BLM+18] also aims at the creation of high-quality novel views for localization. In particular, they train a deep neural network that transforms splat renders into a high-quality image. Unfortunately, their approach does not reach real-time performances, as they use a very deep generator network (about 80 layers). They claim that these high-resolution views could be used as textures for splats, but no further details are provided.

Yang et al. [YGW06] propose rendering points with huge splats and using view-dependent texture mapping to project high-resolution textures into them. However, multiple texture accesses and non-trivial weights are needed for blending the different textures for each splat. Similarly, Arikan et al. [APS+14] also try to produce high-quality renders from points clouds and a set of high-resolution photographs. However, instead of using splats, they generate meshes by representing the clouds as depth images. All input images are used to generate the color for each mesh. Since there is an overlap between the different meshes, they need to discern which fragment should be chosen for each screen pixel.

There is also a family of image-based rendering approaches that exploit the implicit 2.5D structure of point clouds that have been captured using LiDAR technology. These methods can vastly reduce the scene complexity while preserving good rendering quality. Another benefit is that this allows directly applying methods designed for image processing. For instance, we could reduce the geometric detail resolution using image downsampling, which is more straightforward and faster than point cloud simplification. Moreover, different feature information (such as color, normal, and depth) can also be stored at independent resolutions.

Panoramic images have been widely used to represent scenes. In particular, 360-degree images [Che95] generated using a plane-chart cylindrical projection are the most common. These allow looking around towards arbitrary directions. Okura et al. [OKY15] present a system that enables navigating through the scene by interpolating new views from panoramas re-sampled at grid points. In contrast, Benedetto et al. [DGB+14] generate panoramic views at specific points

and support moving between while displaying a pre-computed video sequence.

3.9 LEARNING ON POINT CLOUDS

With the advent of Deep Learning, we have seen an exponential surge of studies applying learning on raw unstructured points in recent years. We briefly provide a quick overview of the different families of methods in the literature. The reader can refer to existing surveys [GWH+19; ASS+18], and the comprehensive Siggraph tutorial by Mitra et al. [MKG+19] for more details.

Point clouds, meshes, and images are examples of unstructured data. However, learning from geometry is a much more challenging task than learning on images due to their irregular distribution in space. For images, the widely accepted paradigm is Convolutional Neural Networks, whereas multiples paradigms have been proposed for point clouds, but none has been widely accepted in the community.

Earlier works like VoxNet [MS15] and ShapeNets [WSK+15] try to apply the convolutional paradigm to 3D. The scene is divided into a regular grid, and occupancy is computed for each cell. Then 3D convolutions are applied to generate the desired result. While this may be a feasible object classification approach, other tasks such as shape generation and fine segmentation are prohibitive since the voxelization must be kept very coarse due to memory and performance requirements.

Other works tried to benefit from the convolutional approach by representing the 3D geometry using a set of 2D image-like representations. Su et al. [SMK+15] propose learning to predict object classes by feeding the network with a set of rendered views from a different point. Later, Tatarchenko et al. [TPK+18] proposed predicting point properties or segmenting a cloud by learning on an image constructed by projecting the neighborhood of a point into a local frame.

More recently, other approaches define convolutional approaches on the continuous space rather than on discrete grids. Hermosilla et al. [HRV+18] introduce the Monte Carlo Convolutions, which approximate a 3D convolutional kernel by Monte Carlo integration. They use Poisson Disk sampling to construct the hierarchy of points. A contemporary work by Atzmon et al. [AML18] instead uses a radial basis function as convolution kernels.

Another completely different approach is learning on a graph built on top of a point cloud. Points are considered vertices, and the relations with their

closest neighbors become edges. Simonovsky et al. [SK17] propose learning on these representations using Edge Conditional Convolutions, designed explicitly for graphs.

PointNet [QSM+17] was the first method that directly consumes points and does not take any spatial relationship between them into account. The input to the neural network is an unordered list of the considered points. Authors propose using a set of symmetric functions on all inputs to become invariant to permutations on this list. Examples of these functions are the *maximum* or the *sum*). PointNet++ [QYS+17] is later introduced to be able to learn local information. PointNet layers are applied to the cloud segments, and their results are aggregated hierarchically. However, these two networks were designed to learn on the global cloud. Hence Guerrero et al. [GKO+18] introduce the PCPNet, an improvement over PointNet, which is designed to learn better and predict the neighborhood's local features of a point.

4

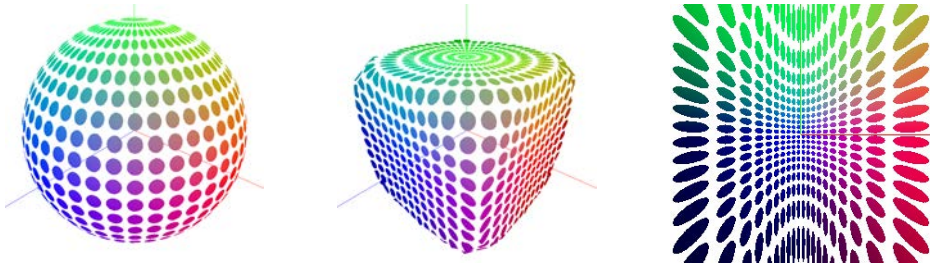
Algorithms for the Improvement of Light Detection and Ranging Point Cloud Data

Nowadays, the use of terrestrial stationary LiDAR devices has become frequent in many scenarios due to the excellent trade-off they provide between cost and scanning quality. Out of all the available 3D scanning technologies, these provide the highest accuracy and sample density, making them especially suitable for digitizing buildings.

Most LiDAR devices use a rotating head and a rotating mirror to capture 360° data. The rotating mirror deflects an infrared laser beam at regular intervals in the vertical direction, whereas the rotating base allows for horizontal sweeping. The device measures the time the beam takes to bounce into a surface and return to the sensor to determine its distance. Nevertheless, the number of samples for each scan line is usually constant. Because of this, a much higher sample density is obtained around the poles than across the equator (Figure 4.1).

Another shortcoming of this strategy is that a single scan cannot wholly capture a scene with complex geometry due to occlusions. In a typical scenario, multiple scans of the same scene must be taken by placing the scanner at different locations. This improves the overall surface coverage.

However, this introduces yet another challenge. The expected range error for a point usually depends on its distance to the sensor, and the scanned materials' reflectance properties (Table 4.1). Consequently, when mixing points scanned from multiple locations, we are also mixing different noise distributions. For instance, for dark surfaces, the range noise might have the same order of magni-



(1) **(2)** **(3)**

Figure 4.1: Illustrating the distribution of samples captured by a terrestrial LiDAR device. In **(1)** we illustrate how samples are generated at regular intervals of the polar angles (θ, ψ). This sampling strategy produces very irregular point densities. In **(2)** we simulate how this distribution would be projected on a cuboid. We can see that on the poles we find the highest concentration of points in detriment of regions close to the equator. Nevertheless, it is on the latter (vertical walls) where the distribution is mostly uniform, if we only consider the central region. Instead, in **(3)** we show that, on an orthogonal wall whose center is orthogonal to the beam direction, as we move away from the center, the surface becomes more tangent to the ray direction and the spacing between samples increases.

tude as the point spacing. In contrast, for close surfaces, this noise may be much smaller.

P20	Black (10%)	Gray (28%)	White (100%)
10m	0.8 mm	0.5 mm	0.4 mm
25m	1.0 mm	0.6 mm	0.5 mm
50m	2.8 mm	1.1 mm	0.7 mm
100m	9.0 mm	4.3 mm	1.5 mm

RTC360	White (89%)
10m	0.4mm
20m	0.5mm

Table 4.1: Range noise (RMS) as a function of depth and material reflectivity for two high-end pulsed laser scanners. *Source: Leica ScanStation P20 Product Specifications.*

In summary, clouds captured using terrestrial LiDAR stations may present highly uneven sampling distributions, missing areas due to occlusion, and mixtures of points with different range noise levels. In this Chapter, we study how to overcome these difficulties to produce useful representations that can be used for tasks such as real-time inspection or surface reconstruction. The rest of the Chapter is organized as follows:

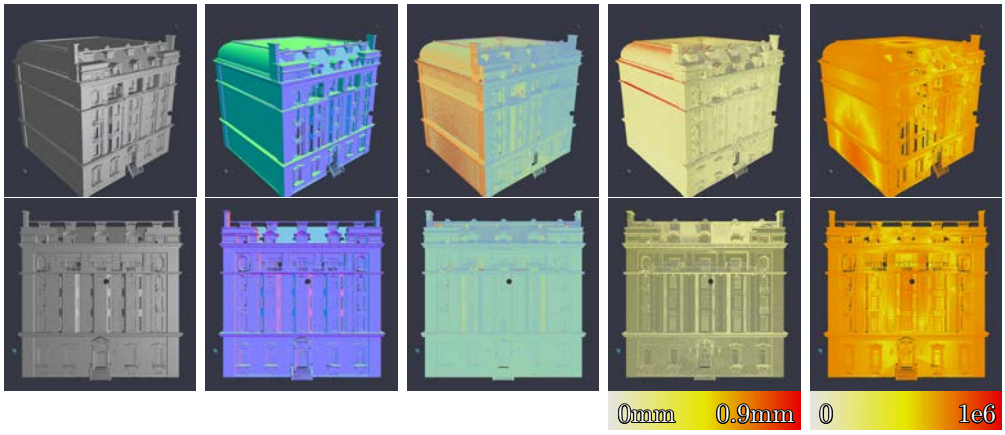
1 **Effective Simplification of Point Clouds:** LiDAR point clouds captured using terrestrial devices are usually massive and very unevenly distributed, i.e., over-sampled and redundant in some areas and under-sampled in others. In Section 4.1, we study the properties a good simplification algorithm must have and propose a method based on a greedy iterative decimation.

2 **Sensor-aware Normal Estimation:** Normal vectors are an essential piece of information for multiple applications related to point clouds, such a rendering, surface reconstruction, 3D printing, segmentation, or simplification. Indeed, as we will see later, robust normals are a valuable piece of information that can be used in the decimation process (Section 4.1). Hence, in Section 4.2, we study the challenges related to point clouds (e.g., uneven point distribution, a mixture of noise levels) and propose an adaptive normal estimation algorithm for raw point cloud data.

3 **Stretch-invariant Panoramic Representation:** Depending on the decimation level, high-frequency information on the point clouds may be lost. In Section 4.3, we study how points properties can be encoded efficiently into textures, previous to simplification, to minimize storage space and enhance the detail at render time.

4.1 EFFECTIVE SIMPLIFICATION OF POINT CLOUDS

As we have seen, LiDAR data is composed of point clouds captured from multiple locations (Figure 4.2), and may have incomplete regions (Figure 4.31), mixtures of points with different noise levels (Figure 4.32) and very unevenly sampled regions (Figure 4.33). Moreover, these clouds can be massive with over a billion points, hence the need for simplification.



(1) Phong (2) Normal (3) Sensor (4) Error (5) Density

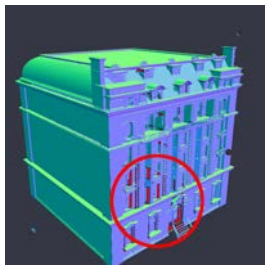
Figure 4.2: Global and frontal views of simulated LiDAR point cloud scanned from 13 different locations of the *mansion* model. (2) shows the color-coded ground truth normals whereas (3) shows the color-coded sensor IDs (location from which each point was captured). (4) shows the color-coded estimated range error using the *Leica P20* technical specifications. Finally, (5) shows the color-coded local point density deviation with respect to the average density, using a logarithmic scale.

Taking into account these characteristics, we formulate the following desired properties for a point cloud simplification algorithm:

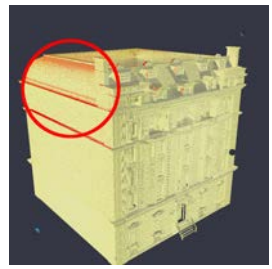
- **Scalability:** As a consequence of the massiveness of the input point clouds, the designed algorithm must be able to process inputs of any size.

  Hence, we propose a voxelized approach that maintains voxels of equal size out-of-core and processes them independently while guaranteeing a nearly-optimal global result. As discussed later, this approach also allows lazy-updating of the cost functions involved in the optimization, speeding up the process.

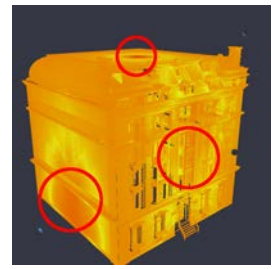
- **Adaptability:** Considering individual scans, surfaces visible from the sensor location result in unevenly distributed samples due to multiple reasons.



(1) Disocclusions



(2) Mixture of Noise Levels



(3) Uneven Densities

Figure 4.3: Displaying typical problems in LiDAR point clouds. In **(1)** we show *red* holes in the cloud in disoccluded areas. In **(2)** we show areas where nearby points, captured from different locations, have very different noise levels. In **(3)** we show regions where the local density deviates significantly from the mean (*red*-colored regions).

Some factors are intrinsic to the surface being captured. For example, surface patches with challenging materials (e.g., mirror-reflective) are poorly captured. Nevertheless, the major sources of uneven sampling are extrinsic factors, namely surface orientation and distance to the sensor [HWS16]. Sampling spacing on a surface would be inversely proportional to sensor distance and the cosine of the incident angle. Moreover, as we have seen, data is not usually captured at regular solid angles (Figure **4.1**). Consequently, during simplification, the designed algorithm must remove the redundancy in over-sampled regions while preserving the points on sparsely-sampled ones.

Hence, we propose an approach that minimizes the maximum local density by minimizing the maximum distance between the two closest points. As we will see later, this approach is more efficient than computing densities of a fixed radius and yields competitive results.

- **Faithfulness:** Strategies for point cloud simplification in the literature can mostly be grouped into *re-sampling* and *sub-sampling*. *Re-sampling* consists of computing a set of different points that represent the same underlying surface as the original one but with a smaller number of points. In contrast, *sub-sampling* consists of finding a criterion to select some points from the original set.

  We adopt a sub-sampling approach because this avoids attribute (e.g., color, position) interpolation at new samples, leading to more accurate results. Manufacturers of high-end LiDAR equipment put significant efforts to provide error guarantees, which allow their devices to provide usable data in scenarios where such guarantees are critical, e.g., forensic appli-

cations [Wal15]. In some countries, a dataset without a complete known error rate would not be considered evidence [Wal15].

- **Efficiency:** Due to the massiveness of the input point clouds, the designed algorithm must work out-of-core, which has a significant impact on performance.

  Hence, we studied other possible performance bottlenecks and found out that finding a suitable search radius for *radius*-nearest-neighbors search is not possible for highly unevenly-sampled data. If too large, heavily oversampled areas will kill performance; if too small, the neighborhood will be empty for most parts of the cloud. Therefore, these methods do not scale well in real-world scanned datasets. Instead, we propose an approach that works using k-nearest-neighbors searches with $k = 1$, which significantly speeds up the process.

- **Progressiveness:** Different computer graphics techniques benefit from having access to different levels-of-detail for a given representation.

  Hence, we propose ranking the input points according to a criterion based on a point's distance to its closest neighbor, which allows us to decimate the cloud iteratively. This way, we have control over the final amount of points, and, if desired, intermediate representations can be stored as different levels-of-detail.

- **Feature-Awareness:** Different regions on the cloud may have different curvature properties. Nearly-planar regions can be very-well approximated with few points, whereas salient regions need much more information to recover high-frequency features.
  Hence, we propose a complementary strategy to our algorithm by incorporating feature-aware information, allowing altering the final distribution of points to obtain higher density in salient regions.

- **Noise-Awareness:** Some regions of the cloud may have points with very different noise levels. While we want to stay faithful to the error guarantees in the raw data (i.e., avoid smoothing), we can decrease the overall noise-level by favoring the preservation of points with a lower error.

  Hence, we propose a complementary strategy to our algorithm by incorporating error-aware information, which will encourage the early decimation of noisy points in areas where they are redundant while keeping them in regions where no other information is available.

Poisson-disk sampling approaches [Yuk15] seem to offer some of the desired properties (such as *adaptability*, *faithfulness* and *progressiveness*) for the selection of subsets with more even samples. This is accomplished by defining a ranking

criterion based on the distance of a point to its neighbors and then removing the desired ratio of samples with the highest rank. Intuitively, by removing points with many nearby neighbors, these approaches minimize the maximum point density. We improve on top of these by adding the proposed solutions.

4.1.1 Problem Formulation

Given a point cloud C and some user-provided decimation factor $\lambda \in (0, 1)$, most simplification algorithms compute a new point cloud C' such that $|C'| \leq \lambda|C|$ and the surfaces represented by C and C' are similar. *Re-sampling* methods compute points in C' as representative points of some neighborhood, thus optimizing their placement and computing their attributes (e.g., color) through some interpolation or averaging scheme. We want to avoid creating new samples. This can decrease accuracy since representative points might end far away from the underlying surface. Hence, in favor of being **faithful** to the raw data, we rather adopt a sub-sampling approach by constraining C' to be a subset of C. Notice that we are assuming low-noise samples (e.g., high-end LiDAR equipment report 0.4 mm RMS range noise at 10 m) since high-noise data would rather benefit from re-sampling.

In favor of being **adaptive** to uneven densities in the raw data, we wish the sub-sampling process to remove points in over-sampled regions while preserving samples in under-sampled areas. This is similar to require that local point densities in C' are as uniform as possible. More formally, given C and λ, we could compute C' as:

$$\begin{aligned} &\underset{C'}{\operatorname{argmin}} \ \max\{\rho_{\mathbf{p}} | \mathbf{p} \in C'\} \\ &\text{subject to } C' \subset C \\ &\qquad\qquad |C'| = \lambda|C| \end{aligned} \tag{4.1}$$

where $\rho_{\mathbf{p}}$ is some local density estimate at point $\mathbf{p} \in C'$. The equality constraint above assumes the user-provided decimation ratio λ has the form $n/|C|$ for some integer n.

Let us consider first local density estimates $\rho_{\mathbf{p}}$ based on points within some fixed distance r from $\mathbf{p} \in C'$. If $\mathcal{N}_r(\mathbf{p})$ denotes the neighbors around $\mathbf{p}$ within radius r, then $\rho_{\mathbf{p}}$ should be directly proportional to $|\mathcal{N}_r(\mathbf{p})|$ and inversely proportional to r, r^2 or r^3 (depending on whether density is measured along a line, on the underlying surface or inside a volume).

We can we rewrite the problem in Equation 4.1 by replacing density estimates by distances to the closest samples. Let $d_{\mathbf{p}}$ be the distance from $\mathbf{p} \in C'$ to its closest point in C', i.e:

$$d_{\mathbf{p}} = \min\{d(\mathbf{p}, \mathbf{p}') | \mathbf{p}' \in C'\} \tag{4.2}$$

where $d(\mathbf{p}, \mathbf{p}')$ is just the Euclidean distance between points $\mathbf{p}, \mathbf{p}'$. Then the problem can be formulated as:

$$\begin{aligned} &\underset{C'}{\text{argmax}} \ \min\{d_{\mathbf{p}} | \mathbf{p} \in C'\} \\ &\text{subject to } C' \subset C \\ &\qquad\qquad |C'| = \lambda |C| \end{aligned} \tag{4.3}$$

Namely, we assume that minimizing the maximum density is the same as maximizing the minimum distance between two closest points. This problem is also known as finding the Poisson disk sample with a maximal radius.

4.1.2 Computing Per-sample Costs

In favor of **progressively** decimating the cloud, we want to implement an approach similar to Yuksel [Yuk15]. Their algorithm consists of an iterative greedy procedure that approaches the problems above by first sorting all the points according to a cost function. Then, points with the highest cost are iteratively removed from the original set while updating the affected points' cost. In particular, Yuksel defines the cost $w_{\mathbf{p}}$ of a point $\mathbf{p}$ as:

$$w_{\mathbf{p}} = \sum_{p' \in \mathcal{N}_r(\mathbf{p})} \left(1 - \frac{\hat{d}(\mathbf{p}, \mathbf{p}')}{r}\right)^{\alpha} \tag{4.4}$$

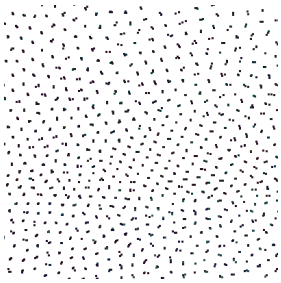
$$\hat{d}(\mathbf{p}, \mathbf{p}') = \begin{cases} d(\mathbf{p}, \mathbf{p}'), & \text{if } d(\mathbf{p}, \mathbf{p}') > r_{min} \\ r_{min} & \text{if } d(\mathbf{p}, \mathbf{p}') \leq r_{min} \end{cases} \tag{4.5}$$

$$r = 2\sqrt[3]{\frac{V}{4\sqrt{2}|C|}} \tag{4.6}$$

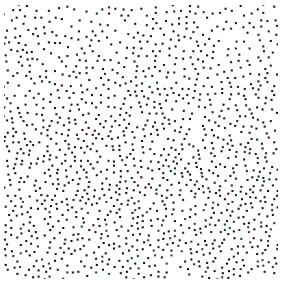
$$r_{min} = r\left(1 - \left(\frac{1}{\lambda}\right)^{\gamma}\right)\beta \tag{4.7}$$

where V is the volume of the sampling domain, $\alpha = 8$, $\gamma = 1.5$ and $\beta = 0.65$.

The authors observe that better results are obtained when "samples with many relatively close neighbors are removed earlier than samples with fewer but very close neighbors" and introduce r_{min} to induce this. However, as shown in Figure 4.4, this strategy tends to produce many tight clusters (3 to 7 points very close to each other).



(1) Yuksel [Yuk15].



(2) Our approach.

Figure 4.4: Yuksel [Yuk15]'s approach **(1)** produces small clusters, but these are well-distributed across the space. Our approach **(2)** maximizes the minimum separation between point samples.

Although Yuksel's method vastly improves local point densities, the algorithm does not scale to massive datasets with highly uneven point distributions. The performance bottleneck is radius-based neighbor searches. Despite using a kd-tree to speed-up such searches for highly unevenly distributed point clouds (such as LiDAR data), using a constant search radius to determine the neighborhoods proves fatal. Using a large radius on heavily-dense areas kills performance, as the number of points to retrieve will be huge. However, using a radius small-enough to make these points, treatable will instead lead to empty neighborhoods for the biggest part of the volume. Hence, in favor of **efficiency**, we propose an adaptive neighborhood based on a fixed amount of neighbors (k), namely a k-neighborhood.

Now, given a k-neighborhood, we wish to know the costs (Equation (4.4)) we would get with a radius-based neighborhood. We can do so if we use a local density estimate $\rho_{\mathbf{p}}$. We must differentiate between costs from points within r_{min} and from points between r_{min} and r. The former one is rather easy to compute:

$$w_{\mathbf{p}}^{r_{min}} = \rho_{\mathbf{p}} \frac{4}{3} \pi r_{min}^3 (1 - \frac{r_{min}}{r})^{\alpha} \tag{4.8}$$

For the latter, we would need to solve an integral as the contribution varies

as a function of the radius:

$$w_{\mathbf{p}}^{r} = \int_{r_{min}}^{r} \rho_{\mathbf{p}} 4\pi x^2 (1 - \frac{x}{r})^{\alpha} dx \tag{4.9}$$

Notice though that $w_{\mathbf{p}} = w_{\mathbf{p}}^{r_{min}} + w_{\mathbf{p}}^{r}$ can be written as $w_{\mathbf{p}} = K\rho_{\mathbf{p}}$. Since $w_{\mathbf{p}}$ is used to sort the points, we can neglect the value of K, which is constant for all the points. Thus, we can use a k-neighborhood to estimate the local densities and use this as a criterion to sort the points. This makes sense because by removing the points with the largest densities, we minimize the maximum density, which is the objective of our first problem formulation in Equation (4.1).

Using the local density as a sorting criterion still inherits one shortcoming from Yuksel [Yuk15]'s algorithm, as in the simplified point clouds, we still find small clusters. As we have shown, this method optimizes using a criterion closely related to density. This value does not carry enough information, i.e., it does not give any information about how points are distributed within the neighborhood. Optimizing for density does not guarantee uniformly distributed results because points could be very close to each other.

So, in favor of making the algorithm **adapt** better to the distribution of points, instead of optimizing for equal densities, we propose optimizing directly for the maximum distance to the closest point. This is equivalent to using a k-neighborhood with $k = 1$, namely computing the local point density as a value inversely proportional to the distance to the closest point. This is fact equivalent to our second problem formulation in Equation (4.1), which avoids clusters as shown in Figure **4.42**.

4.1.3 Updating Per-sample Costs

Similar to Yuksel [Yuk15], every time we remove a point, we should update its neighbors' cost. This can be achieved by keeping the points in a priority queue.

Using Yuksel's cost definition in Equation (4.4), cost updates are relatively straightforward. For each point $\mathbf{p}$, we can keep a list of all its neighbor points $\mathbf{p}'$. Upon removing $\mathbf{p}$, we can update the cost of every point $\mathbf{p}'$ in constant time and then update their position in the queue. When using density estimates on a k-neighborhood as cost values, we can also update the cost of neighboring points until we have updated the cost of a point k times. After that, we need to search the k-neighborhood again to update the density and keep the algorithm running.

Using density on a 1-neighborhood means we need to perform a nearest neighbor search for each point $\mathbf{p}'$ at each step of the algorithm, becoming a performance bottleneck. This is alleviated due to the strategy explained next.

4.1.4 Sub-sampling Algorithm

When handling point clouds with billions of points, maintaining a priority queue in memory as in [Yuk15] is not feasible. In favor of **scalability**, we have designed a strategy that works for voxelized out-of-core representation and can produce close-enough results.

We first generate a voxelization of the point cloud and store it out-of-core. In all our experiments with architectural models, voxels covered $10^3 m^3$. Then we iterate the following two steps until we reach the target number of points $\lambda|C|$:

1 **Cost updates**: We individually read the voxels from disk, and, for each point, we compute its cost $w_{\mathbf{p}}$ as the inverse of the distance to its closest neighbor. For correctness, this neighbor must also be searched in the surrounding 6, 18, or 26 voxels. For efficiency purposes, this search can be restricted to just the current voxel during the first iterations. We keep an updated max priority queue (global, not per-voxel) to keep the computed costs. As we shall see, in the queue, we only keep the $\lambda|C|$ points with the lowest cost. Points with a cost higher than those in the queue are not stored in the priority queue. This might seem counter-intuitive (since points with higher cost need to be removed first), but it is explained next.

2 **Point removal:** Now, at the top of the queue, we have an upper bound w_{up} of the cost the $\lambda|C|$-th sample will have. Thus, if we remove any point with a cost above w_{up}, we ensure that $\lambda|C|$ samples will remain. We individually read the voxels from the disk and remove any point whose cost is higher than w_{up}. However, we need to avoid removing any point whose closest neighbor has been removed in the current iteration since these points require a cost update. That is why we need to iterate the two steps.

These two steps are repeated until $|C'| = \lambda|C|$.

Thanks to this strategy, we avoid the need to recompute the cost of a point whose closest neighbor has been removed every time one point is removed. In favor of **efficiency**, costs are only recomputed once for each step.

If we assume that, on average, every point being removed invalidates the cost of another point, each iteration of the two steps above removes about one-half of the intended points. This means that iteration i removes about $(1-\lambda)/2^i|C|$ points and leaves about the same quantity of points for the next iterations. The implications of this behavior are discussed below.

4.1.5 Results and Discussion

We implemented the proposed sub-sampling algorithm using non-parallel C++ code. The program was forced to run out-of-core regardless of the model size, i.e., voxels were kept on disk for all test models. The test hardware was a commodity PC equipped with an Intel Core i7-4790K CPU, 32GB of RAM, and a 2TB Toshiba DT01ACA200 HDD running Ubuntu 20.04.

We tested the algorithm with both synthetic and real point clouds representing architectural models. The size of the point clouds varied from about 400K points to about 1.2 billion points. The number of scans within each point cloud also varied from 2 to 38 scans per dataset. Real models were taken using actual LiDAR equipment, including a Leica P20 ScanStation and Leica RTC 360.

For real datasets, scanner locations were restricted to be about 1.5 m above the ground (the scanner device was mounted on a tripod). In contrast, for synthetic datasets, we could test more varied scanner locations within the scanning volume.

Synthetic models

We first discuss the results with synthetic models. These models were simulated by imitating the sampling pattern and range noise distribution of a high-end LiDAR scanner (Leica P20). We generated such synthetic models because we could limit the scan number and scan resolution to get point clouds small enough to fit in core memory, thus enabling comparison with competing approaches.

Figures 4.5 to 4.7 shows and discusses these results. Even though point clouds combine multiple scans (blue spheres represent scan locations), the uneven distribution of samples is quite apparent (Figures 4.5 to 4.7-2). We have regions with clearly over-sampled areas due to sampling concentration around poles and under-sampled areas on surfaces (e.g., floors) roughly tangential to the beam direction. We used a color scale to convey local density variation further. In particular, we encode the deviation of the local density from the average density. In a perfectly uniformly distributed cloud, the local density should be uniform, and thus such deviation from the average should be zero (represented as white

color). However, LiDAR data exhibits large density deviations. We had to use a logarithmic scale to better represent local density variations across multiple orders of magnitude: *white* encodes 0, *yellow* encodes 10^3 and *red* encodes (10^6).

The output of our sub-sampling algorithm ($\lambda = 0.1$) is shown in Figures 4.5 to 4.7-3. The much more uniform local densities are apparent, considering the point distribution and the color uniformity (in these images, the nearly-white color also conveys the small local density deviations). The distribution of local density deviations is shown in Figures 4.5 to 4.7-4, for both the input point cloud and our sub-sampled results. Notice that the improvements span one to three orders of magnitude (curve color outline also encodes local density deviation).

We compared our algorithm with Yuksel [Yuk15]'s method and MeshLab's implementation of Corsini's method [CCS12], which is adapted to point clouds (the original method targets polygonal meshes). Figures 4.5 to 4.7(6-8) shows the distribution of distances from a point to its i-th closest neighbor. Considering only the closest neighbor would be unfair since Yuksel [Yuk15]'s method tends to produce some tight clusters, as illustrated in Figure 4.4 and confirmed in our experiments. Yuksel [Yuk15]'s and Corsini's methods lead to smaller distances to the closest neighbor, i.e., Poisson disk samples with smaller radii. Similarly, Yuksel's method also yields more irregular distributions (especially when the closest and second closest neighbors are considered). Although we maximized only for the minimum distance between samples, we can observe that distances to the i-th closest neighbors are regularly distributed, with curves for the i-th and the $(i + 1)$-th closest neighbors differing by a roughly constant shift. This means that neighbors are placed at regular intervals, which further confirms that our sub-sampling strategy produces uniformly distributed samples. Similarly, Corsini's curves are also quite well distributed. In our tests, their smaller distance to the closest neighbor can be explained by the fact that their method cannot get sufficiently close to a target decimation factor.

Running times for these algorithms are shown in Figures 4.5 to 4.7-5. We include running times for Corsini's method [CCS12] despite it does not guarantee (and rarely meets) the target number of samples. Our running times include reading and writing voxels from/to disk at each iteration, whereas Yuksel's and Corsini's methods run in-core. Despite this, in most cases, our algorithm is faster than Yuksel's for practical values of λ. For highly-uneven data, moderate decimation factors, e.g., $\lambda = 0.5$, prevent any sub-sampling algorithm from removing enough samples to get a reasonably uniform distribution. For LiDAR data, useful decimation ratios are below $0.1 - 0.2$. We also observed that the largest performance benefits occurred in those models with very dense regions, where radius searches become very inefficient. Corsini's approach was the fastest in all

tests, although we are not aware of any available out-of-core implementation.

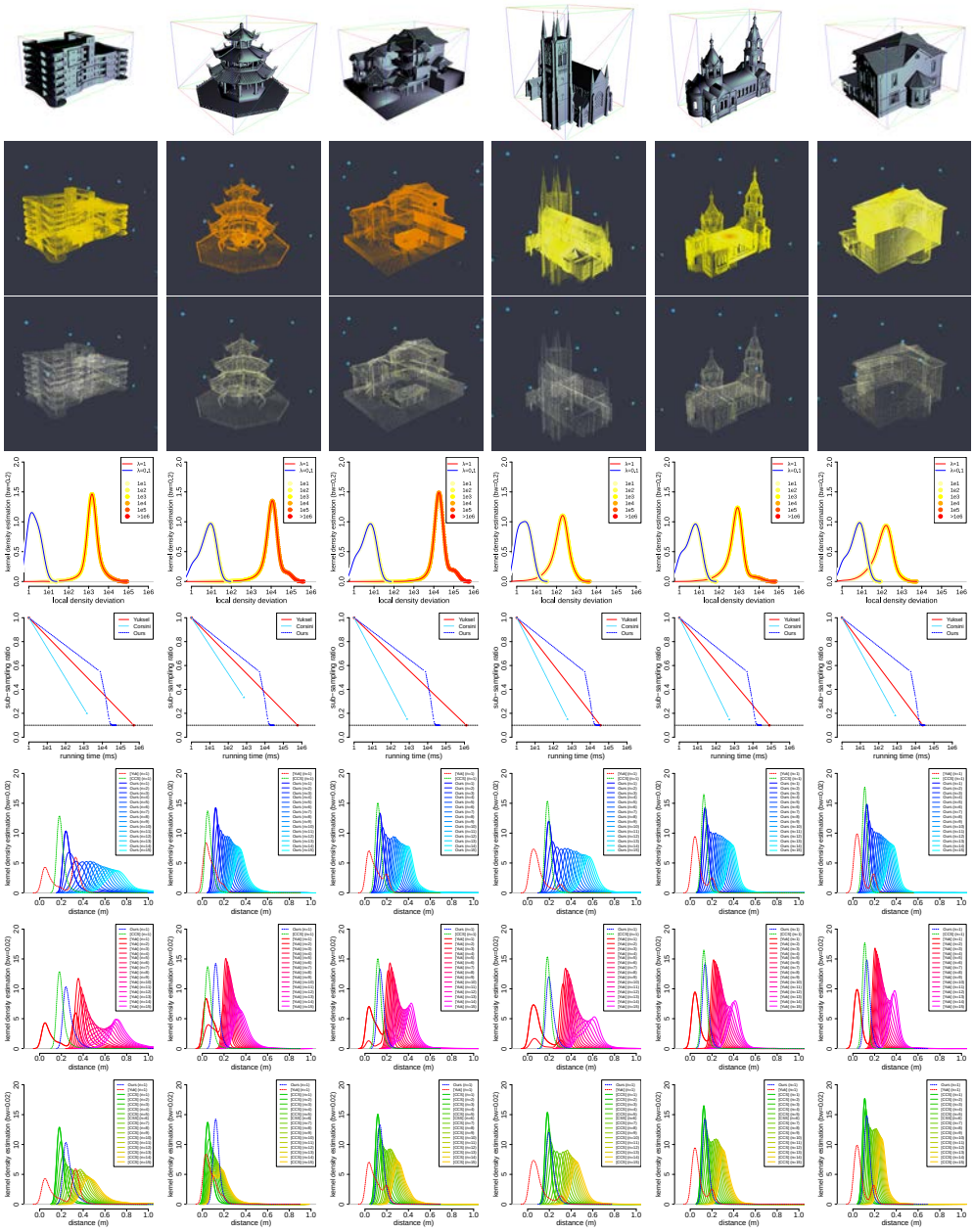


Figure 4.5: Evaluation on synthetic data. From top to bottom: **(1)** Input mesh. **(2)** Simulated clouds. **(3)** Simplified clouds. **(4)** Distribution of local density deviations. **(5)** Run times. **(6,7,8)** Distribution of distances to the closest point.

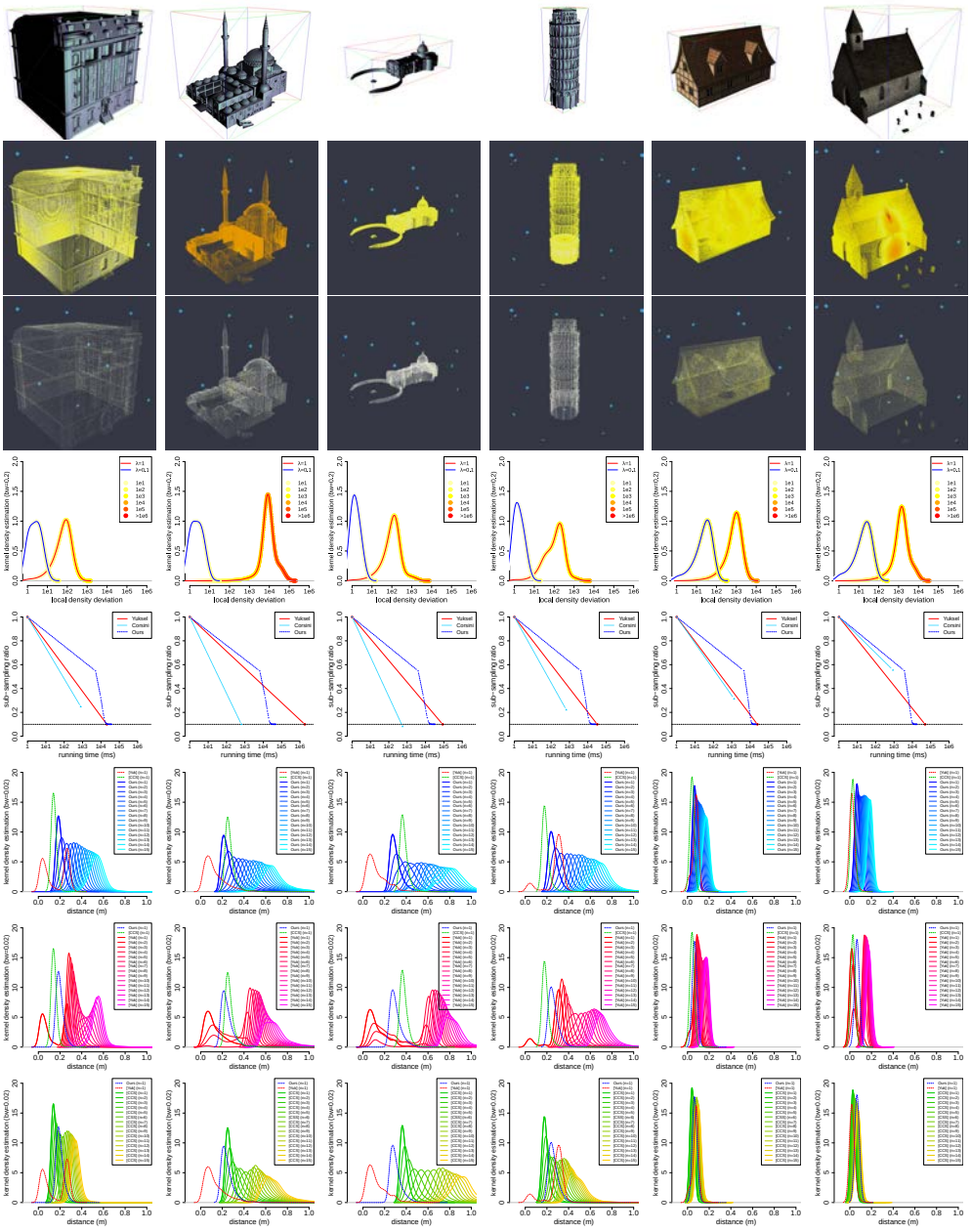


Figure 4.6: Evaluation on synthetic data. From top to bottom: **(1)** Input mesh. **(2)** Simulated clouds. **(3)** Simplified clouds. **(4)** Distribution of local density deviations. **(5)** Run times. **(6,7,8)** Distribution of distances to the closest point.

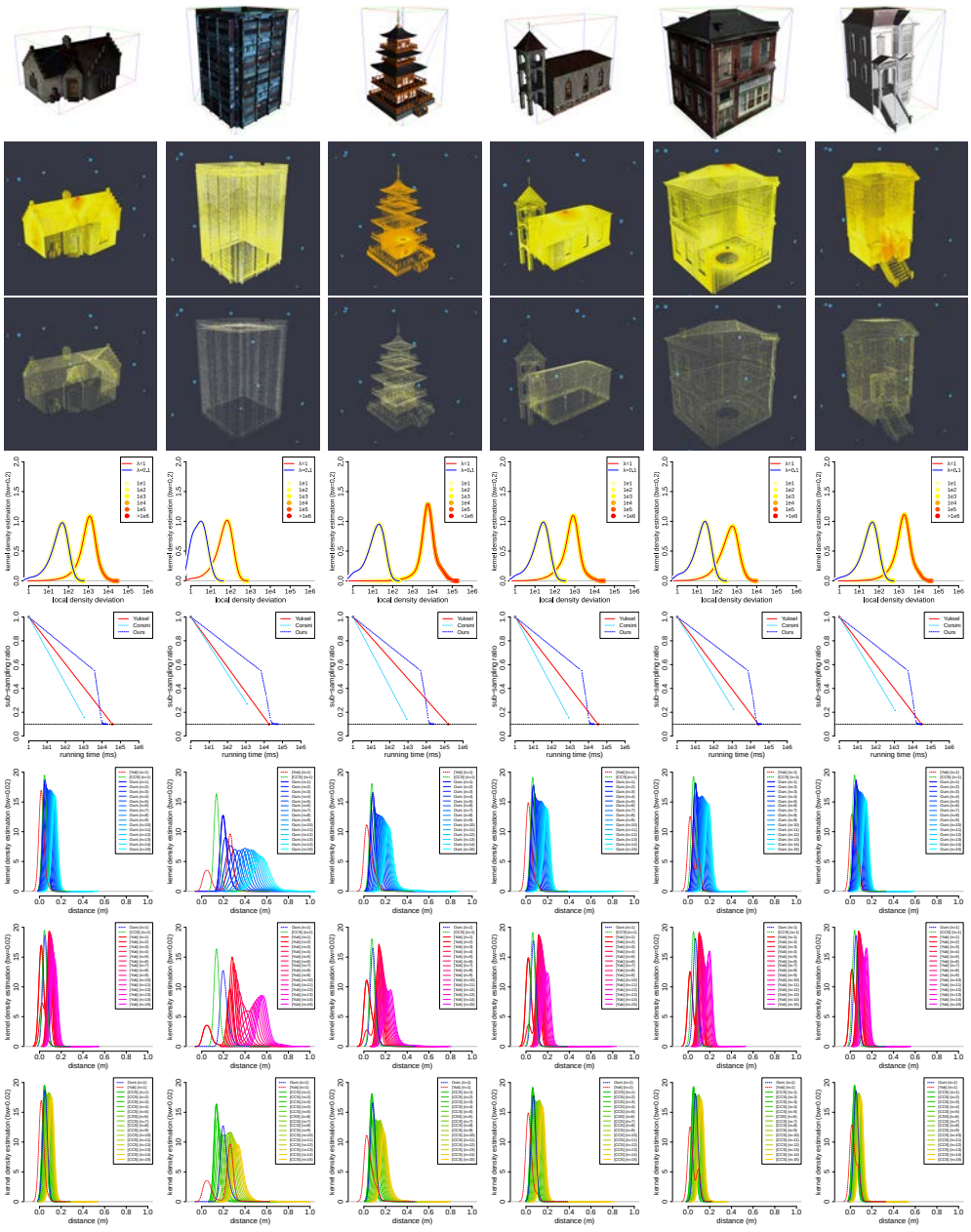


Figure 4.7: Evaluation on synthetic data. From top to bottom: **(1)** Input mesh. **(2)** Simulated clouds. **(3)** Simplified clouds. **(4)** Distribution of local density deviations. **(5)** Run times. **(6,7,8)** Distribution of distances to the closest point.

Real models

We now discuss the results with actual LiDAR data from high-end LiDAR scanners (Figure 4.8). These models were too large to fit in core memory, and thus we could not compare our algorithm against Yuksel's and Corsini's methods.

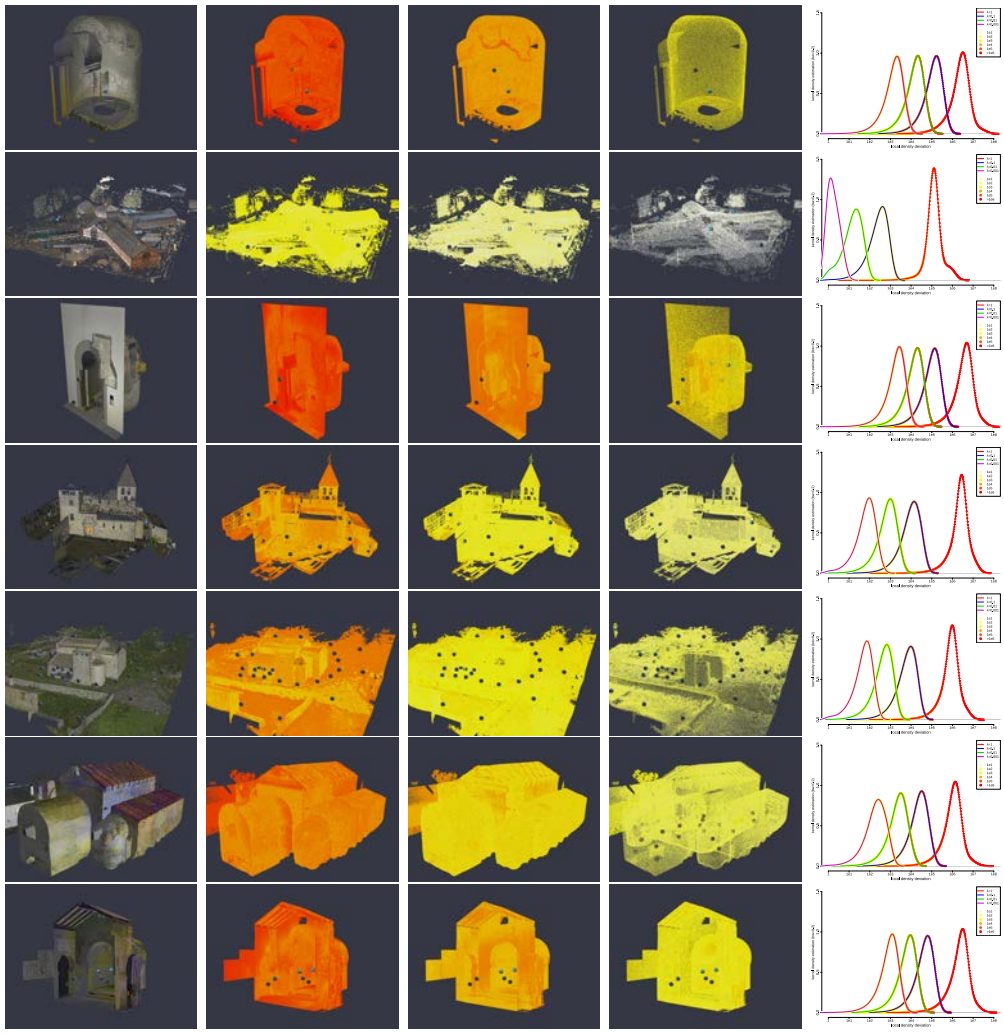


Figure 4.8: Evaluation of our results on real data. From top to bottom: Apse A (297 Mpts), Market (372 Mpts), Abse B (557 Mpts), Doma exterior (1026 Mpts), Pedret exterior (1191 Mpts), Pedret interior (1213 Mpts) and Museum (1250 Mpts). From left to right: **(1)** Phong-shaded Cloud. Blue spheres are scan locations. **(2,3,4)** Simplified clouds (10%, 1% and 0.1% of the original samples). Color-coded local point density deviation w.r.t. to the mean density, using a log scale and *white* (0), *yellow* (10^3) and *red* (10^6) interpolated colors. **(5)** Distribution of local density deviations on the original (rightmost curve) and the three sub-sampled clouds.

Running times for our algorithm varied from about 5 hours (about 300 million points) to 56 hours (1.2 billion points). Overall, the results follow the same tendency as synthetic models, as expected. Since the scanner was mounted on a tripod, large density variations on the floor were quite apparent in the original datasets, from highly dense regions near sensor locations to poorly sampled surfaces far away from these locations. Our decimation algorithm succeeded in equalizing densities and reaching exactly the target point cloud after about 30 iterations. Figure **4.8** demonstrates our algorithm's excellent behavior in reducing the local density deviation to the average density (last column). The shift between the curves representing density deviations for the 100%, 10%, 1%, and 0.1% clouds span about four orders of magnitude. The most significant improvement (about five orders of magnitude) was observed in the Market model. This model has a small number of scans compared to the extent of the captured volume (the building footprint was about 10,000 m^2). Scan sparsity results in large density disparities that were substantially equalized after the decimation. We also observed this trend in other models with sparse scans (e.g., Doma exterior).

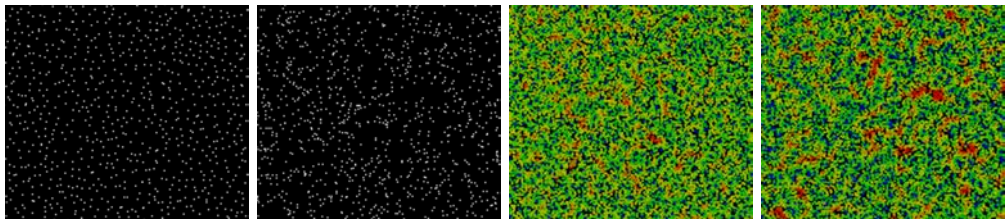


Figure 4.9: From left to right: **(1)** distribution of samples selected by our algorithm on a roughly planar surface (close-up from our 10% decimation of the Apse B model). Some apparently tight point pairs are actually far apart in 3D, due to some samples from distant scans being off the surface; **(2)** samples selected by random sampling; **(3)** error measured as the distance from each original point to its closest point in our decimated sample; **(4)** the same error in the random sample.

Figure **4.9** shows a representative distribution of samples selected by our algorithm on a planar surface. Although the distribution does not exhibit clear Blue-noise properties, surface coverage is reasonably good, as demonstrated by cloud-to-cloud error distances, which in this case, measure sampling gaps.

Performance analysis

We further analyzed our running times with synthetic and real models (up to 1.2 billion points), with substantial sub-sampling ratios (10% and 1%). The results are shown and discussed in Figure **4.10**.

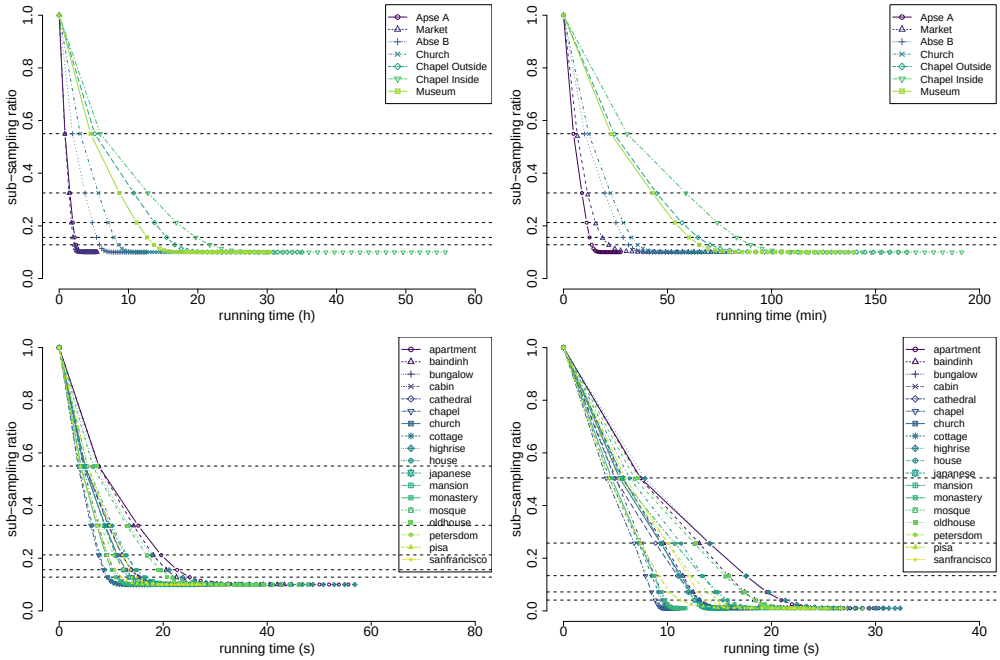


Figure 4.10: Running times for real **(top)** and synthetic **(bottom)** datasets. The **(top)** row shows the times for simplifying from 100% to 10% of the original samples **(left)** and for simplifying from 10% to 1% of the original samples **(right)** for the real models Apse A (297 Mpts), Market (372 Mpts), Abse B (557 Mpts), Doma exterior (1026 Mpts), Pedret exterior (1191 Mpts), Pedret interior (1213 Mpts) and Museum (1250 Mpts). On the **(bottom)** row we show the times for simplifying from 100% to 10% of the original samples **(left)** and from 100% to 1% of the original samples **(right)**for the synthetic models apartment (676 Kpts), baidinh (470 Kpts), bungalow (662 Kpts), cabin (778 Kpts), cathedral (437 Kpts), chapel (538 Kpts), church (429 Kpts), cottage (654 Kpts), highrise (659 Kpts), house (512 Kpts), japanese (552 Kpts), mansion (614 Kpts), monastery (550 Kpts), mosque (514 Kpts), oldhouse (804 Kpts), petersdom (397 Kpts), pisa (457 Kpts) and sanfrancisco (690 Kpts). We have found experimentally that each decimation iteration roughly removes half of the samples to the target (the first iterations are represented as horizontal dashed lines). E.g. when aiming to a 10% of the original samples, the first iteration will remove 45% of those, the second 22.5% and so on. We have also found that the execution does not only depend on the number of points but also on their distribution, for instance Museum (1250 Mpts) took shorter time than Pedret interior (1213 Mpts).

Remarkably, we could confirm experimentally the percentage of points removed at each iteration. For all models, each decimation iteration roughly removed half of the samples to the target. For example, when aiming at 10% of the original samples, the first iteration removed 45% of those and the second 22.5%. Similar behavior was observed for all iterations. This confirmed our hypothesis that iteration i would remove about $(1-\lambda)/2^i|C|$ points and would leave about the same quantity of points for the next iterations. This is a direct consequence of our flagging strategy *dirty* points (those requiring a cost update). Due to our algorithm's incremental nature, we observed that we could stop after a fixed number of iterations and already get a sub-sampled cloud close to the target decimation ratio. For example, after 14 iterations, our algorithm has already generated a $\lambda + 0.001$ decimation. This would further speed-up the execution at the expense of a bounded deviation from the target number of points. The impact of this early termination on density quality is negligible even in highly-uneven datasets. For example, a high-density disk with 2^i samples could be decimated to a single point after about i iterations. This means that, e.g., 14 iterations are enough to equalize densities even when the target density is four orders of magnitude smaller than that of the densest areas in the input cloud.

Additional cost terms

In favor of making the simplification process **feature-aware**, the proposed cost function can be modified by incorporating orientation-based terms (assuming the input point cloud is oriented). However, differentiating between high-frequency detail and noise is a challenging task. Because of this, in Section 4.2, we will elaborate on how to compute normal vectors on noisy data in a robust way.

Assuming we can compute robust normals, the idea is to modify the cost function to penalize the removal of nearby points that exhibit a large orientation difference. The motivation is twofold. First, when computing pair-wise distances, we should ideally consider geodesic rather than Euclidean distances. We could assume the geodesic distance to be close to the Euclidean distance for close points with similar normal directions. In contrast, points with opposite normal directions might belong to different surface sheets (e.g., different sides of a thin object). Thus their true geodesic distance might deviate arbitrarily from the Euclidean one. Similarly, our overall goal of equalizing point densities should be understood as referring to densities associated with samples belonging to the same local surface patch.

Figure 4.11 shows some results using our original cost function $w_{\mathbf{p}}$ multiplied by a $1 + \alpha \cos\theta = 1 + \alpha \mathbf{n_p} \cdot \mathbf{n_{p'}}$ factor, θ being the angle between the normal $\mathbf{n_p}$ of point $\mathbf{p}$ and that of its closest point $\mathbf{p'}$. The control parameter *alpha* regulates

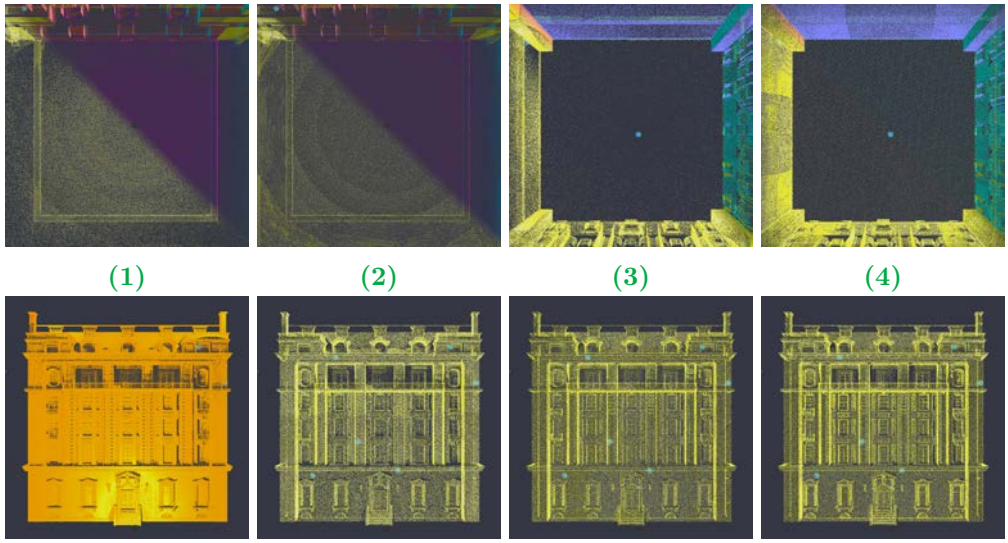


Figure 4.11: Effect of incorporating additional terms to the cost function. We used the *Mansion* polygonal model (Figure **4.2**), which was synthetically sampled to 57 Mpts. We first sub-sampled the cloud to 10% using the unmodified cost function. Images **(1)** and **(3)** show bottom/top views of the resulting cloud. Rendering the cloud with normal culling, some gaps are apparent because the interior and exterior sides of some surfaces were processed jointly (in this model, walls were modeled with a single two-side surface sheet, with no thickness). We can avoid this by incorporating a normal-based weight to the cost function so that samples from different surface sides are ranked independently **(2)**, **(4)**. Image **(5)** shows the local density deviation on the original cloud. Images **(6,7,8)** also show density deviation on 10% decimation computed with different cost functions: original cost **(6)**, using distances to the closest point on the same surface, based on normal agreement **(7)**, and using the original cost but with a weight factor based on the normal variation on the neighborhood **(8)**. Images **(6,7,8)** show that making the cost orientation-aware causes sample points to accumulate around feature edges.

how much influence the orientation has overall the simplification.

Figure 4.11 illustrates how this factor improves sampling, e.g., on both sides of thin structures. In Figure 4.11 this occurs almost everywhere because many walls were modeled with a single surface sheet. In real-world models, we also observed the benefits of this approach (although to a lesser extent), e.g., on very thin objects such as metal plates.

Analogously, we could also add a color-deviation term to penalize the removal of points with distinct colors so that density is preserved in those regions with high-frequency color details. This could be desirable, e.g., when decimating artifacts (e.g., frescoes) for which color information might be more relevant than shape, for certain studies.

Finally, since we assume the scanned data has been acquired with a high-end scanner for which accurate error models are known, we plan to incorporate such error models into our cost strategy as future work. This will make our sub-sampling sensor-aware in the same spirit as some point processing methods [CAC+18]. For example, a particular region might include accurate samples from a nearby-frontal scan and other less-accurate samples from distant or tangential scans. Adding this sensor-aware term to the cost definition prioritizes removing inaccurate samples in those regions where more accurate points are available. Otherwise, the algorithm would preserve the noisy ones. Figure 4.11 demonstrates this idea on a synthetic model.

4.2 SENSOR-AWARE NORMAL ESTIMATION

Robust normal estimation in unorganized point cloud data is a classical problem that has been widely studied. Robust normals are critical for different point-based applications such as visualization, surface reconstruction, or simplification. Many works tackle this task by performing a covariance analysis on the local neighborhood around each point [HDD+92; ABC+01; PGK02; MN03; ACT+07; HLZ+09; MOG11; HWG+13; CLZ13; ZCL+13; NBW14; LY15; KL17; CCZ+18; SDC+20]. Usually, the neighbor-based k-neighborhood ($k \in \mathbb{N}$) is preferred over the radius-based r-neighborhood ($r \in \mathbb{R}$). This is due to both for efficiency reasons (as we have seen in the previous section) and because the former adapts better to the local point densities (Figure 4.12).

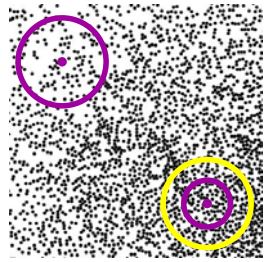


Figure 4.12: k-neighborhoods provide a way of computing adaptive r-neighborhoods on point clouds with uneven densities. The k-neighborhood (*purple* circle) is able to retrieve higher-frequency information in very dense regions by using a smaller search radius, whereas using a fixed radius r-neighborhood (*yellow* circle) smooths-out these.

Using the k-neighborhood, with fixed k value, for the covariance analysis works well whenever the point properties across the cloud, except for local density, are homogeneous (e.g., noise level). However, LiDAR point clouds can be highly heterogeneous for reasons such as:

- Non-uniform noise level across the cloud (error depends on the range distance).
- Regions with a mixture of points with very different noise levels (Figure 4.32).
- Missing parts due to disoccluded areas (Figure 4.31).
- High anisotropy of the point distributions (points arranged in scan-lines).
- Presence of sharp features.

Some of these characteristics are shared among point clouds from common sources (LiDAR, triangulation, multi-view-stereo) and have drawn different authors' attention. For instance, there are different works that focus on being adaptive to the noise level [MN03; DS06; MOG11; CLZ13; ZCL+13; CLZ13], robust against outliers [LCL+07; HLZ+09; NBW14; LY15; KL17; SDC+20] or keen on edge preservation [LSK+10; MOG11; BM12; CLZ13; ZCL+13; LZC+15; HWG+13; BM16; CCZ+18]. Most of these methods rely on approximating the covariance matrix at a point using its k-neighborhood. In contrast, some rely on the estimation of Voronoi covariances [ACT+07; MOG11]. There are also some approaches to detect cloud boundaries and highly-anisotropic neighborhoods [GUM01].

However, none of these methods specifically consider the properties unique to LiDAR point clouds, namely heterogeneous noise-levels and error directionality. Taking into account these characteristics, we formulate the following desired properties for a LiDAR point cloud normal estimation algorithm:

- **Noise-Awareness:** A LiDAR point cloud is composed of the union of various registered range scans, where a single surface can be represented by points captured from both nearby and distant sensors. When processing them, it must be considered that these points may have highly different expected measurement noise.

  Moreover, we consider the most relevant error to happen in the beam direction. Error orthogonal to the beam direction (i.e., error in angular measurements) is mostly systematic and due to temperature changes [Wal15]. Since this shift is mostly consistent, it has little impact on the normal estimation.

  Hence, we propose a sensor-aware noise model and analyze its effect on the traditional principal component analysis (PCA) approach. We derive a way to account for directionality when computing covariance matrices. Moreover, we also derive neighborhood search bounds to ensure that this correction is robust enough with a given probability. Moreover, we propose using error-based weights to combine the information from points with different noise levels.

- **Orientation-Awareness:** In unorganized point clouds, estimated normals vectors need to be coherently oriented. This is key in order to be able to define *inside* and *outside* sub-spaces for tasks such as surface reconstruction or visibility culling.

  Fortunately, orienting normals in LiDAR point clouds is a trivial task if the sensor's position from which each point was captured is known. Any

scanning equipment generally provides this information.

- **Feature-Awareness:** Feature over-smoothing happens when considering points across a feature to estimate normals. This may happen when using too large search radii.

  Our approach can find the minimum search radius that guarantees noise is being correctly filtered. Namely, the minimum radius to distinguish between high-frequency detail and noise. Moreover, our robust estimation of covariance matrices can be combined with other algorithms in the literature that further emphasize features.

- **Adaptability:** It has been argued that using k-neighborhoods allows adapting to the underlying point density. Moreover, our algorithms take into account local density when computing neighborhood search bounds.

- **Scalability:** This property is probably easier to fulfill than for the case of simplification. As we have seen, most normal estimation algorithms rely on a very local neighborhood, which allows processing inputs of any size. Nevertheless, we note that performing a global refinement step to preserve hard edges [ZCL+13] or computing a global Voronoi subdivision [ACT+07; MOG11] are not feasible for massive points clouds. These approaches should be adapted to work on a subdivision of the cloud.

  Hence, we propose an approach that works on the local neighborhood of a point. This allows efficiently and independently processing each of the voxels of the voxelization of the input clouds.

- **Efficiency:** The algorithm for normal estimation must be efficient in order to be able to process massive point clouds within a feasible time. For PCA-based algorithms on the k-neighborhood, the baseline is Hoppe et al. [HDD+92]. Anything more complex will undoubtedly be slower. Our algorithm outperforms Hoppe's in terms of accuracy while keeping execution time-bounded within a constant factor.

Existing methods assume that locally every point within a small region of the point cloud has a measurement error with comparable statistical properties. In LiDAR point clouds, the point's expected measurement noise depends on the material's reflective properties and distance to the sensor. Therefore, we can have mixtures of points with significantly different noise levels, which violates this assumption. We propose and discuss a normal estimation method that takes these properties into account. Our approach considers, for each point, a 1D directional probability distribution with variance proportional to its associated measurement error.

4.2.1 Problem Formulation

Consider a point cloud C_Q which results from the union of a set registered scans from locations $Q \subset \mathbb{R}^3$, namely $C_Q = \bigcup_{\mathbf{q} \in Q} C_{\mathbf{q}}$. We assume the positions $\mathbf{q}$, as well as the registration matrices for each cloud, are given as inputs. The sensor at a given position $\mathbf{q} \in Q$ casts a beam and hits surface π at an unknown true surface point $\mathbf{p}$, capturing the noisy point $\tilde{\mathbf{p}}$ along with its reflected intensity $i_{\mathbf{p}}$.

In current LiDAR equipment, the measurement error along the ray direction depends mostly on the point-to-sensor distance and the reflected intensity. Using the device technical specifications (see Table 4.1) and the distance $d(\mathbf{q}, \tilde{\mathbf{p}}) = ||\tilde{\mathbf{p}} - \mathbf{q}||$ from each point to its sensor location we can estimate the standard deviation $\sigma_{\mathbf{p}}$ of the associated error for that sample. Notice that $\sigma_{\mathbf{p}}$ is spatially-varying and different for each sample.

We base our approach on a scanner-centric error model. We assume that $\tilde{\mathbf{p}}$ lays in the laser beamline, with parametric equation:

$$\tilde{\mathbf{p}} = \mathbf{p} + \lambda(\mathbf{p} - \mathbf{q}), \qquad \lambda \in \mathbb{R} \tag{4.10}$$

In our model, $\tilde{\mathbf{p}}$ is slightly apart from $\mathbf{p}$, at a distance given by a Gaussian distribution with a standard deviation $\sigma_{\mathbf{p}}$. LiDAR scanners are usually much more accurate in angular measurements than in range measurements. Therefore, we neglect errors in orthogonal directions.

For instance, the angular uncertainty of the Leica P20 is 8 arc seconds, i.e., about 0.002 degrees (0.4 mm at 10m). RMS range noise depends on the per-sample distance and surface reflectivity. For the P20, the range noise at 10m for dark surfaces is 0.8 mm, twice larger than the angular error. Moreover, error in angular measurements is mostly systematic. Temperature changes cause individual components and laser deflectors (with different thermal expansion coefficients) to move at the micron scale [Wal15]. This process causes shifts that are shift is mostly consistent. Therefore, their impact on normal estimation is negligible compared to random noise on range measurements.

Although $\mathbf{p}$ is unknown, we can estimate the unit-length direction $\mathbf{d}$ of the laser beam line because we assume $\tilde{\mathbf{p}}$ lays on it:

$$\mathbf{d} = \frac{\mathbf{p} - \mathbf{q}}{\|\mathbf{p} - \mathbf{q}\|} = \frac{\tilde{\mathbf{p}} - \mathbf{q}}{\|\tilde{\mathbf{p}} - \mathbf{q}\|}. \tag{4.11}$$

Let $\mathfrak{N}(0, \sigma_{\mathbf{p}})$ be a normal statistical distribution with zero mean and variance $\sigma_{\mathbf{p}}$. Neglecting systematic range errors (i.e. neglecting bias), noisy points $\tilde{\mathbf{p}}$ can be modeled as (Figure 4.1):

$$\tilde{\mathbf{p}} = \mathbf{p} + \mathbf{d}n_{\mathbf{p}}, \quad n_{\mathbf{p}} \sim \mathfrak{N}(0, \sigma_{\mathbf{p}}) \tag{4.12}$$

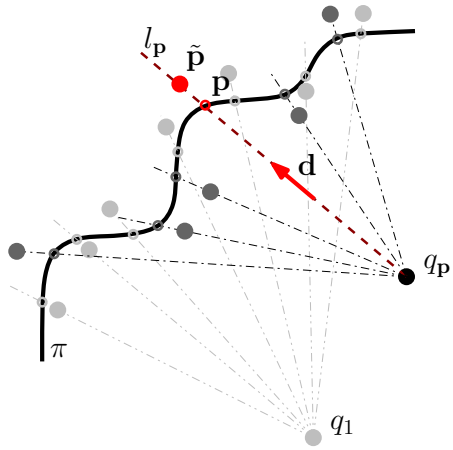


Figure 4.13: Surface π scanned from two different sensor positions. q_1 is the scanner position for the light-grey points and $\mathbf{q_p}$ is the scanner position for the dark-grey points. For each point $\mathbf{p}$, a noisy point $\tilde{\mathbf{p}}$ is captured due to measurement imperfection. We assume point $\tilde{\mathbf{p}}$ lays on the line defined by points $\mathbf{q_p}$ and $\mathbf{p}$. Sample spacing has been exaggerated for clarity; in LiDAR equipment neighbor samples share a nearly constant direction $\mathbf{d}$.

Let $\mathcal{N}_r(\tilde{\mathbf{p}})$ be the neighbors of point $\tilde{\mathbf{p}}$ within a search radius r, i.e. the subset of scanned points within r distance from $\tilde{\mathbf{p}}$. The traditional least squares method estimates the local geometry going through $\mathcal{N}_r(\tilde{\mathbf{p}})$ as the plane that minimizes the squares distances to $\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})$. This, in fact, restricts the search space to those planes containing the centroid of the points in the neighborhood, and can be efficiently solved by performing principal component analysis (PCA) on the covariance matrix (see derivation later):

$$\mathbf{CV}(\mathcal{N}_r(\tilde{\mathbf{p}})) = \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} (\tilde{\mathbf{p}}_i - \mathbf{c}_{\tilde{\mathbf{p}}})(\tilde{\mathbf{p}}_i - \mathbf{c}_{\tilde{\mathbf{p}}})^T \tag{4.13}$$

with $\mathbf{c}_{\tilde{\mathbf{p}}} = \frac{1}{k} \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} \tilde{\mathbf{p}}_i$ and $k = |\mathcal{N}_r(\tilde{\mathbf{p}})|$.

Given this problem formulation, the normal of the optimal least-squares plane can be estimated as the eigenvector of $\mathbf{CV}(\mathcal{N}_r(\tilde{\mathbf{p}}))$ with the smallest eigenvalue.

Following, we will develop the terms of this formula to show that refactor the covariance matrix of the noisy points $\mathbf{CV}(\mathcal{N}_r(\tilde{\mathbf{p}}))$ can be expressed in terms of the covariance matrix of the true surface points $\mathbf{CV}(\mathcal{N}_r(\mathbf{p}))$ plus a term depending on $\mathbf{d}$ and an approximation of $\sigma_{\mathbf{p}}$ (Figure 4.14). Using this, we will be able to estimate the normal vectors of the noisy points from the covariance matrix of the true surface points (Subsection 4.2.2).

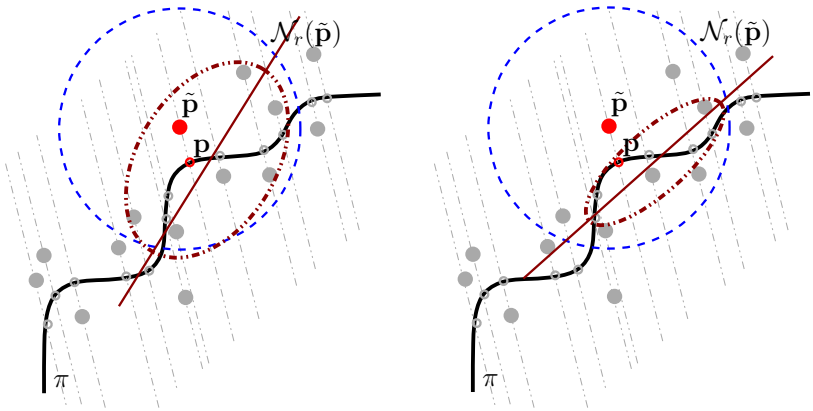


Figure 4.14: Surface π scanned from one sensor. When the sensor is far away from π, directions $\mathbf{d}$ are similar. The red dashed lines show ellipsoids fitted to the noisy points (left) or to the true surface points (right). The first one is usually thicker than the second. The fitted planes also differ due to noise. The blue circle shows the neighborhood radius used.

The robustness of this approximation (i.e., how often and how far it drifts from the real values) is determined by the number of neighbors in $\mathcal{N}_r(\tilde{\mathbf{p}})$. We use the Chebyshev inequality [Leo08] to derive probabilistic bounds on the neighborhood sizes that yield certain guarantees on the approximation error (Subsection 4.2.3).

This solves the problem of estimating normals on a single scan. However, in sparse areas, the algorithm may benefit from the information of other overlapping scans. To combine this information, we first study introducing weights inversely proportional to a point's noise level. This helps reduce the contribution of noisy points to the least-squares minimization problem (Subsection 4.2.4). Furthermore, later we evaluate how this translates to mixing the corrected covariance matrices introduced in Subsection 4.2.2.

Least squares problem derivation

In the traditional least squares approximation we want to compute the plane $\mathbf{n}_{\mathbf{p}}^T x - d = 0$ that minimizes the squared distances to the points $\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})$:

$$\begin{aligned} &\underset{\mathbf{n_p},d}{\operatorname{argmin}} \left(\frac{1}{2k} \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} (\mathbf{n_p}^T \tilde{\mathbf{p}}_i - d)^2 \right) \\ &\text{subject to } ||\mathbf{n_p}|| = 1 \end{aligned} \tag{4.14}$$

$k = |\mathcal{N}_r(\tilde{\mathbf{p}})|$. Next we compute the partial derivatives while encoding the constraint using a Lagrangian multiplier λ_L:

$$\begin{aligned} \frac{\partial}{\partial \mathbf{n_p}} \left(\frac{1}{2k} \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} (\mathbf{n_p}^T \tilde{\mathbf{p}}_i - d)^2 \right) - \lambda_L \mathbf{n_p} = 0 \\ \mathbf{n_p}^T \left(\frac{1}{k} \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} \tilde{\mathbf{p}}_i \tilde{\mathbf{p}}_i^T \right) - d \left(\frac{1}{k} \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} \tilde{\mathbf{p}}_i \right) = \lambda_L \mathbf{n_p} \end{aligned} \tag{4.15}$$

$$\begin{aligned} \frac{\partial}{\partial d} \left(\frac{1}{2k} \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} (\mathbf{n_p}^T \tilde{\mathbf{p}}_i - d)^2 \right) = 0 \\ \mathbf{n_p}^T \left(\frac{1}{k} \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} \tilde{\mathbf{p}}_i \right) = d \end{aligned} \tag{4.16}$$

now, let the centroid of $\mathcal{N}_r(\tilde{\mathbf{p}})$ be $\mathbf{c}_{\tilde{\mathbf{p}}} = \frac{1}{k} \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} \tilde{\mathbf{p}}_i$:

$$\begin{aligned} \mathbf{n_p}^T \left(\frac{1}{k} \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} \tilde{\mathbf{p}}_i \tilde{\mathbf{p}}_i^T \right) - \mathbf{n_p}^T (\mathbf{c}_{\tilde{\mathbf{p}}} \mathbf{c}_{\tilde{\mathbf{p}}}{}^T) = \lambda_L \mathbf{n_p} \\ \mathbf{n_p}^T \left(\frac{1}{k} \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} \tilde{\mathbf{p}}_i \tilde{\mathbf{p}}_i^T - \mathbf{c}_{\tilde{\mathbf{p}}} \mathbf{c}_{\tilde{\mathbf{p}}}{}^T \right) = \lambda_L \mathbf{n_p} \\ \left(\frac{1}{k} \mathbf{CV}(\mathcal{N}_r(\tilde{\mathbf{p}})) \right) \mathbf{n_p} = \lambda_L \mathbf{n_p} \\ \mathbf{CV}(\mathcal{N}_r(\tilde{\mathbf{p}})) = \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} \tilde{\mathbf{p}}_i \tilde{\mathbf{p}}_i^T - k \mathbf{c}_{\tilde{\mathbf{p}}} \mathbf{c}_{\tilde{\mathbf{p}}}{}^T = \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} (\tilde{\mathbf{p}}_i - \mathbf{c}_{\tilde{\mathbf{p}}})(\tilde{\mathbf{p}}_i - \mathbf{c}_{\tilde{\mathbf{p}}})^T \end{aligned} \tag{4.17}$$

This is the typical formulation for an eigenvalue problem. The search space is restricted to the set of planes containing the centroid $\mathbf{c}_{\tilde{\mathbf{p}}}$, λ is an eigenvalue and $\mathbf{n_p}$ an eigenvector of the covariance matrix $\mathbf{CV}(\mathcal{N}_r(\tilde{\mathbf{p}}))$ (Figure 4.14). As we formulated a minimization problem, we should pick the eigenvector with the smallest eigenvalue as the normal vector estimate.

4.2.2 Covariance Matrix Correction

In this section, we describe how, using our noise model, we can estimate normal vectors from the covariance matrix of the true surface points. We decompose the covariance matrix of the noisy points as the sum of three terms: one that depends on the covariance matrix of the true surface points and two terms that depend on the standard deviation ($\sigma_{\mathbf{p}}$) and direction ($\mathbf{d}$) of the error. Consequently, we will be able to isolate the first term (covariance matrix of the true surface points) and estimate the normal vector from it. We assume point samples have been grouped by sensor location and reflectivity. This results in clusters with similar noise distributions (this will be discussed in Subsection 4.2.5). The following analysis considers noisy points within each individual cluster.

Developing the covariance matrix of the noisy points we get (see derivation later):

$$\begin{aligned}\mathbf{CV}(\mathcal{N}_r(\tilde{\mathbf{p}})) &= \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} (\tilde{\mathbf{p}}_i - \mathbf{c}_{\tilde{\mathbf{p}}})(\tilde{\mathbf{p}}_i - \mathbf{c}_{\tilde{\mathbf{p}}})^T \\ &\approx \mathbf{CV}(\mathcal{N}_r(\mathbf{p})) + T_1 + T_2\end{aligned} \tag{4.18}$$

Neighborhoods $\mathcal{N}_r(\mathbf{p})$ and $\mathcal{N}_r(\tilde{\mathbf{p}})$ are not centered around the same point. Moreover, points $\tilde{\mathbf{p}}$ are slightly apart from their true surface counterparts $\mathbf{p}$. Hence, some points from $\mathcal{N}_r(\mathbf{p})$ may "move-out" and some points not in $\mathcal{N}_r(\mathbf{p})$ may "move-in" when considering $\mathcal{N}_r(\tilde{\mathbf{p}})$. Nevertheless, this is unlikely to happen if r is large compared to the noise values. Therefore, we assume a one-to-one correspondence between the points in $\mathcal{N}_r(\mathbf{p})$ and their noisy counterparts in $\mathcal{N}_r(\tilde{\mathbf{p}})$.

Furthermore, we assume that directions $\mathbf{d}$ for the points in $\mathcal{N}_r(\tilde{\mathbf{p}})$ are equal. This assumption is valid as long as r is small compared to $d(\mathbf{q}, \mathbf{p})$. In particular, for terrestrial LiDAR equipment r is three or four orders of magnitude smaller than $d(\mathbf{q}, \mathbf{p})$. E.g, we can configure the Leica P20 to deliver 3.1 mm resolution when scanning an orthogonal wall at 10 meters. Even for a large $k = 100$

neighborhood $\mathcal{N}_r(\mathbf{p})$, the maximum angular deviation between $\mathbf{p}$ and $\mathbf{p}_i|\mathbf{p}_i \in \mathcal{N}_r(\mathbf{p})$ would be approximately 1.7e−3 rad. Thus, making the assumption above quite reasonable. The scanner also supports 1.6 mm and 0.8 mm resolutions, which result in even smaller angular deviations within neighborhoods.

Term T_1 can be written as (see derivation later):

$$\begin{aligned} T_1 &= \left(\mathbf{d}\mathbf{d}^T \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} (n_{\mathbf{p}_i} - \bar{n})^2 \right) \\ &= \mathbf{d}\mathbf{d}^T (k-1) s_{\mathbf{p}}^2 \end{aligned} \tag{4.19}$$

where $\bar{n}$ is the sample mean of the point noises $n_{\mathbf{p}}$ and $s_{\mathbf{p}}^2 = \frac{1}{k-1} \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} (n_{\mathbf{p}_i} - \bar{n})^2$ is their sample variance.

Let $\mathbf{E}[s_{\mathbf{p}}^2]$ denote the expected value of $s_{\mathbf{p}}^2$. Since $\mathbf{E}[s_{\mathbf{p}}^2] = \sigma_{\mathbf{p}}^2$, we could estimate T_1 as $\mathbf{d}\mathbf{d}^T(k-1)\sigma_{\mathbf{p}}^2$ for a sufficiently large sample. In Subsection **4.2.3** we will study the sample size needed to estimate $s_{\mathbf{p}}^2$ within a bounded error.

The second matrix term T_2 can be written as (see derivation later):

$$\begin{aligned} T_2 &= \left(\sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} (n_{\mathbf{p}_i} - \bar{n})(\mathbf{p}_i - \mathbf{c}_{\mathbf{p}}) \right) \mathbf{d}^T \\ &\quad + \mathbf{d} \left(\sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} (n_{\mathbf{p}_i} - \bar{n})(\mathbf{p}_i - \mathbf{c}_{\mathbf{p}}) \right)^T \\ &= (k-1)(\mathbf{s}(n_{\mathbf{p}}, \mathbf{p}^{\mathbf{c}})\mathbf{d}^T + \mathbf{d}\mathbf{s}(n_{\mathbf{p}}, \mathbf{p}^{\mathbf{c}})^T) \end{aligned} \tag{4.20}$$

where $\mathbf{s}(n_{\mathbf{p}}, \mathbf{p}^{\mathbf{c}})$ is the vector of sample covariances between the (x, y, z) components of the centered points $\mathbf{p}^{\mathbf{c}} = \mathbf{p} - \mathbf{c}_{\mathbf{p}}$ and their associated noises $n_{\mathbf{p}}$.

It is known that $\mathbf{E}[\mathbf{s}(n_{\mathbf{p}}, \mathbf{p}^{\mathbf{c}})] = \mathbf{CV}(n_{\mathbf{p}}, \mathbf{p}^{\mathbf{c}})$. By definition we also know that:

$$\mathbf{CV}(n_{\mathbf{p}}, \mathbf{p}^{\mathbf{c}}) = \mathbf{E}[n_{\mathbf{p}}\mathbf{p}^{\mathbf{c}}] - \mathbf{E}[n_{\mathbf{p}}]\mathbf{E}[\mathbf{p}^{\mathbf{c}}] \tag{4.21}$$

Next, we will use the independence property between the distribution of the points $\mathbf{p}$ and the distribution of their noises $n_{\mathbf{p}}$:

$$\mathbf{E}[n_{\mathbf{p}}\mathbf{p}^{\mathbf{c}}] - \mathbf{E}[n_{\mathbf{p}}]\mathbf{E}[\mathbf{p}^{\mathbf{c}}] = \mathbf{E}[n_{\mathbf{p}}]\mathbf{E}[\mathbf{p}^{\mathbf{c}}] - \mathbf{E}[n_{\mathbf{p}}]\mathbf{E}[\mathbf{p}^{\mathbf{c}}] = 0 \tag{4.22}$$

In summary, on average, the term T_2 will vanish.

These results yield a straight-forward way to compute an approximation of the covariance matrix for the true surface points:

$$\mathbf{CV}(\mathcal{N}_r(\mathbf{p})) \approx \mathbf{CV}(\mathcal{N}_r(\tilde{\mathbf{p}})) - \mathbf{d}\mathbf{d}^T(k-1)\sigma_{\mathbf{p}}^2 \tag{4.23}$$

This formula is very intuitive. Recall that the outer product $\mathbf{d}\mathbf{d}^T$ acts as an orthogonal projection operator onto the line spanned by $\mathbf{d}$, since $(\mathbf{d}\mathbf{d}^T)\mathbf{y} = (\mathbf{d}\cdot\mathbf{y})\mathbf{d}$. Consequently, this formula shows that the effect of the noise along the line-of-sight direction $\mathbf{d}$ increases the variance along this direction.

By assuming that all the points in $\mathcal{N}_r(\tilde{\mathbf{p}})$ have similar beam directions $\mathbf{d}$, we can determine this effect and correct it. Therefore, we can compute the normal vectors for the noisy points from the true surface points' covariance matrix. This will provide better (noise-aware) normal estimates of the true surface (Figure 4.14).

Corrected covariance matrix derivation

The centroid $\mathbf{c}_{\tilde{\mathbf{p}}}$ of $\mathcal{N}_r(\tilde{\mathbf{p}})$ can be written as:

$$\begin{aligned}\mathbf{c}_{\tilde{\mathbf{p}}} &= \frac{1}{k}\sum_{\tilde{\mathbf{p}}_i\in\mathcal{N}_r(\tilde{\mathbf{p}})}\tilde{\mathbf{p}}_i = \frac{1}{k}\sum_{\tilde{\mathbf{p}}_i\in\mathcal{N}_r(\tilde{\mathbf{p}})}(\mathbf{p}_i + \mathbf{d}n_{\mathbf{p}_i}) \\ &= \frac{1}{k}\sum_{\tilde{\mathbf{p}}_i\in\mathcal{N}_r(\tilde{\mathbf{p}})}\mathbf{p}_i + \frac{1}{k}\sum_{\tilde{\mathbf{p}}_i\in\mathcal{N}_r(\tilde{\mathbf{p}})}\mathbf{d}n_{\mathbf{p}_i}\end{aligned} \tag{4.24}$$

Assuming $\mathbf{d}$ and $\sigma_{\mathbf{p}_i}$ to be equal for all the points in $\mathcal{N}_r(\tilde{\mathbf{p}})$:

$$\frac{1}{k}\sum_{\tilde{\mathbf{p}}_i\in\mathcal{N}_r(\tilde{\mathbf{p}})}\mathbf{d}n_{\mathbf{p}_i} = \mathbf{d}\frac{1}{k}\sum_{\tilde{\mathbf{p}}_i\in\mathcal{N}_r(\tilde{\mathbf{p}})}n_{\mathbf{p}_i} = \mathbf{d}\bar{n} \tag{4.25}$$

Where $\bar{n}$ is the sample mean of $n \sim \mathcal{N}(0, \sigma_{\mathbf{p}})$, therefore $\bar{n} \approx 0$. Assuming $\mathcal{N}_r(\tilde{\mathbf{p}}) \approx \mathcal{N}_r(\mathbf{p})$:

$$\mathbf{c}_{\tilde{\mathbf{p}}} = \frac{1}{k} \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} \mathbf{p}_i + \mathbf{d}\bar{n} = \mathbf{c}_{\mathbf{p}} + \mathbf{d}\bar{n} \approx \mathbf{c}_{\mathbf{p}} \tag{4.26}$$

Next, recall that the covariance matrix for the centred cloud can be estimated as:

$$\mathbf{CV}(\mathcal{N}_r(\tilde{\mathbf{p}})) = \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} (\tilde{\mathbf{p}}_i - \mathbf{c}_{\tilde{\mathbf{p}}})(\tilde{\mathbf{p}}_i - \mathbf{c}_{\tilde{\mathbf{p}}})^T \tag{4.27}$$

Let $\mathbf{p}_\mathbf{i}^\mathbf{c} = \mathbf{p}_i - \mathbf{c}_\mathbf{p}$. Then

$$(\tilde{\mathbf{p}}_i - \mathbf{c}_{\tilde{\mathbf{p}}}) = (\mathbf{p}_i + \mathbf{d}n_{\mathbf{p}_i}) - (\mathbf{c}_\mathbf{p} + \mathbf{d}\bar{n}) = \mathbf{p}_\mathbf{i}^\mathbf{c} + \mathbf{d}(n_{\mathbf{p}_i} - \bar{n}) \tag{4.28}$$

Therefore:

$$\begin{aligned}
\mathbf{CV}(\mathcal{N}_r(\tilde{\mathbf{p}})) &= \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} (\mathbf{p}_\mathbf{i}^\mathbf{c} + \mathbf{d}(n_{\mathbf{p}_i} - \bar{n}))(\mathbf{p}_\mathbf{i}^\mathbf{c} + \mathbf{d}(n_{\mathbf{p}_i} - \bar{n})))^T \\
&= \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} \mathbf{p}_\mathbf{i}^\mathbf{c} \mathbf{p}_\mathbf{i}^{\mathbf{c}T} + \mathbf{d}\mathbf{d}^T \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} (n_{\mathbf{p}_i} - \bar{n})^2 \\
&+ \left(\sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} (n_{\mathbf{p}_i} - \bar{n})(\mathbf{p}_\mathbf{i}^\mathbf{c}) \right) \mathbf{d}^T \\
&+ \mathbf{d} \left(\sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} (n_{\mathbf{p}_i} - \bar{n})(\mathbf{p}_\mathbf{i}^\mathbf{c}) \right)^T
\end{aligned} \tag{4.29}$$

$\mathbf{CV}(\mathcal{N}_r(\mathbf{p})) = \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} \mathbf{p}_\mathbf{i}^\mathbf{c} \mathbf{p}_\mathbf{i}^{\mathbf{c}T}$ is the covariance matrix of the true points. Now we simply define T_1 and T_2 as follows:

$$T_1 = \left(\mathbf{d}\mathbf{d}^T \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} (n_{\mathbf{p}_i} - \bar{n})^2 \right) \tag{4.30}$$

$$T_2 = \left(\sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} (n_{\mathbf{p}_i} - \bar{n})(\mathbf{p}_\mathbf{i}^\mathbf{c}) \right) \mathbf{d}^T + \mathbf{d} \left(\sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r(\tilde{\mathbf{p}})} (n_{\mathbf{p}_i} - \bar{n})(\mathbf{p}_\mathbf{i}^\mathbf{c}) \right)^T \tag{4.31}$$

We will proceed by analyzing T_1. Let $s^2_\mathbf{p}$ be the sample variance of $n \sim \mathcal{N}(0, \sigma^2_\mathbf{p})$. Then

$$E[s^2(n)] = \sigma^2_\mathbf{p}, \quad Var(s^2(n)) = \frac{2\sigma^4_\mathbf{p}}{k-1} \tag{4.32}$$

$$T_1 = \mathbf{d}\mathbf{d}^T(k-1)s^2_\mathbf{p} \approx \mathbf{d}\mathbf{d}^T(k-1)\sigma^2_\mathbf{p} \tag{4.33}$$

We now develop T_2. Let $\mathbf{s}(n_\mathbf{p}, \mathbf{p}_\mathbf{i}^\mathbf{c})$ be the vector of sample covariances between the (x, y, z) components of the centered points $\mathbf{p}_\mathbf{i}^\mathbf{c} = \mathbf{p}_i - \mathbf{c}_\mathbf{p}$ and their associated noises $n_\mathbf{p}$:

$$T_2 = (k-1)(\mathbf{s}(n_\mathbf{p}, \mathbf{p}_\mathbf{i}^\mathbf{c})\mathbf{d}^T + \mathbf{d}\mathbf{s}(n_\mathbf{p}, \mathbf{p}_\mathbf{i}^\mathbf{c})^T) \approx 0 \tag{4.34}$$

$$E[\mathbf{s}(n_\mathbf{p}, \mathbf{p}_\mathbf{i}^\mathbf{c})] = 0, \quad Var(\mathbf{s}(n_\mathbf{p}, \mathbf{p}_\mathbf{i}^\mathbf{c})) = \frac{\sigma^2_\mathbf{p} r^2}{k-1} \tag{4.35}$$

In summary:

$$\mathbf{CV}(\mathcal{N}_r(\mathbf{p})) \approx \mathbf{CV}(\mathcal{N}_r(\tilde{\mathbf{p}})) - \mathbf{d}\mathbf{d}^T(k-1)\sigma^2_\mathbf{p} \tag{4.36}$$

4.2.3 Neighborhood Size Bounds

We have shown how to estimate the true surface points' covariance matrices by replacing sample variances and covariances with their distribution counterparts. The accuracy of these approximations will depend on the number of samples taken into account. In this section, we use Chebyshev inequality [Leo08]

to obtain bounds on the neighborhood size. These guarantee the accuracy of the approximations by assuring that, with a certain probability, our coefficient estimations will be within a fixed error. these

Following, we are going to show that Equations (4.19) and (4.20) are probabilistic approximations. Meaning, they will work on average, but they will diverge with a certain probability. This probability is a function of the variances associated with the corresponding terms.

T_1 correction term

Knowing that $\mathbf{E}[s_{\mathbf{p}}^2] = \sigma_{\mathbf{p}}^2$, we have shown that the T_1 matrix can be estimated as:

$$T_1 \approx \mathbf{d}\mathbf{d}^T(k-1)\sigma_{\mathbf{p}}^2 \tag{4.37}$$

Since $\mathbf{d}$ are unit vectors, $(k-1)|s_{\mathbf{p}}^2 - \sigma_{\mathbf{p}}^2|$ is an upper bound of the error introduced in Equation (4.23) by approximating $s_{\mathbf{p}}^2$ using $\sigma_{\mathbf{p}}^2$. Moreover, assuming each point in $\mathcal{N}_r(\mathbf{p})$ is exactly at distance r from $\mathbf{p}$, an upper-bound on the magnitude of the coefficients of $\mathbf{CV}(\mathcal{N}_r(\mathbf{p}))$ is kr^2. However, points will usually be more evenly-distributed, and the average magnitude of these coefficients will be a fraction of kr^2. This will be referred as ϵkr^2.

Now, we will use Chebyshev inequality [Leo08] to ensure that, with a certain probability δ, the magnitude of the introduced errors is relatively smaller than the magnitude of the coefficients of $\mathbf{CV}(\mathcal{N}_r(\mathbf{p}))$:

$$P(|(k-1)(s^2 - \sigma_{\mathbf{p}}^2)| < \epsilon kr^2) \geq 1 - \frac{Var((k-1)s_{\mathbf{p}}^2)}{(\epsilon kr^2)^2} \geq \delta \tag{4.38}$$

It is known that $Var(s_{\mathbf{p}}^2) = \frac{2\sigma_{\mathbf{p}}^4}{k-1}$. Therefore:

$$P(|(k-1)(s^2 - \sigma_{\mathbf{p}}^2)| < \epsilon kr^2) \geq 1 - \frac{2\sigma_{\mathbf{p}}^4(k-1)}{\epsilon^2 k^2 r^4} \geq 1 - \frac{2\sigma_{\mathbf{p}}^4}{\epsilon^2 k r^4} \tag{4.39}$$

Using the local point density $\rho_{\mathbf{p}}$ at point $\mathbf{p}$ we can relate k and r. Particularly, we will assume that points in $\mathcal{N}_r(\mathbf{p})$ lay on a plane, consequently $\rho_{\mathbf{p}} = \frac{k}{\pi r^2}$. By isolating r we get:

$$r^6 \geq \frac{2\sigma_{\mathbf{p}}^4}{\epsilon^2 \rho_{\mathbf{p}} \pi (1-\delta)} \tag{4.40}$$

Similarly, we can also express this bounds in terms of the number of neighbors k:

$$k^3 \geq \frac{2\sigma_{\mathbf{p}}^4 \pi^2 \rho_{\mathbf{p}}^2}{\epsilon^2 (1-\delta)} \tag{4.41}$$

T_2 vanishing term

Knowing $\mathbf{E}[\mathbf{s}(n_{\mathbf{p}}, \mathbf{p^c})] = \mathbf{CV}(n_{\mathbf{p}}, \mathbf{p^c}) = 0$, we have shown that T_2 can be estimated as:

$$T_2 \approx (k-1)(\mathbf{s}(n_{\mathbf{p}}, \mathbf{p^c})\mathbf{d}^T + \mathbf{ds}(n_{\mathbf{p}}, \mathbf{p^c})^T) \approx 0 \tag{4.42}$$

An upper-bound on the error introduced in Equation (4.23) by approximating $\mathbf{s}(n_{\mathbf{p}}, \mathbf{p^c})$ using $\mathbf{CV}(n_{\mathbf{p}}, \mathbf{p^c})$ is $2(k-1)(\mathbf{s}(n_{\mathbf{p}}, \mathbf{p^c}) - \mathbf{CV}(n, \mathbf{p^c})) = 2(k-1)\mathbf{s}(n_{\mathbf{p}}, \mathbf{p^c})$.

Again, we use Chebyshev inequality [Leo08] to ensure that, with a certain probability δ, the magnitude of the introduced errors is smaller than the magnitude of the coefficients of $\mathbf{CV}(\mathcal{N}_r(\mathbf{p}))$:

$$P(|2(k-1)\mathbf{s}(n_{\mathbf{p}}, \mathbf{p^c}))| < \epsilon k r^2) \geq 1 - \frac{Var(2(k-1)\mathbf{s}(n_{\mathbf{p}}, \mathbf{p^c}))}{(\epsilon k r^2)^2} \geq \delta \tag{4.43}$$

It can be shown that $Var(\mathbf{s}(n_{\mathbf{p}}, \mathbf{p^c})) = \frac{\sigma_{\mathbf{p}}^2 Var(\mathbf{p^c})}{k-1}$, by assuming that the distribution of the points $\mathbf{p^c}$ is independent of the distribution of their noises $n_{\mathbf{p}}$. The distribution of the points $\tilde{\mathbf{p}}_{\mathbf{c}} | \tilde{\mathbf{p}} \in \mathcal{N}_r(\mathbf{p})$ is unknown, but it is trivial to see that an upper bound for its variance is r^2. Therefore, $Var(\mathbf{s}(n_{\mathbf{p}}, \mathbf{p^c})) \leq \frac{\sigma_{\mathbf{p}}^2 r^2}{k-1}$:

$$P(|2(k-1)\mathbf{s}(n_{\mathbf{p}}, \mathbf{p^c}))| < \epsilon k r^2) \geq 1 - \frac{4\sigma_{\mathbf{p}}^2}{\epsilon^2 k r^2} \geq \delta \tag{4.44}$$

Using the point density $\rho_{\mathbf{p}}$:

$$r^4 \geq \frac{4\sigma_{\mathbf{p}}^2}{\epsilon^2 \rho_{\mathbf{p}} \pi (1-\delta)} \tag{4.45}$$

Similarly, we can also express this bounds in terms of the number of neighbors k:

$$k^2 \geq \frac{2\sigma_{\mathbf{p}}^2 \pi \rho_{\mathbf{p}}}{\epsilon^2 (1-\delta)} \tag{4.46}$$

The interpretation of the results in Equation (4.40) and Equation (4.45) is very intuitive. In both cases, the neighborhood radius is proportional to (a power of) the noise's standard deviation and inversely proportional to (a power of) the point cloud density. In short, if the noise is small and the point density is high, our bound will pick a small radius and vice versa. Furthermore, recall that our bounds ensure tolerance within ϵ with probability δ. Therefore, choosing a small tolerance and a high probability makes the radius increase and vice-versa.

Let the radius estimated via Equation (4.40) be r_1 and the radius estimated via Equation (4.45) be r_2. We can analyze the relation between these two radii by taking the rate between r_1 and r_2. We can determine that $r_1 > r_2$ when:

$$\frac{r_1}{r_2} = \frac{\sqrt[6]{\sigma_{\mathbf{p}}}\ \sqrt[12]{\epsilon^2 \rho_{\mathbf{p}} \pi (1-\delta)}}{\sqrt[3]{2}} > 1 \tag{4.47}$$

Equivalently:

$$\frac{\sigma_{\mathbf{p}}^2 \epsilon^2 \rho_{\mathbf{p}} \pi (1-\delta)}{16} > 1 \tag{4.48}$$

On our test scanner (Leica P20), the maximum reported value for $\sigma_{\mathbf{p}}$ is $0.009m$. By configuring the scanner to leave a spacing of $3.1mm$ between samples, we obtain an average density of $10^5 points/m^2$. In this setup, even if we chose $\epsilon = 1$ and $\delta = 0$, the rate value would be of the order of 10^{-5}. Consequently, we propose to pick a radius greater or equal to r_2, which would be the maximum of the two.

Notice that using r_2 bounds the error on the covariance matrix's coefficients, not on the final normal vector that would result from the eigenanalysis of this matrix.

4.2.4 Mitigating the Effect of Mixtures of Noise Levels

So far, we have seen how to model the noise on a LiDAR scan when the noise level $\sigma_{\mathbf{p}}$ and direction $\mathbf{d}$ are constant within the neighborhood $\mathcal{N}_r(\tilde{\mathbf{p}})$ of point $\tilde{\mathbf{p}}$. As discussed, this happens when considering points captured from a single scan location and with similar reflected intensity. However, in regions with multiple overlapped clouds, noisy points may benefit from taking into account information from points scanned from a different location.

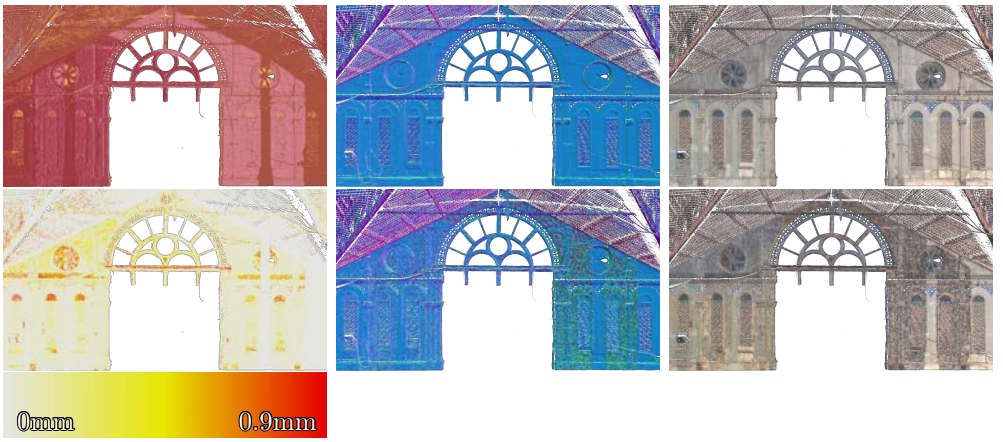


Figure 4.15: Showing the advantages of using the Weighted Least Squares formulation (Equation (4.49) instead of the traditional Least Squares estimation (PCA approach by Hoppe [HDD+92]). From top to bottom: **(1)** Color-coded sensor IDs (location from which each point was captured), normals estimated using PCA on the Weighted Least Squares covariance matrix computed on a k-neighborhood with $k = 15$ and render using these normals. **(2)** Color-coded estimated range error, normals estimated using Hoppe [HDD+92] on a k-neighborhood with $k = 15$ and render using these normals. Notice the correspondence between noisy points and noisy normals.
Points with the lowest noise levels usually come from the closest sensor (colored with (•) in this example). Because of this, the density of these points is usually higher in comparison with points coming from other sensors.
On the one hand, when estimating the normals for low-noise points, the Weighted Least Squares formulation reduces the influence of noisy points. On the other, when estimating the normals for noisy points, the Weighted Least Squares formulation helps enhance the relevance of the information provided by low-noise points and this yields more robust normals.

In order to study the potential benefits of this, we first consider a setting where we use the extended neighborhood $\mathsf{N}_r(\tilde{\mathbf{p}}) = \bigcup_{\mathbf{q}\in Q} \mathcal{N}_r^{\mathbf{q}}(\tilde{\mathbf{p}})$ for points $\tilde{\mathbf{p}}$, namely the union of the points within radius r scanned from the different locations $\mathbf{q} \in Q$.

In this new setting, the points in $\mathsf{N}_r(\tilde{\mathbf{p}})$ may have highly different noise levels, and this violates the Least Squares formulation which assumes homocedasticity (constant noise variance through the samples). When dealing with heteroscedastic data, it is more appropriate to use the Weighted Least Squares (**WLSQ**) formulation. In particular, we want to give higher relevance to more reliable points, and we do this by weighting the *cost* of each point by a value inversely-proportional to its estimated noise level:

$$\begin{aligned} &\underset{\mathbf{n_p},d}{\operatorname{argmin}} \left(\frac{1}{2w} \sum_{\tilde{\mathbf{p}}_i \in \mathsf{N}_r(\tilde{\mathbf{p}})} w_{\tilde{\mathbf{p}}_i} (\mathbf{n}_\mathbf{p}^T \tilde{\mathbf{p}}_i - d)^2 \right) \\ &\text{subject to } ||\mathbf{n_p}|| = 1 \end{aligned} \tag{4.49}$$

where $w = \sum_{\tilde{\mathbf{p}}_i \in \mathsf{N}_r(\tilde{\mathbf{p}})} w_{\tilde{\mathbf{p}}_i}$ and $w_{\tilde{\mathbf{p}}_i} = \exp\left(\lambda_w^2 \sigma_\mathbf{p}^2\right)$ for some constant λ_w. This yields the new centroid and covariance matrix definitions (see derivation later):

$$\mathbf{c}_{\tilde{\mathbf{p}}} = \frac{1}{w} \sum_{\tilde{\mathbf{p}}_i \in \mathsf{N}_r(\tilde{\mathbf{p}})} w_{\tilde{\mathbf{p}}_i} \tilde{\mathbf{p}}_i \tag{4.50}$$

$$\mathbf{CV}(\mathsf{N}_r(\tilde{\mathbf{p}})) = \sum_{\tilde{\mathbf{p}}_i \in \mathsf{N}_r(\tilde{\mathbf{p}})} w_{\tilde{\mathbf{p}}_i} (\tilde{\mathbf{p}}_i - \mathbf{c}_{\tilde{\mathbf{p}}})(\tilde{\mathbf{p}}_i - \mathbf{c}_{\tilde{\mathbf{p}}})^T \tag{4.51}$$

We will refer to this formulation as *Weighted Least Squares* (**WLSQ**). Using PCA on the resulting covariance matrix, we will estimate a new normal for a point $\tilde{\mathbf{p}}$. In Figures **4.15** and **4.16** we show the benefits of this strategy against the traditional PCA approach [HDD+92].

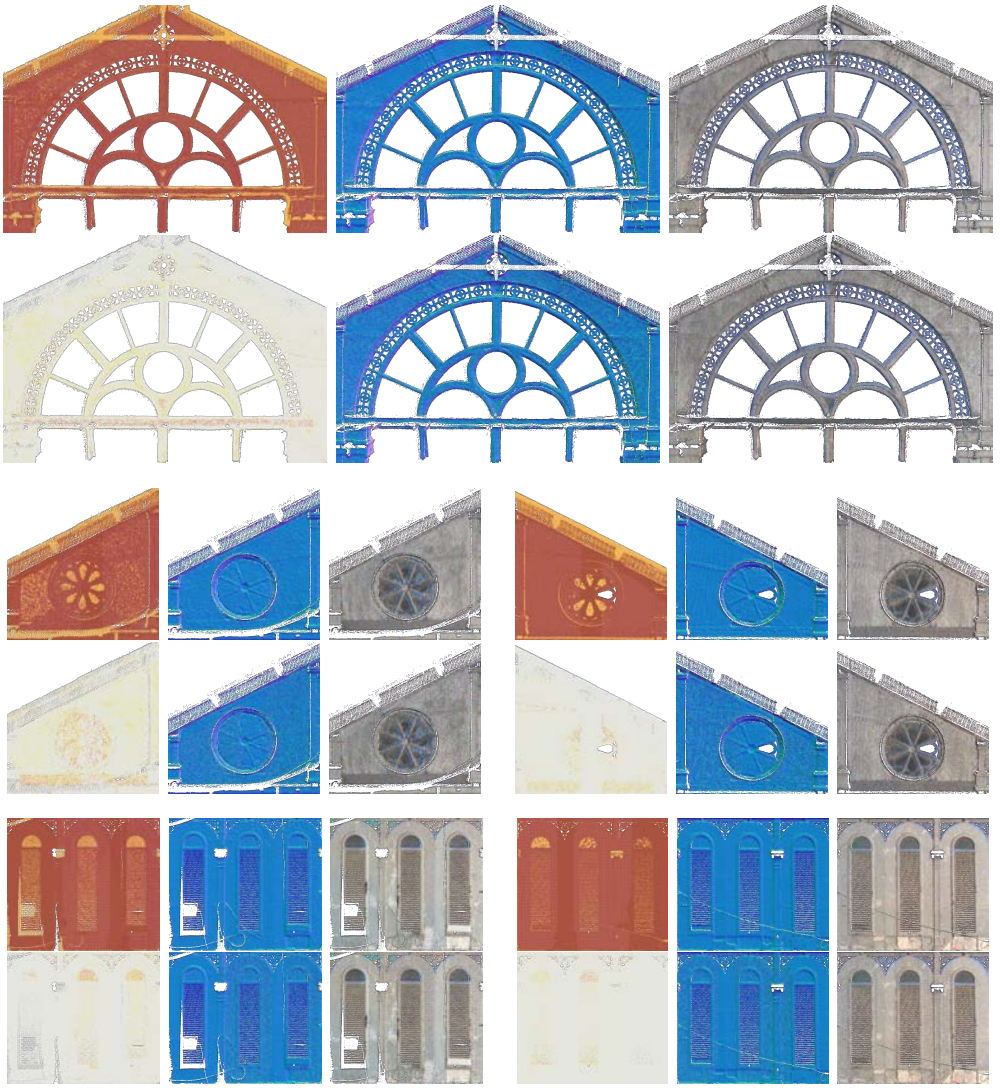


Figure 4.16: Zoomed regions from figure **4.15**. The images are also arranged in the same layout.

Weighted least-squares problem derivation

Taking the derivatives of Equation (4.49):

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{n_p}} \left(\frac{1}{2w} \sum_{\tilde{\mathbf{p}}_i \in \mathsf{N}_r(\tilde{\mathbf{p}})} w_{\tilde{\mathbf{p}}_i} (\mathbf{n_p}^T \tilde{\mathbf{p}}_i - d)^2 \right) - \lambda_L \mathbf{n_p} &= 0 \\
\mathbf{n_p}^T \left(\frac{1}{w} \sum_{\tilde{\mathbf{p}}_i \in \mathsf{N}_r(\tilde{\mathbf{p}})} w_{\tilde{\mathbf{p}}_i} \tilde{\mathbf{p}}_i \tilde{\mathbf{p}}_i^T \right) - d \left(\frac{1}{w} \sum_{\tilde{\mathbf{p}}_i \in \mathsf{N}_r(\tilde{\mathbf{p}})} w_{\tilde{\mathbf{p}}_i} \tilde{\mathbf{p}}_i \right) &= \lambda_L \mathbf{n_p} \\
\mathbf{n_p}^T \left(\frac{1}{w} \sum_{\tilde{\mathbf{p}}_i \in \mathsf{N}_r(\tilde{\mathbf{p}})} w_{\tilde{\mathbf{p}}_i} \tilde{\mathbf{p}}_i \tilde{\mathbf{p}}_i^T \right) - d \mathbf{c}_{\tilde{\mathbf{p}}} &= \lambda_L \mathbf{n_p}
\end{aligned} \tag{4.52}$$

$$\begin{aligned}
\frac{\partial}{\partial d} \left(\frac{1}{2w} \sum_{\tilde{\mathbf{p}}_i \in \mathsf{N}_r(\tilde{\mathbf{p}})} w_{\tilde{\mathbf{p}}_i} (\mathbf{n_p}^T \tilde{\mathbf{p}}_i - d)^2 \right) &= 0 \\
\mathbf{n_p}^T \left(\frac{1}{w} \sum_{\tilde{\mathbf{p}}_i \in \mathsf{N}_r(\tilde{\mathbf{p}})} w_{\tilde{\mathbf{p}}_i} \tilde{\mathbf{p}}_i \right) &= d \\
\mathbf{n_p}^T \mathbf{c}_{\tilde{\mathbf{p}}} &= d
\end{aligned} \tag{4.53}$$

where, λ_L is Lagrangian multiplier used to encode the constraint $||\mathbf{n_p}|| = 1$. Finally, by combining the results from both derivatives:

$$\begin{aligned}
\mathbf{n_p}^T \left(\frac{1}{w} \sum_{\tilde{\mathbf{p}}_i \in \mathsf{N}_r(\tilde{\mathbf{p}})} w_{\tilde{\mathbf{p}}_i} \tilde{\mathbf{p}}_i \tilde{\mathbf{p}}_i^T \right) - \mathbf{n_p}^T (\mathbf{c}_{\tilde{\mathbf{p}}} \mathbf{c}_{\tilde{\mathbf{p}}}{}^T) &= \lambda_L \mathbf{n_p} \\
\mathbf{n_p}^T \left(\frac{1}{w} \sum_{\tilde{\mathbf{p}}_i \in \mathsf{N}_r(\tilde{\mathbf{p}})} w_{\tilde{\mathbf{p}}_i} \tilde{\mathbf{p}}_i \tilde{\mathbf{p}}_i^T - \mathbf{c}_{\tilde{\mathbf{p}}} \mathbf{c}_{\tilde{\mathbf{p}}}{}^T \right) &= \lambda_L \mathbf{n_p} \\
\left(\frac{1}{w} \mathbf{CV}(\mathsf{N}_r(\tilde{\mathbf{p}})) \right) \mathbf{n_p} &= \lambda_L \mathbf{n_p} \\
\mathbf{CV}(\mathsf{N}_r(\tilde{\mathbf{p}})) = \sum_{\tilde{\mathbf{p}}_i \in \mathsf{N}_r(\tilde{\mathbf{p}})} w_{\tilde{\mathbf{p}}_i} \tilde{\mathbf{p}}_i \tilde{\mathbf{p}}_i^T - w \mathbf{c}_{\tilde{\mathbf{p}}} \mathbf{c}_{\tilde{\mathbf{p}}}{}^T &= \sum_{\tilde{\mathbf{p}}_i \in \mathsf{N}_r(\tilde{\mathbf{p}})} w_{\tilde{\mathbf{p}}_i} (\tilde{\mathbf{p}}_i - \mathbf{c}_{\tilde{\mathbf{p}}})(\tilde{\mathbf{p}}_i - \mathbf{c}_{\tilde{\mathbf{p}}})^T
\end{aligned} \tag{4.54}$$

As we formulated a minimization problem, the optimal normal vector estimate is the eigenvector of $\mathbf{CV}(\mathsf{N}_r(\tilde{\mathbf{p}}))$ with the smallest eigenvalue.

4.2.5 Implementation Details for Multiple Materials and Sensor Locations

In Subsection 4.2.2, we have seen how to robustly estimate normals for points with roughly constant noise levels $\sigma_{\mathbf{p}}$ and directions $\mathbf{d}$ on their neighborhood. $\mathbf{d}$ depends on the point coordinates $\tilde{\mathbf{p}}$ and the scanning location $\mathbf{q}$ whereas $\sigma_{\mathbf{p}}$ depends on the technical specifications of the LiDAR device, on the point-to-sensor distance $d(\mathbf{q}, \tilde{\mathbf{p}})$ and on the per-sample reflected intensity i of the last beam (Table 4.1). Hence, we need to cluster together points captured from the same device and location and similar reflective properties. This will give us a set S of clusters of points where these properties are uniform.

In Subsection 4.2.4 we have seen how noisy points may benefit from using information from overlapping clusters of points with a lower noise level. In this section we study how to combine both ideas. We can rewrite Equation (4.51) as:

$$\begin{aligned} \mathbf{CV}(\mathsf{N}_r(\tilde{\mathbf{p}})) &= \sum_{s \in S} w_s \sum_{\tilde{\mathbf{p}}_i \in \mathcal{N}_r^s(\tilde{\mathbf{p}})} (\tilde{\mathbf{p}}_i - \mathbf{c}_{\tilde{\mathbf{p}}})(\tilde{\mathbf{p}}_i - \mathbf{c}_{\tilde{\mathbf{p}}})^T \\ &\approx \sum_{s \in S} w_s \mathbf{CV}(\mathcal{N}_r^s(\tilde{\mathbf{p}})) \end{aligned} \tag{4.55}$$

where $w_s = \exp(\lambda_w^2 \sigma_s^2)$, for some constant λ_w, σ_s is the noise level across cluster $s \in S$ and $\mathcal{N}_r^s(\tilde{\mathbf{p}})$ is the set of points that belong to cluster $s \in S$ within radius r of point $\tilde{\mathbf{p}}$.

Notice that estimating $\mathbf{CV}(\mathcal{N}_r^s(\mathbf{p})$ from $\mathbf{CV}(\mathcal{N}_r^s(\tilde{\mathbf{p}})$ by using Equation (4.18) is not mathematically correct. This is because $\mathbf{c}_{\tilde{\mathbf{p}}}$ was computed on $\mathsf{N}_r(\tilde{\mathbf{p}})$ where noise levels $\sigma_{\mathbf{p}}$ and directions $\mathbf{d}$ are not constant. However, we propose studying this heuristic approximation and later evaluate its effect on the quality of the resulting normals:

$$\mathbf{CV}(\mathsf{N}_r(\mathbf{p})) \approx \sum_{s \in S} w_s \left(\mathbf{CV}(\mathcal{N}_{r_s}^s(\tilde{\mathbf{p}}) - T_1 - T_2) \right) \tag{4.56}$$

Notice that this also implies computing a different radius r_s for each cluster $s \in S$. Intuitively, this should allow recovering more detailed features in regions with very dense clusters with low noise levels (i.e., where the estimated search radius is small).

Suppose the point cloud does not fit in main memory. In this case, our algorithm starts by dividing it into non-overlapping cubic voxels of a feasible size (e.g., $10^3 m^3$) and process each voxel individually while also considering its surrounding 26 neighbors.

Then we build a search structure (a kd-tree) for each cluster in the considered sub-partition (either the whole cloud or a voxel + its 26 neighbors). We iterate through all the considered points and estimate their normals one by one. For a given point $\tilde{\mathbf{p}}$, we estimate one sensor-aware covariance matrix (Equation (4.18)) for each neighborhood from one different cluster and, then, combine these matrices.

Considering an individual cluster, we first estimate the local point density at $\tilde{\mathbf{p}}$ by using the distance to the farthest point on its k-neighborhood (using an initial $k = 15$). Then, we compute r_1 and r_2 using Equation (4.40) and Equation (4.45), and take the largest one as the selected search radius r. We could use this radius to estimate the local point density again and iterate this process. However, we have experimentally found that there is little improvement.

Once we have computed r, we retrieve the neighbors of $\tilde{\mathbf{p}}$ within this radius. We compute the covariance matrix for this set of points and correct it according to Equation (4.23).

At this point, we have one corrected covariance matrix for each cluster surrounding the query point. In Equation (4.56), we have described a heuristic way to combine these matrices. Following, we also describe possible heuristic strategies to derive a single normal out of them (these are evaluated later):

- **S1:** Estimate and select the normal from the cluster, including the query point.
- **S2:** Estimate one normal for each cluster and select the best one according to some criteria. In particular, we pick the normal that, together with the centroid, defines the plane that minimizes the distance to the query point.
- **S3:** Estimate one normal for each cluster and perform a weighted average of these. We use the weights of the form $w_s = \exp(\lambda_w^2 \sigma_s^2)$, for some constant λ_w and where σ_s is the noise level across cluster.
- **S4:** Perform a weighted average of the different covariance matrices (Equation (4.56)) and then use PCA to obtain a single normal.

We discuss and compare these four strategies in the next section.

4.2.6 Results and Discussion

We implemented the proposed normal estimation methods using non-parallel C++ code (although parallelization can be trivially added). In particular, we tested the normal estimation based on Weighted Least Squares (**WLSQ**) and the different strategies introduced in Subsection **4.2.5** (**S1** to **S4**).

We tested the algorithm with both synthetic and real point clouds representing architectural models. Details on their sizes and other properties are provided in the corresponding sections. We use synthetic models because ground-truth normals are available. Thus, we can perform different kinds of numerical evaluation. In particular, we are interested in studying the effect of parameters ϵ and δ in Equation (4.40) and Equation (4.45), as well as the quality of the different heuristic strategies proposed in Subsection **4.2.5**. Finally, we will provide a numerical comparison against competing approaches and a visual evaluation of synthetic and real datasets.

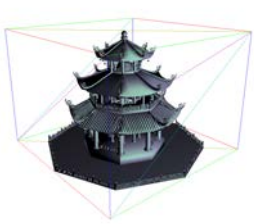
We first evaluate and compare our method using synthetic datasets, for which ground-truth normals are available and thus allow for quantitative comparison. Results with real scanned datasets are discussed afterward.

Synthetic Models

Each scan was simulated by reproducing the sampling pattern and range noise distribution of a high-end LiDAR scanner (Leica P20). For this task, we chose to sample mostly architectural models (Figure **4.17**) since these are predominant in stationary LiDAR scans. For each scan location, we converted the 3D mesh-based models into point clouds by casting rays and regularly sampling the polar angles (θ, ϕ) between $[0, 2\pi)$ and $(\pi/4, \pi)$ respectively. Notice these models include a large number of sharp edges for which we have implemented no special treatment.

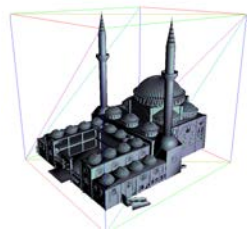
In this case, we generated $10,000 \times 20,000$ samples from each virtual sensor (using up to 6 sensors per model). Figure **4.18** shows the union of points captured from different locations for an example LiDAR synthetic scan. For all models, sensor locations were placed outside the model, except for the Sponza Atrium.

For each ray hit, we stored the 3D position and normal of the intersected mesh surface. We then added Gaussian noise along the line-of-sight. This noise followed a centered normal distribution with a standard deviation of σ. To estimate the appropriate σ for each point, we interpolated the values given by the technical specification of the Leica P20 (Table **4.1**) using the sample's point-to-sensor distance and reflective properties. This process ensures estimated measurement errors to resemble those of real LiDAR equipment closely.

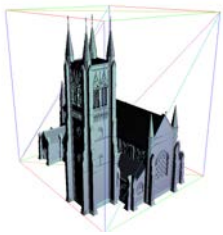


(1) Baidinh

(2) Church

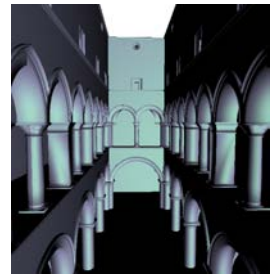


(3) Mosque



(4) Cathedral

(5) Pisa



(6) Sponza atrium

Figure 4.17: Models used to generate synthetic point clouds.

Figure 4.18: Example of synthetic data generated from the Cathedral model. Colors indicate points captured from the same sensor and the spheres are the sensor positions.

Error Metric

Traditional RMS is not appropriate for evaluating normal estimation methods on noisy data. For smooth surfaces, normals estimated on small-amplitude noise is the primary source of error. However, these errors can become hidden in the RMS metric by the presence of other larger ones. These usually happen on sharp features, where the estimated normals have been over-smoothed, or on poorly-sampled areas, where there is not enough information to produce a robust estimate. Thus, this metric favors small r values, which better reproduce local curvature at the expense of noise. Other standard metrics (MAE, SNR, PSNR) reproduce similar behaviors. Hence, they do not respond reasonably to low noise.

Boulch et al. [BM12] introduced the modified RMS_τ metric to compare against ground-truth normals. It sets a common penalization score for normal deviations above a certain threshold τ:

$$RMS_\tau = \sqrt{\frac{1}{N}\sum_{\forall \mathbf{p}} e_{\mathbf{p}}^2} \tag{4.57}$$

$$e_{\mathbf{p}} = \begin{cases} \widehat{\mathbf{n_p}, \mathbf{n'_p}}, & \text{if } \widehat{\mathbf{n_p}, \mathbf{n'_p}} < \frac{\pi}{180}\tau \\ \pi/2, & \text{otherwise} \end{cases} \tag{4.58}$$

where $\mathbf{n_p}$ is the ground-truth normal at $\mathbf{p}$ and $\mathbf{n'_p}$ is the estimated one. We use $\tau = 10$ degrees as proposed by Boulch et al. [BM12], and report normalized RMS_{10}/π values for all experiments.

Tuning ϵ and δ parameters

Equation (4.40) and Equation (4.45) show how to pick a radius r that ensures that the magnitude of the errors is, at most, an ϵ-fraction of the magnitude of the coefficients in $\mathbf{CV}(\mathcal{N}_r(\mathbf{p}))$, with probability δ. This intuition may help to pick reasonable values for these parameters. Moreover, in this section, we will provide further empirical insight.

Let $\gamma = \epsilon^2(1-\delta)$. Notice that r_1 and r_2 depend on γ, as shown in Equations (4.40) and (4.45). In Figure 4.19, we show the effect of γ in terms of RMS_{10}, for the models in Figure 4.17. There is some heterogeneity in the behavior of the error curves for the different models. We think this is caused because our algorithm does not explicitly treat sharp edges (it smooths normals across them). This is a major error source; hence each model's distribution of edges greatly impacts the error. Despite these differences, global minima occur around

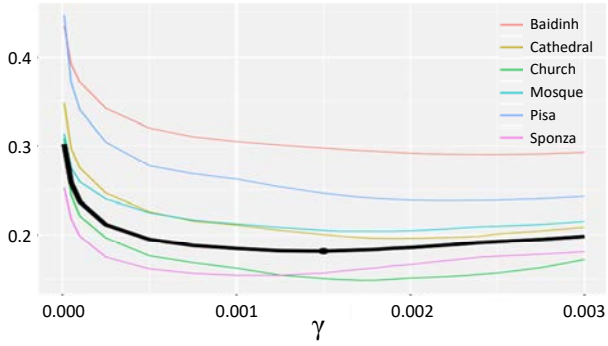


Figure 4.19: RMS_{10} error as a function of γ. The average over all models is shown in black.

$\gamma = 0.0015$, which works well in the average case. This is equivalent to setting, for instance, $\epsilon = 0.1$ and $\delta = 0.85$. Unless explicitly stated, all experiments in this article will use these values.

Strategy selection

We have proposed four different strategies for combining the normal information coming from different point clusters. Table 4.2 compares their accuracy on the synthetic datasets.

	S1	S2	S3	S4
Baidinh	0.2989	**0.2981**	0.3061	0.3004
Cathedral	0.1999	**0.1973**	0.2011	0.2014
Church	0.1506	**0.1485**	0.1634	0.1524
Mosque	**0.2053**	0.2057	0.2134	0.2063
Pisa	0.2464	**0.2454**	0.2574	0.2480
Sponza	0.1569	**0.1550**	0.1554	0.1553

Table 4.2: RMS_{10} obtained using the different strategies for combining normal information from multiple point clusters. The lowest values are highlighted in boldface.

Overall, all strategies seem reasonable. The candidate normals yielded by each of the different methods must meet rigorous error bounds. Thus, reaching a significant quality. Strategies **S2**, **S3**, and **S4** merge information from different clusters, potentially improving normal estimation in regions where a single cloud is poorly sampled. However, these regions are scarce, and their importance on the averaged RMS_{10} scores becomes diluted. This explains why the performance of **S1** is on par. Furthermore, **S3** and **S4** may smooth across edges, which degrades the quality of the estimates. Hence, we decided to pick **S2** for our experiments, which is also the strategy that performed slightly better than the others.

Comparison with competing approaches

We compared our methods **WLSQ** and **S2** with six competing approaches: Hoppe et al. [HDD+92], Mitra et al. [MN03], Merigot et al. [MOG11], Jet fitting [CP05] and Huang et al. [HLZ+09]. We used our own implementation for the first two, and CGAL [14; 13] implementations for the last three.

WLSQ and [HDD+92; HLZ+09; CP05] are sensitive to the chosen k-neighborhood size, therefore we tested multiple k values. Following Andersson et al. [AGP+04b], values between 3 and 50 allow to faithfully approximate local geometric surface properties under the assumption of reasonable sampling parameters and locally uniform sampling. Therefore, we decided to test values $k \in \{10, 25, 50\}$. Furthermore, we added $k = 100$ to account for the effect of large noise values.

For Mitra et al. [MN03] we fixed $\epsilon = 0.1$, as suggested by the authors. In contrast, we found no particular value suggestion for constants c_1 and c_2. As in their reported experiments we let $c_1 = c_2 = c$, and tested the algorithm with $c = \{1, 5, 10, 20\}$.

For Merigot et al. [MOG11] we used $R = 1.6$ and $r = 0.05$ since the authors argue they achieve stable results around these values. Finally, for Huang et al. [HLZ+09] we used a support radius $h = 4\sqrt{d_{bb}/m}$ to consolidate the cloud to a 10% and 90% of the original size, where d_{bb} is the length of the bounding box diagonal, and m is the total number of points. Then, we computed the normals for the original cloud using neighboring consolidated points.

Generally, scanned data lacks ground truth. Thus, assessing the convenience of a chosen k is difficult, and finding the optimal k through trial-and-error would require user intervention.

In Table **4.3** we compare the performance of the different methods in terms of RMS_{10}. As expected, a good choice of neighborhood size k is critical for the methods sensitive to this parameter. Some datasets (e.g., Cathedral) require small neighborhoods ($k = 10$), others (e.g., Church) prefer medium neighborhoods ($k = 25$), whereas one dataset (Sponza atrium) benefits from large ($k = 50, 100$) neighborhoods. The performance of Mitra et al. [MN03] is also closely dependent on the value choice for c. This parameter is closely related to the neighborhood size.

Our **S2** method outperformed all competing approaches. It provided the lowest RMS_{10} score compared to any other method with any of the tested k and c parameters. The **WLSQ** method ranked second, outperforming the remaining methods almost always, for a good choice of k.

Data set	Method	k not used	k=10 c=1	k=25 c=5	k=50 c=10	k=100 c=20
Baidinh	Hoppe et al.		0.3255	0.3460	0.3930	0.4377
	Quadric Jets		0.3496	0.3475	0.3906	0.4374
	WLOP (10%)		0.4643	0.5034	0.5284	0.5637
	WLOP (90%)		0.3306	0.3542	0.4015	0.4440
	Merigot et al.	0.5144				
	Mitra et al.		0.6014	0.3304	0.3484	0.3864
	WLSQ		0.3069	0.3386	0.3854	0.4336
	S2	**0.2981**				
Cathedral	Hoppe et al.		0.2556	0.2398	0.2849	0.3360
	Quadric Jets		0.2690	0.2501	0.2922	0.3414
	WLOP (10%)		0.3452	0.4103	0.4586	0.5109
	WLOP (90%)		0.2541	0.2461	0.2931	0.3440
	Merigot et al.	0.4318				
	Mitra et al.		0.5602	0.2388	0.2586	0.2941
	WLSQ		0.2140	0.2338	0.2810	0.3329
	S2	**0.1973**				
Church	Hoppe et al.		0.2422	0.1989	0.2372	0.2884
	Quadric Jets		0.2603	0.2120	0.2539	0.3094
	WLOP (10%)		0.2993	0.3877	0.4561	0.5247
	WLOP (90%)		0.2388	0.2047	0.2457	0.2980
	Merigot et al.	0.4853				
	Mitra et al.		0.5709	0.2079	0.2272	0.2648
	WLSQ		0.2048	0.1903	0.2280	0.2762
	S2	**0.1485**				
Mosque	Hoppe et al.		0.2838	0.2444	0.2692	0.3022
	Quadric Jets		0.2888	0.2485	0.2764	0.3136
	WLOP (10%)		0.3194	0.3664	0.4073	0.4565
	WLOP (90%)		0.2812	0.2486	0.2744	0.3077
	Merigot et al.	0.3875				
	Mitra et al.		0.5631	0.2390	0.2537	0.2829
	WLSQ		0.2550	0.2425	0.2706	0.3044
	S2	**0.2057**				
Pisa	Hoppe et al.		0.3196	0.3011	0.3527	0.4108
	Quadric Jets		0.3427	0.3135	0.3604	0.4157
	WLOP (10%)		0.4258	0.5004	0.5686	0.6216
	WLOP (90%)		0.3177	0.3081	0.3620	0.4200
	Merigot et al.	0.5013				
	Mitra et al.		0.6198	0.2999	0.3293	0.3787
	WLSQ		0.2922	0.2937	0.3450	0.4060
	S2	**0.2454**				
Sponza	Hoppe et al.		0.8474	0.5435	0.3021	0.1565
	Quadric Jets		0.8446	0.5440	0.3122	0.1679
	WLOP (10%)		0.5078	0.2311	0.2201	0.2480
	WLOP (90%)		0.8491	0.5454	0.3033	0.1660
	Merigot et al.	-				
	Mitra et al.		0.7227	0.1792	0.1709	0.1990
	WLSQ		0.8472	0.5425	0.3015	0.1562
	S2	**0.1550**				

Table 4.3: Error in the estimated normal vectors ($RMSE_{10}$) for the compared methods on the test datasets. For the Sponza atrium model we could not estimate normals with Merigot et al. due to its high memory requirements. Our method achieved the lowest error in all datasets.

For a more robust analysis of the results in Table 4.3, we applied Bayesian analysis [Kru14]. We computed the posterior probability of each method performing better (lower RMS_{10}) than our best method **S2** when randomly choosing the neighborhood size (through k or c). These probabilities were modeled as Bernoulli random variables with uniform Beta prior. Furthermore, we excluded Merigot et al. [MOG11] and WLOP (10%) from the analysis due to their poor performance on the test datasets.

We report the posterior mean and the 95% confidence interval (CI). According to our experiments reported in Table 4.3, our method **S2** is the most likely to achieve the lowest error. Mean posterior probabilities for the other approaches were below 50%: 6% (0%, 11% CI).

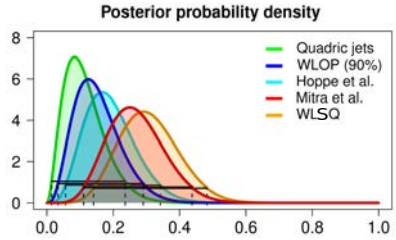


Figure 4.20: Posterior probability of each method providing results similar to our method.

In Figure 4.20, we show the posterior probability of each method performing similarly to **S2**. We neglected RMS_{10} differences below one-third of the standard deviation over all models and methods (0.046). Mean posterior probabilities were:

- 13% (1%, 24% CI) for Quadric Jets [CP05].
- 16% (3%, 28% CI) for WLOP [HLZ+09].
- 20% (6%, 34% CI) for Hoppe et al. [HDD+92].
- 27% (11%, 44% CI) for Mitra et al. [MN03].
- 31% (14%, 48% CI) for **WLSQ**

These methods are more likely to estimate normals with higher deviations (RMS_{10} difference above 0.046) than **S2**. Moreover, the RMS_{10} differences across the integration strategies (**S1** to **S4**, Table 4.2) are far below the 0.046 threshold above.

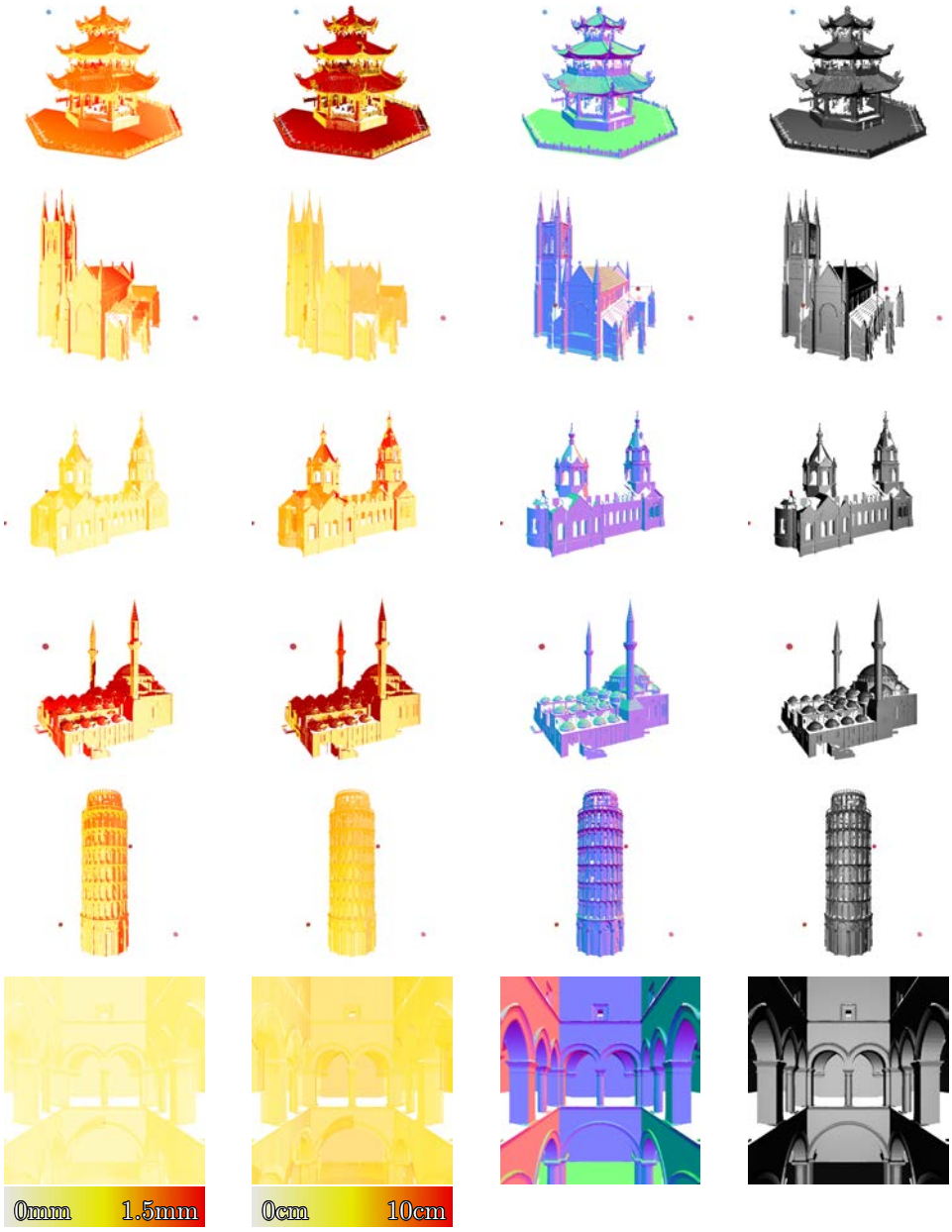


Figure 4.21: Evaluation of our results on synthetic data. From left to right: **(1)** Range error in the input cloud. **(2)** Search radius selected by our approach. **(3)** Estimated normals. **(4)** Render of the point cloud using the estimated normals.

Visual Evaluation

Figure 4.21 displays our normals on synthetic data for a visual evaluation. The first column shows each point's expected range error on top of the models. Line-of-sight occlusions can cause some regions to exhibit abrupt changes in the error values since different sensors capture them (e.g., Cathedral). Other regions exhibit mixtures of error levels caused by points with different point-to-sensor distances or different reflective properties. This situation is usually challenging for most normal estimation methods.

The second column shows the neighborhood radius chosen by our method for each query point. The estimated radii respond to the range error level and sampling changes. For instance, sudden radii changes happen when crossing onto a shadow cast by one sensor (see, e.g., Baidinh). The third and fourth columns try to depict the quality of the estimated normals. Our method achieves compelling normals on flat surfaces and achieves consistent performance across occlusion boundaries, demonstrating its robustness against range noise.

Our method assumes smooth surfaces and does not treat sharp features differently. Nevertheless, our covariance matrix correction is a step that could be integrated into most methods designed to recover sharp edges, potentially achieving both benefits.

Real models

Figures 4.22 to 4.24 visually compare estimated normals on real scanned data. These test sets were obtained with the Leica P20, configured to deliver 3.1mm spacing at 10m. For the competing approaches, the quality of the visual results largely depends on the chosen neighborhood size. Small neighborhoods fail at filtering noise on planar regions, whereas large neighborhoods over-smooth edges and hide high-frequency geometric details. None of the tested parameters (k, c) were satisfactory. This was probably caused by varying surface curvature, sampling density, and range error across the models.

In Figures 4.25 to 4.30 we can see similar results on data captured using the Riegl VZ400 Scanner. This data is provided by Dorit Borrmann and Jan Elseberg from Jacobs University Bremen gGmbH, Germany [12]. Although the noise distribution is completely different (0.5mm at 100m) our method achieves compelling results without specifically tuning any parameter.

Our method adapts the neighborhood radius on a per-sample basis. Thanks to this, it proved successful in removing noise on planar regions while preserving fine relief detail on all test cases.

Although we require knowing the sensor location from which each point sample was acquired, this should not limit the approach's applicability to multi-scan point clouds. Sensor locations are known at acquisition time and also during registration. This information can be preserved by keeping registered scans in separate files. Scan-processing software, such as Leica's Cyclone, can output the registration matrices and preserve the original scan files for further processing.

Performance analysis

We tested all the methods on a commodity PC equipped with an Intel Core i7-4800MQ CPU and 32GB of RAM running Ubuntu 18.04. For synthetic datasets, all models were kept in-core. Real datasets were divided into cubic voxels of $10^3 m^3$ and stored out-of-core. We processed each voxel independently, considering its 26 neighbors for correctness. However, reported times only consider the execution time for the core of the algorithm and omit the time for reading voxels from disk, as this is the same for all algorithms.

Average performance on the test models was about 6.9 Kq/s (thousands of queries —per-sample normal estimations— per second) for our strategy **S2**. This is slower than simpler approaches. For instance performance was about 219 Kq/s for our **WLSQ** approach with $k = 25$, 111.7 Kq/s for our strategy **S1** and 235 Kq/s for [HDD+92]. However, this still allows for estimating robust normal vectors of large (multiple million points) models within minutes.

Out of the other tested methods, only quadric jets obtained better performance (about 52.9 Kq/s) than strategy **S2**. WLOP [HLZ+09] (with 4.4 Kq/s) and Merigot et al [MOG11] (with 2.4 Kq/s) were slower.

We also tested the method by Khaloo and Lattanzi [KL17]. In particular, we used the MM-estimator implementation in the rrcov [TF09] R library. This implementation offered a much slower performance than any other method (0.02 Kq/s). Thus, making it unfeasible to compute normals on our test models.

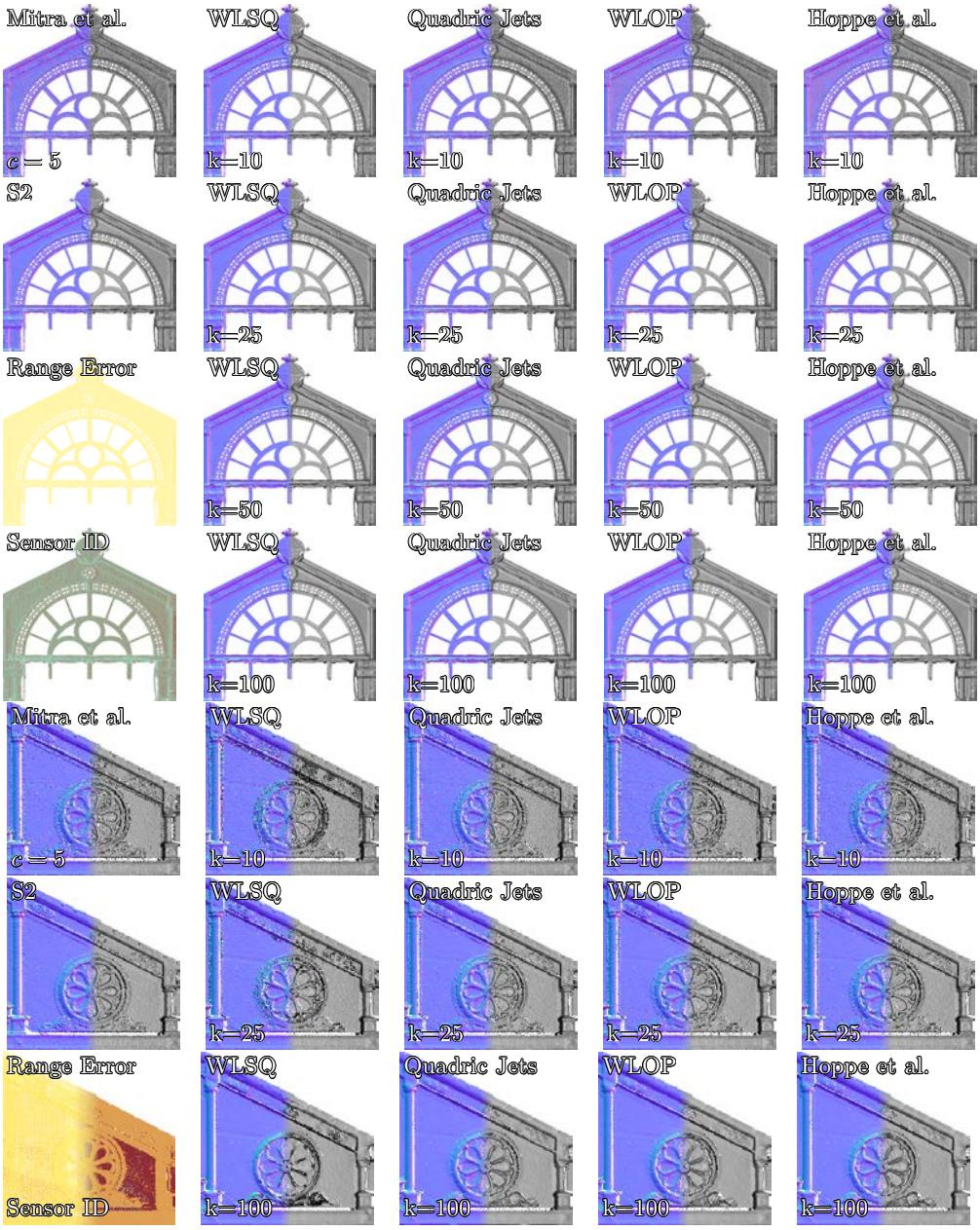


Figure 4.22: Evaluation of our results on real data. Images show estimated normals (left half) and shading (right half). From left to right: different algorithm including **(1)** Mitra et al. [MN03] and our proposed **(S2)** (Covariance Correction) **(2)** Our proposed **(WLSQ)** estimation. **(3)** Quadric Jets [CP05] **(4)** WLOP [HLZ+09] **(5)** Hoppe et al. [HDD+92].

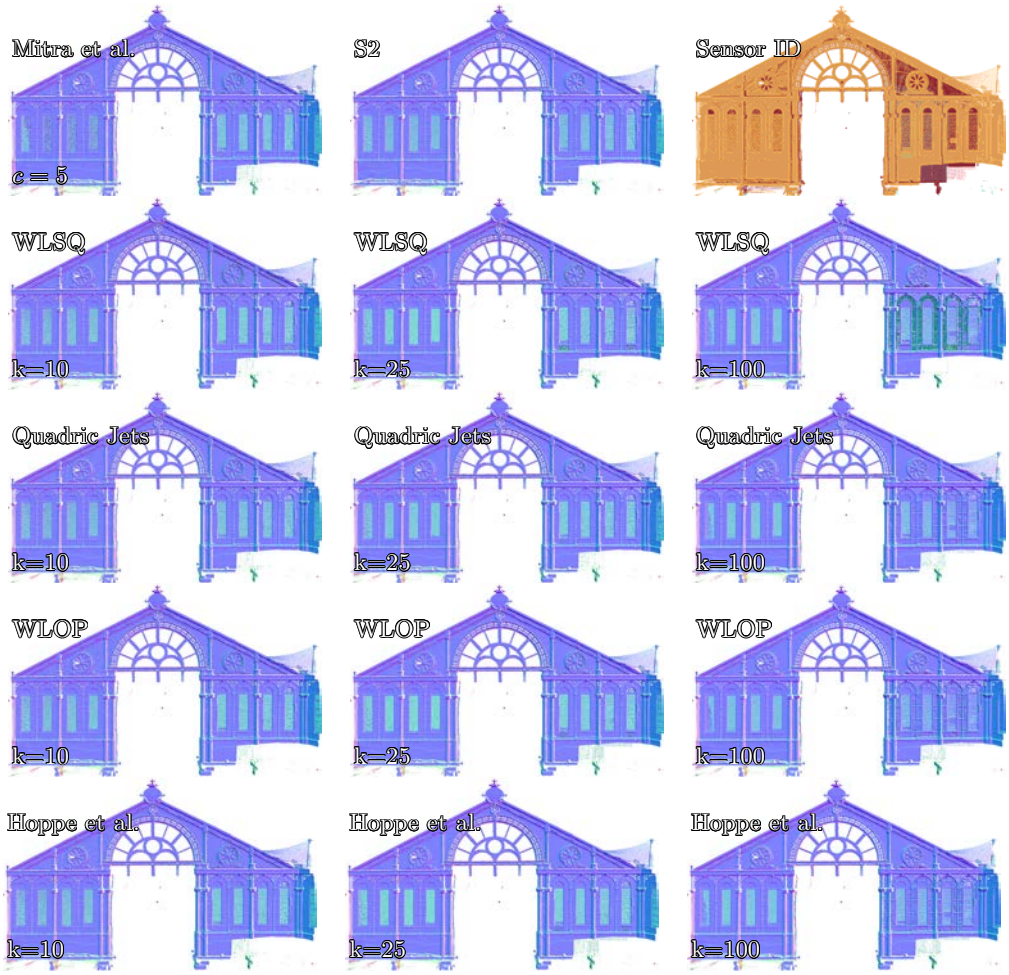


Figure 4.23: Evaluation of our results on real data. Images show estimated normals. From top to bottom: different algorithm including **(1)** Mitra et al. [MN03] and our proposed **(S2)** (Covariance Correction) **(2)** Our proposed **(WLSQ)** estimation. **(3)** Quadric Jets [CP05] **(4)** WLOP [HLZ+09] **(5)** Hoppe et al. [HDD+92].

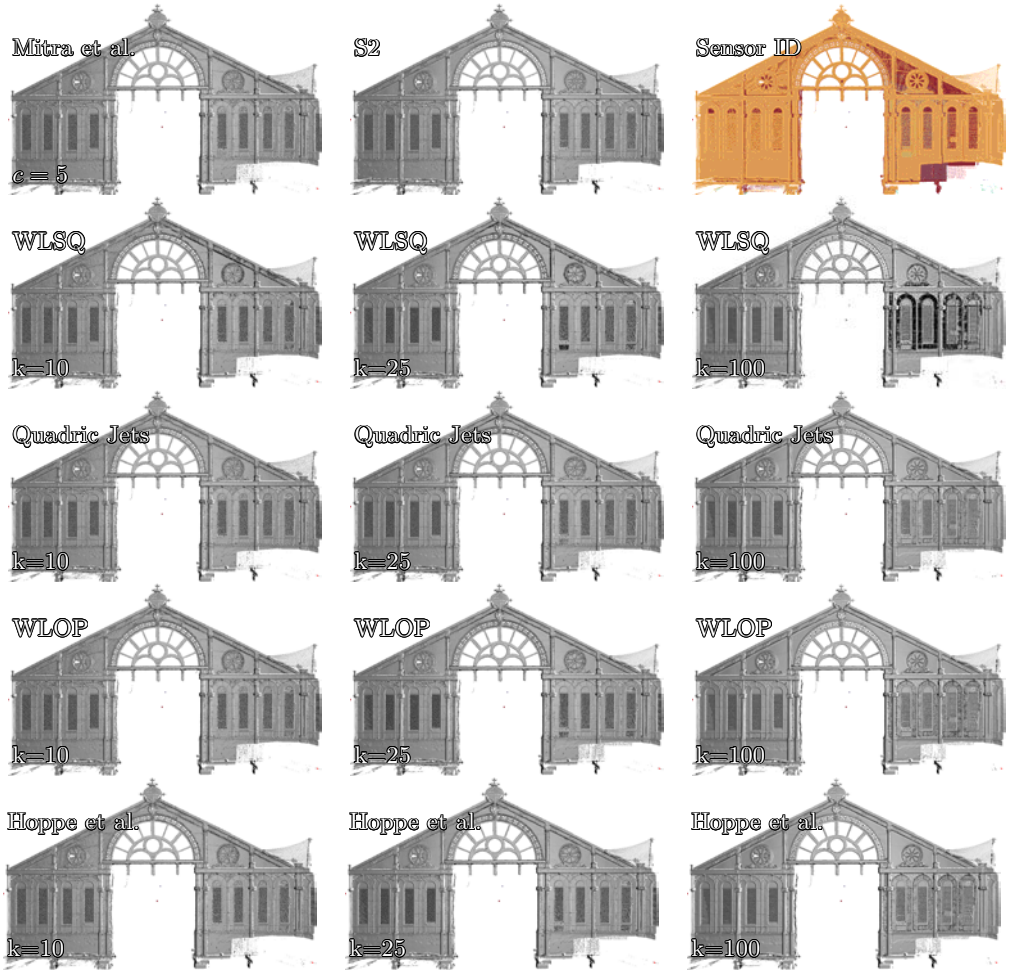


Figure 4.24: Evaluation of our results on real data. Images show shading using estimated normals. From top to bottom: different algorithm including **(1)** Mitra et al. [MN03] and our proposed **(S2)** (Covariance Correction) **(2)** Our proposed **(WLSQ)** estimation. **(3)** Quadric Jets [CP05] **(4)** WLOP [HLZ+09] **(5)** Hoppe et al. [HDD+92].

Figure 4.25: Evaluation of our results on real data. Images show estimated normals. From top to bottom: different algorithm including **(1)** Mitra et al. [MN03] and our proposed **(S2)** (Covariance Correction) **(2)** Our proposed **(WLSQ)** estimation. **(3)** Quadric Jets [CP05] **(4)** WLOP [HLZ+09] **(5)** Hoppe et al. [HDD+92]. This LiDAR point cloud recorded by Dorit Borrmann and Jan Elseberg from Jacobs University Bremen gGmbH, Germany [12]. It was captured using a Riegl VZ400 Scanner, with an accuracy of 0.5mm at 100m.

Figure 4.26: Evaluation of our results on real data. Images show shading using estimated normals. From top to bottom: different algorithm including **(1)** Mitra et al. [MN03] and our proposed **(S2)** (Covariance Correction) **(2)** Our proposed **(WLSQ)** estimation. **(3)** Quadric Jets [CP05] **(4)** WLOP [HLZ+09] **(5)** Hoppe et al. [HDD+92]. This LiDAR point cloud recorded by Dorit Borrmann and Jan Elseberg from Jacobs University Bremen gGmbH, Germany [12]. It was captured using a Riegl VZ400 Scanner, with an accuracy of 0.5mm at 100m.

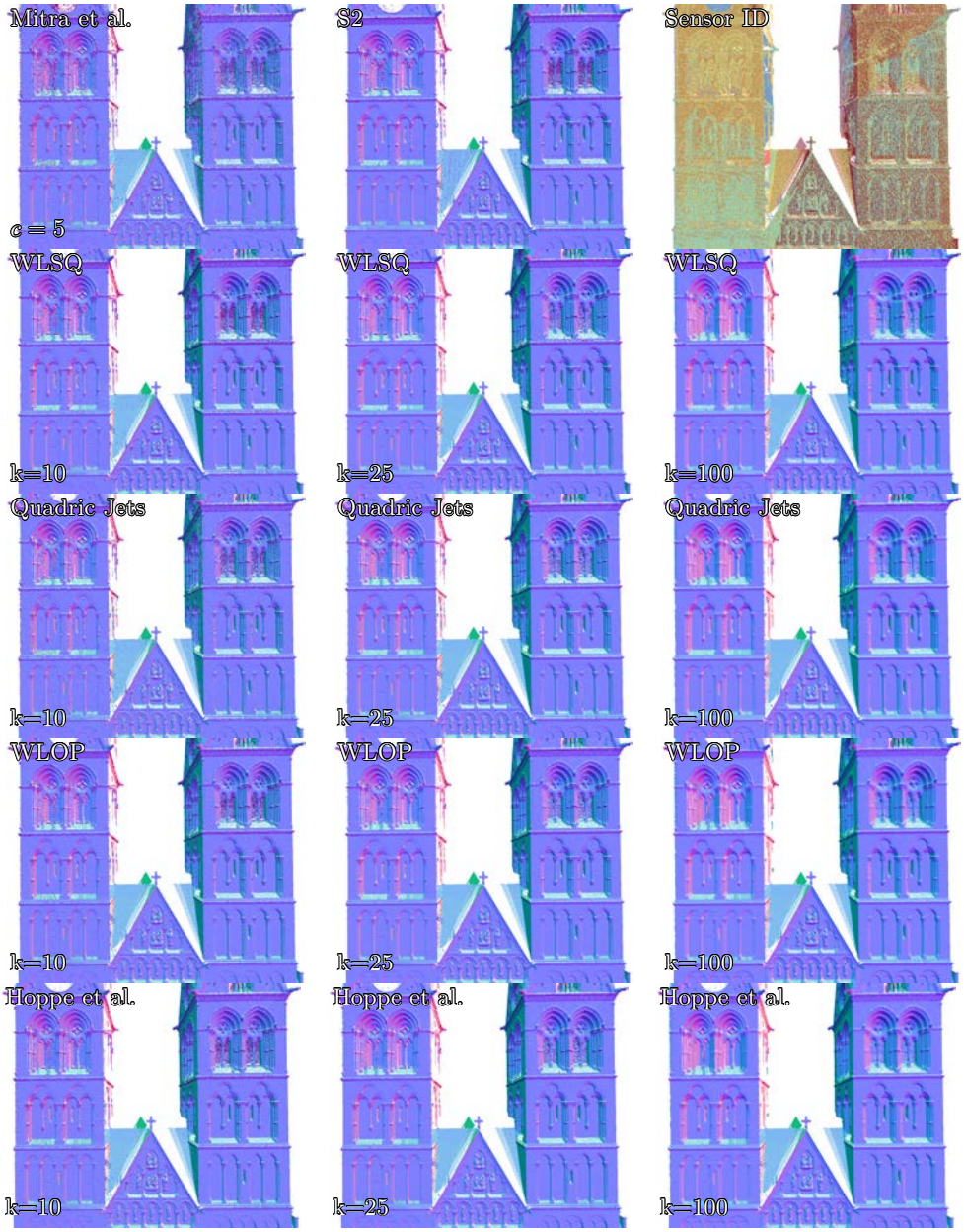


Figure 4.27: Evaluation of our results on real data. Images show estimated normals. From top to bottom: different algorithm including **(1)** Mitra et al. [MN03] and our proposed **(S2)** (Covariance Correction) **(2)** Our proposed **(WLSQ)** estimation. **(3)** Quadric Jets [CP05] **(4)** WLOP [HLZ+09] **(5)** Hoppe et al. [HDD+92]. This LiDAR point cloud recorded by Dorit Borrmann and Jan Elseberg from Jacobs University Bremen gGmbH, Germany [12]. It was captured using a Riegl VZ400 Scanner, with an accuracy of 0.5mm at 100m.

Figure 4.28: Evaluation of our results on real data. Images show shading using estimated normals. From top to bottom: different algorithm including **(1)** Mitra et al. [MN03] and our proposed **(S2)** (Covariance Correction) **(2)** Our proposed **(WLSQ)** estimation. **(3)** Quadric Jets [CP05] **(4)** WLOP [HLZ+09] **(5)** Hoppe et al. [HDD+92]. This LiDAR point cloud recorded by Dorit Borrmann and Jan Elseberg from Jacobs University Bremen gGmbH, Germany [12]. It was captured using a Riegl VZ400 Scanner, with an accuracy of 0.5mm at 100m.

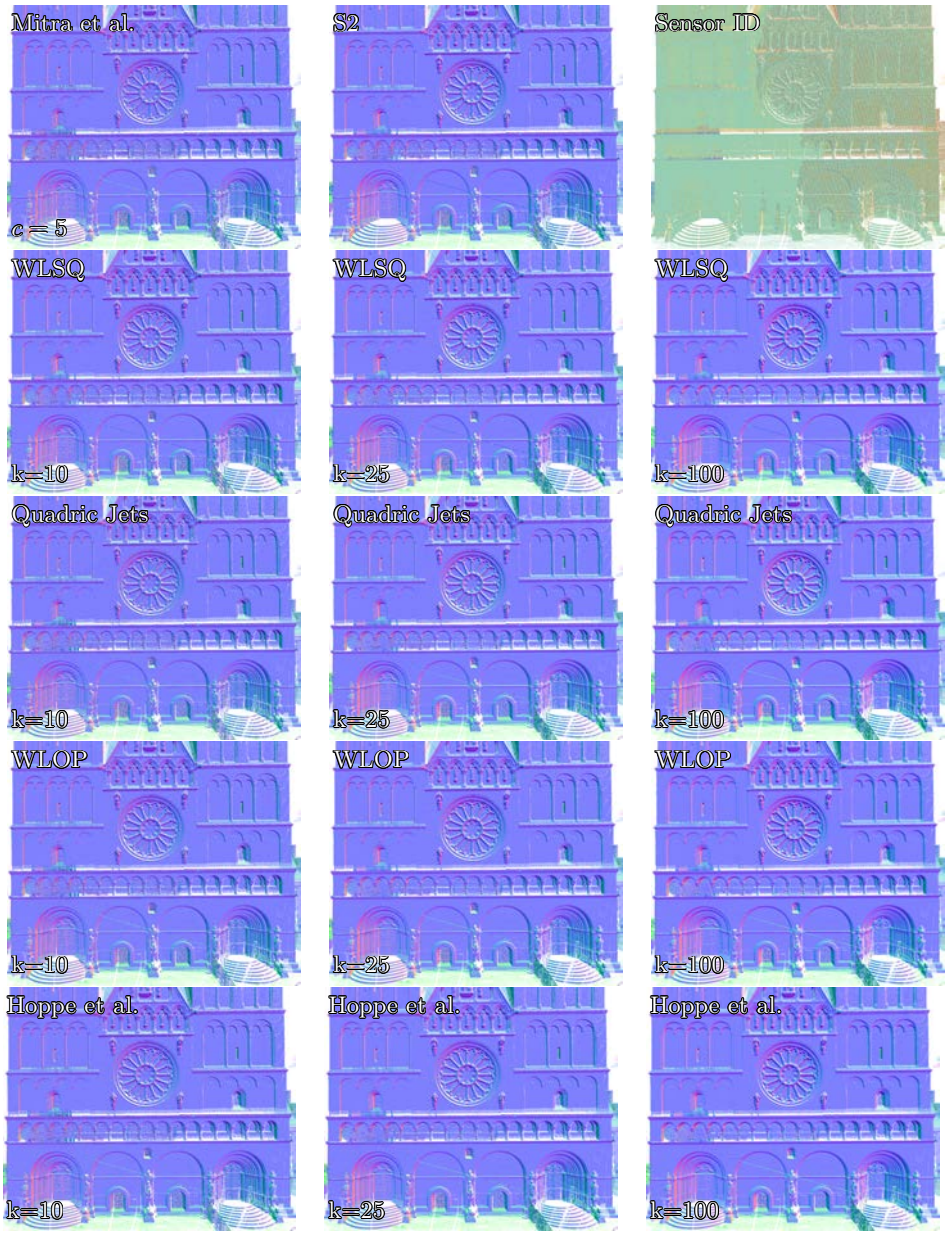


Figure 4.29: Evaluation of our results on real data. Images show estimated normals. From top to bottom: different algorithm including **(1)** Mitra et al. [MN03] and our proposed **(S2)** (Covariance Correction) **(2)** Our proposed **(WLSQ)** estimation. **(3)** Quadric Jets [CP05] **(4)** WLOP [HLZ+09] **(5)** Hoppe et al. [HDD+92]. This LiDAR point cloud recorded by Dorit Borrmann and Jan Elseberg from Jacobs University Bremen gGmbH, Germany [12]. It was captured using a Riegl VZ400 Scanner, with an accuracy of 0.5mm at 100m.

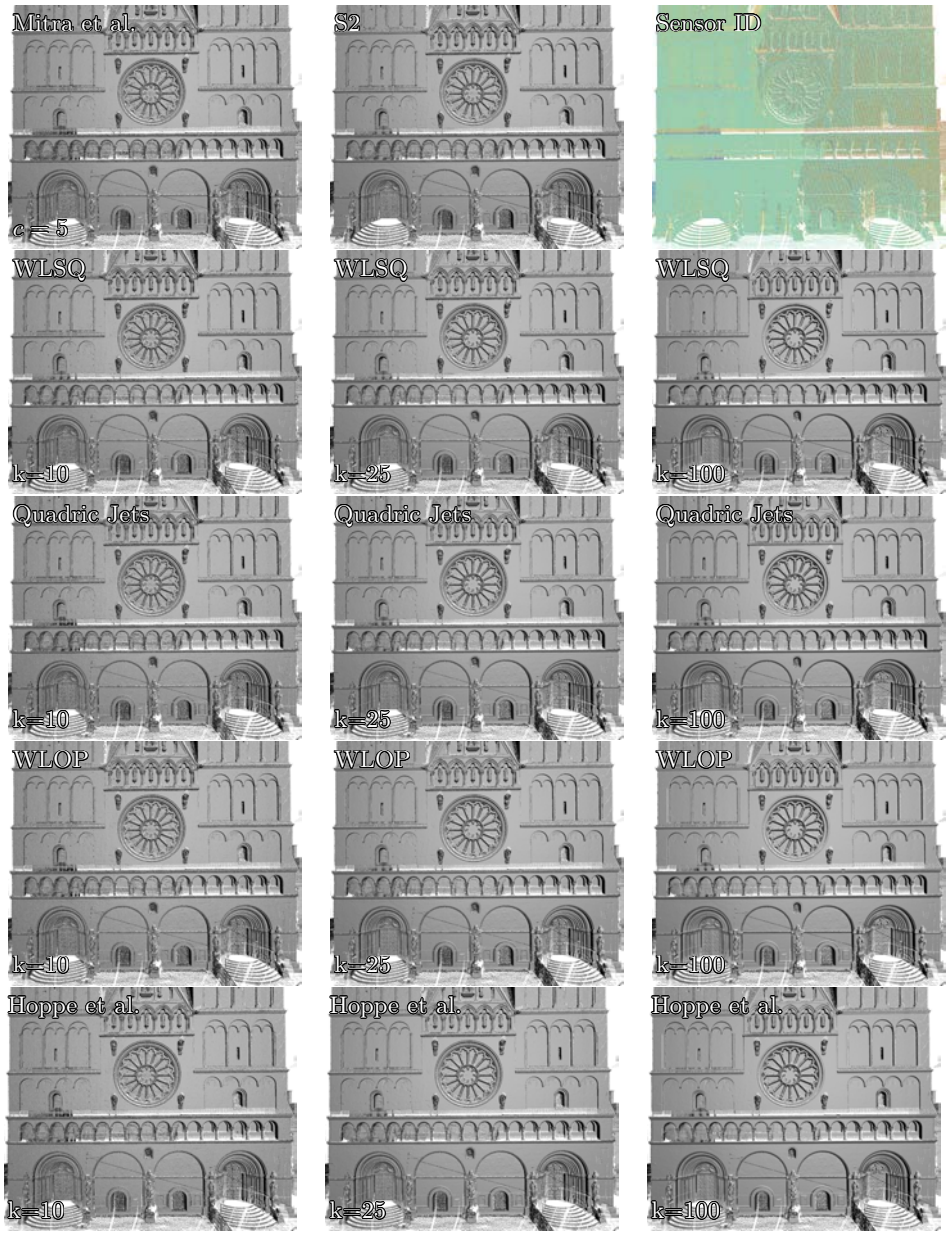


Figure 4.30: Evaluation of our results on real data. Images show shading using estimated normals. From top to bottom: different algorithm including **(1)** Mitra et al. [MN03] and our proposed **(S2)** (Covariance Correction) **(2)** Our proposed **(WLSQ)** estimation. **(3)** Quadric Jets [CP05] **(4)** WLOP [HLZ+09] **(5)** Hoppe et al. [HDD+92]. This LiDAR point cloud recorded by Dorit Borrmann and Jan Elseberg from Jacobs University Bremen gGmbH, Germany [12]. It was captured using a Riegl VZ400 Scanner, with an accuracy of 0.5mm at 100m.

4.3 LOW-STRETCH PANORAMIC REPRESENTATION

In Section 4.1 we have developed the need for simplification on massive LiDAR scans. These are typically composed of a collection of clouds captured from multiple locations and can amount to billions of points. In Section 4.2, we have seen how to estimate normal information in a robust way for this type of cloud. We have also seen how this information can be used to preserve geometric detail during the simplification process. However, different point properties have different spatial distribution across the cloud, and we are bound to lose high-frequency information during the decimation.

One clear example is color information. Color is not usually correlated with geometric detail and, if the simplification process is optimized to preserve the latter, it is unlikely that it will conserve the former.

We want a method that will allow us to augment the simplified models with the lost detail. Taking into account these characteristics, we formulate the following desired properties:

- **Resolution-Awareness:** The main benefit of the simplification process is generating a light-weight enough representation for tasks such as real-time inspection or surface reconstruction. We want a method to recover high-frequency detail to enhance our simplified clouds without sacrificing these benefits.

  We propose encoding and storing this detail information into textures. While this technique is widespread for mesh representations, it has been very little studied for point clouds. One benefit this representation offers is a clear trade-off between resolution and consumed resources. There is a large number of traditional methods to decrease texture resolution. Moreover, nowadays, there are also multiple convolutions neural networks-based approaches for the opposite task, i.e., increase their resolution.

- **GPU-Friendliness:** Increasing the detail of these points clouds during rendering needs to be done in a way that does not hinder the system's performance.

  Nowadays, GPUs and drivers have extensive support for textures, making them an ideal representation for rendering tasks.

- **Storage-Efficiency:** The storage space needed for a cloud is decimated at the same rate the clouds are simplified. Nevertheless, we require keeping the property detail at the original resolution, which may have an expensive storage cost.

Using textures allows us to use already-available compression algorithms. Moreover, lossless formats (such as *TIFF* or *PNG*) can be used to store sensitive information (such as normals or other geometric properties). In contrast, lossy and more aggressive compression (such as *JPEG*) can be used for less critical information (such as color).

- **Adaptability:** Like simplification algorithms, the number of resources invested in storing detail should be proportional to the represented surface. For concentric LiDAR point clouds, this means storing similar detail resolution at regular solid angles.

  As we have seen, however, LiDAR data is not usually captured at regular solid angles (Figure 4.1). Hence, we propose using the stretch-invariant polar capped mapping [SM01] to encode properties into textures since it has the advantage of minimizing the distortion across every region of the unit sphere.

- **Faithfulness:** The recovered information must faithfully represent the original detail. As we have previously argued, using different image encoding algorithms (lossy or lossless) and different image resolutions, we can control the faithfulness level of the stored information. Moreover, we study how to correctly interpolate point information into textures to preserve the original sharpness.

- **Editability:** This is a direct advantage of the texture representation. Editing properties, such as color, directly on the point cloud requires much effort. Doing so on top of textures enables the use of already-available editing tools.

Encoding properties into textures is a widely adopted practice for mesh representations. We extend the idea to point clouds by taking advantage of how concentric LiDAR data is captured, i.e., each point maps to unique polar coordinates (there are no occlusions) from the sensor location. We process each cloud in a LiDAR scan individually and generate one different texture for each cloud and desired property. Typical point properties are normals, depth, colors, and IR intensity. There are also more elaborated properties like luminance coefficients (which can be computed since, effectively, a single cloud is equivalent to a concentric depth map).

4.3.1 Problem Formulation

Consider a point cloud C_Q which results from the union of a set registered scans from locations $Q \subset \mathbb{R}^3$, namely $C_Q = \bigcup_{\mathbf{q} \in Q} C_{\mathbf{q}}$. We want to generate a set of textures $T_{\mathbf{q}}$ encoding certain properties of scan $C_{\mathbf{q}}$. We assume points $\mathbf{p} \in C_{\mathbf{q}}$ contain at least 3D position information and other properties, which can be given as input or computed. Namely $\mathbf{p} = (x, y, z, p_0, ..., p_n)$.

Given a budget of resources $\mathcal{B}$ and a loss function $\mathcal{L}$ we want to compute $T_Q = \bigcup_{\mathbf{q} \in Q} T_{\mathbf{q}}$ such that:

$$\underset{T}{\operatorname{argmin}} \left(\mathcal{L}((C_Q, \emptyset), (C'_Q, T_Q)) \right) \tag{4.59}$$

where $C'_Q \subset C_Q$ is a simplification of C_Q for some factor λ. The optimization process is constrained not to use resources that exceed budget $\mathcal{B}$.

For instance, $\mathcal{L}$ could be a function that measures the error of rendering C'_Q using splats to cover the same surface as C_Q. In this case, $\mathcal{B}$ could be the amount of storage needed to store T_Q or the amount of time needed to render C'_Q.

The need for computing T_Q is clear when rendering C'_Q with flat splats. This is homologous to using nearest-neighbor interpolations, hence discarding all high-frequency detail (Figure 4.31).

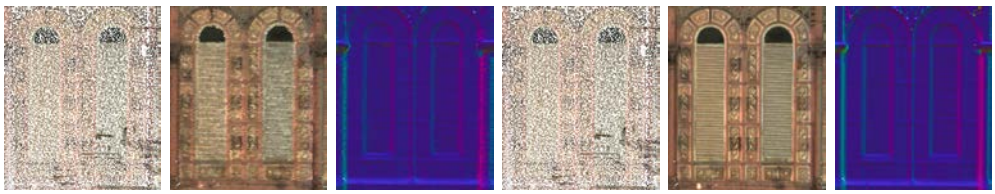


Figure 4.31: From left to right: **(1,2,3)** Renders of a simplified point cloud using simple splats. **(4,5,6)** Analogous render using textured splats. On **(2,4)** we increase the splatting radius to cover the same surface as the original cloud. Regular splats appear completely flat and the high frequency detail is lost, whereas textured splats are able to reproduce the high frequency features for both color **(5)** and normal data **(6)**.

Particularly, we propose computing $T_{\mathbf{q}}$ by representing $C_{\mathbf{q}}$ using an spherical representation $P_{\mathbf{q}}$ and mapping point properties to texture coordinates using the map $f_{P_{\mathbf{q}}} : \mathbb{R}^3 \to \mathbb{R}^2$. We can also define the inverse map $f^{-1}_{P_{\mathbf{q}}} : \mathbb{R}^2 \to \mathbb{R}^3$ to map texture coordinates to 3D points in the unit sphere.

An spherical representation $P_{\mathbf{q}}$ of a point cloud $C_{\mathbf{q}}$, generated from $\mathbf{q}$, is always a valid discrete representation of $C_{\mathbf{q}}$ because occlusions in P are identical to occlusions in $C_{\mathbf{q}}$. Typical terrestrial LiDAR scanners use a cylindrical plane-chart projection to define each infrared pulse ray direction. However, this projection yields highly-uneven sample densities across the unit sphere. Therefore, following Snyder et al. [SM01], we propose using a stretch-invariant polar capped map to store the point clouds' detail information.

The stretch-invariant polar capped map was designed to be an optimal two-component map that minimizes distortion across every unit sphere region. Let ψ and θ represent latitude/longitude angles on a unit sphere. Points whose spherical coordinate ψ is in the range of $[-\pi/4, \pi/4]$ (equator) are encoded using a plane-chart cylindrical projection. Two azimuthal equidistant maps are using for the poles. One for points with $\psi \in [-\pi/2, -\pi/4)$ and another one for points with $\psi \in (\pi/4, \pi/2]$. This representation is shown in Figure 4.32.

Figure 4.32: Stretch-invariant polar capped color map for a given scan.

4.3.2 Mapping Coordinates

Given a 3D point $\mathbf{p} \in C_{\mathbf{q}}$, we can compute its texture coordinates by using the forward mapping $f_{P_{\mathbf{q}}}$. Let $(\bar{x}, \bar{y}, \bar{z})$ be the vector obtained by normalizing the vector from $\mathbf{q}$ to $\mathbf{p}$, and let ψ be $\arcsin(\bar{z})$. The texture coordinates for P can be computed as follows.

For the bottom cap ($\psi \leq -\pi/4$), $r_b = \frac{2}{\pi}\arccos(-\bar{z})$ and

$$f_{P_{\mathbf{q}}} = \left.\begin{cases} s = (\bar{x} r_b / \sin(\pi r_b/2) + 0.5)/6 \\ t = (\bar{y} r_b / \sin(\pi r_b/2) + 0.5)/6 \end{cases}\right\} \psi \leq -\pi/4 \tag{4.60}$$

Similarly, for the top cap ($\psi > \pi/4$), $r_t = \frac{2}{\pi}\arccos(\bar{z})$ and

$$f_{P_\mathbf{q}} = \left.\begin{cases} s = (\bar{x} r_t / \sin(\pi r_t/2) + 1.5)/6 \\ t = (\bar{y} r_t / \sin(\pi r_t/2) + 0.5)/6 \end{cases}\right\} \psi > \pi/4 \tag{4.61}$$

Finally, for the band across the equator ($-\pi/4 < \psi <= \pi/4$) we set $\theta = \arctan(\bar{y}, \bar{x})$ and get the texture coordinates using

$$f_{P_\mathbf{q}} = \left.\begin{cases} s = (\theta/(2\pi) + 0.5)/1.5 \\ t = (\psi + \pi/4)/(2\pi) \end{cases}\right\} -\pi/4 < \psi <= \pi/4 \tag{4.62}$$

Following, we define the inverse map $f^{-1}_{P_\mathbf{q}}$ from texture coordinates (s,t) to 3D points on the unit sphere. Let u_b be $6s-0.5$, v_b be $t-0.5$ and $r_b = \sqrt{u_b^2 + v_b^2}$. For the bottom cap ($s \leq 1/6$), the direction (x,y,z) encoded by a texel (s,t) is given by

$$f^{-1}_{P_\mathbf{q}}(s,t) = \left.\begin{cases} x = (u_b/r_b)\sin(\pi r_b/2) \\ y = (v_b/r_b)\sin(\pi r_b/2) \\ z = -\cos(\pi r_b/2) \end{cases}\right\} s \leq 1/6 \tag{4.63}$$

Similarly, for the top cap ($1/6 < s \leq 2/6$) we define $u_t = 6s-1.5$, $v_t = t-0.5$ and $r_t = \sqrt{u_t^2 + v_t^2}$, and get the direction (x,y,z) by

$$f^{-1}_{P_\mathbf{q}}(s,t) = \left.\begin{cases} x = (u_t/r_t)\sin(\pi r_t/2) \\ y = (v_t/r_t)\sin(\pi r_t/2) \\ z = \cos(\pi r_t/2) \end{cases}\right\} 1/6 < s \leq 2/6 \tag{4.64}$$

Finally, for the band across the equator ($s > 2/6$), we set $\psi = t\pi/2 - \pi/4$, $\theta = (1.5s-0.5)2\pi$ and get the direction using a simple equirectangular projection:

$$f^{-1}_{P_\mathbf{q}}(s,t) = \left.\begin{cases} x = \cos(\psi)\cos(\theta) \\ y = \cos(\psi)\sin(\theta) \\ z = \sin(\psi) \end{cases}\right\} s > 2/6 \tag{4.65}$$

4.3.3 Color Estimation and Enhancement

We have seen how LiDAR scanners capture 3D information by measuring time-of-flight, i.e., the time an infrared beam takes to bounce on a given surface and return to the sensor. Consequently, position and IR intensity information are captured at the same time and are reliably registered. This is, however, not the case for color information. More specifically, color is captured not during the scanning process but after this has finished.

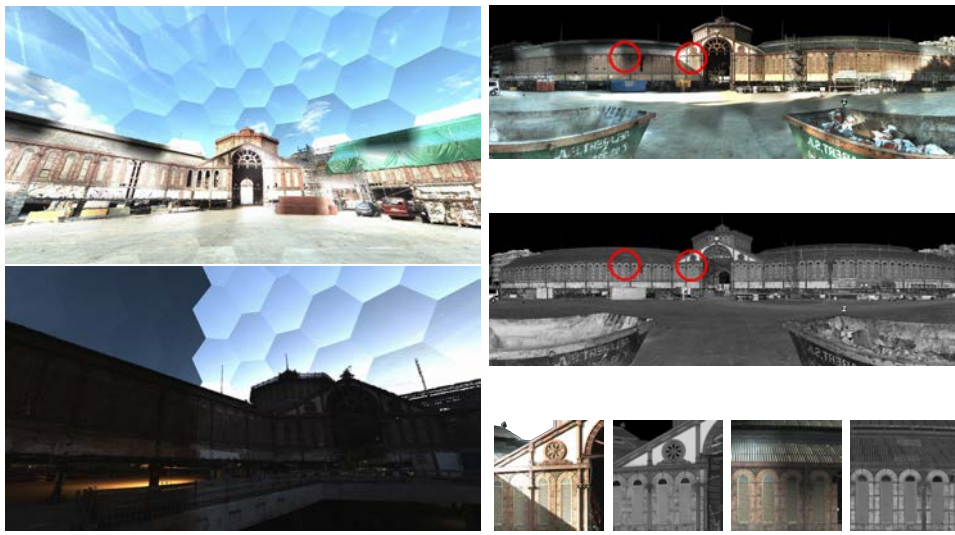


Figure 4.33: Showing the color artifacts produced by the Leica P20. Left: Image depicting the spherical tiling used by the scanner to arrange individual photographs. For each individual photograph, the exposure has been adjusted independently, which causes sharp discontinuities. Moreover, we can also appreciate the change in lighting conditions during the capture process (before and after the sunset). Right: The tiling artifacts appear in the generated panoramas. Color discontinuities and shadowed regions are clearly visible. Nevertheless, the reflected IR intensity captured by the sensor is less correlated with external lighting conditions. These values mostly depend on the reflectivity for the laser beam infrared wavelength. Hence, they are more coherent across a single scan and between different scans.

Different intrinsic and extrinsic factors introduce artifacts in the captured color. On the extrinsic side, the photo acquisition process can take several minutes, and the general illumination of the scene may change after some time. While we can usually control indoor scenes' illumination, we can hardly do the same for outdoor scenes. In Figure 4.33, we can see an example where the sun sets while color is being captured. In Figure 4.40, we can see an example of an outdoor scene captured on a cloudy day (diffuse illumination is generally for photography) with different ambient light intensities.

Figure 4.34: Showing the color artifacts produced by the Leica RTC 360. This system captures HDR images, which greatly improves color consistency. Nevertheless, some sharp color discontinuities can still be seen in these images.

On the intrinsic side, the scanners capture color by taking a mosaic of pictures that cover the unit sphere (Figure **4.33**). Then these images are sampled to obtain the color for each point. Color cameras usually adjust the total amount of light the sensor receives by tuning settings such as exposure time, lens aperture, or shutter speed. If the settings are adjusted for each photograph, the mosaic approach induces sharp color discontinuities. This is especially significant in the Leica P20 (Figure **4.33**). Newer devices, such as the Leica RTC 360, take several pictures with different settings and generate an HDR image. This partially solves the problem, but some discontinuities still appear (Figure **4.34**). These images may contain more general artifacts such as clipped highlights and lens flares (Figure **4.35**).

Figure 4.35: Showing the color artifacts produced by the Leica RTC 360. Lens flares typically appear on images captured in scene with very bright sources. These are usually caused by the internal light refraction and scattering on imperfections of the lenses. In these images we can see several lens flares due to the scanner mosaicing strategy, namely composing several images into a single one.

Finally, the fact color and geometry are not captured simultaneously introduces two types of artifacts caused by moving objects (Figure 4.36). The first type could define as "false negatives", namely objects present in the geometry but are not captured in color. These are very hard to detect visually and require a close inspection of the geometry. The second type could be defined as "false positives", namely objects captured by the color not present in the geometry. These are relatively easy to detect while visually inspecting the geometry as they usually appear as objects glued into walls or floors.

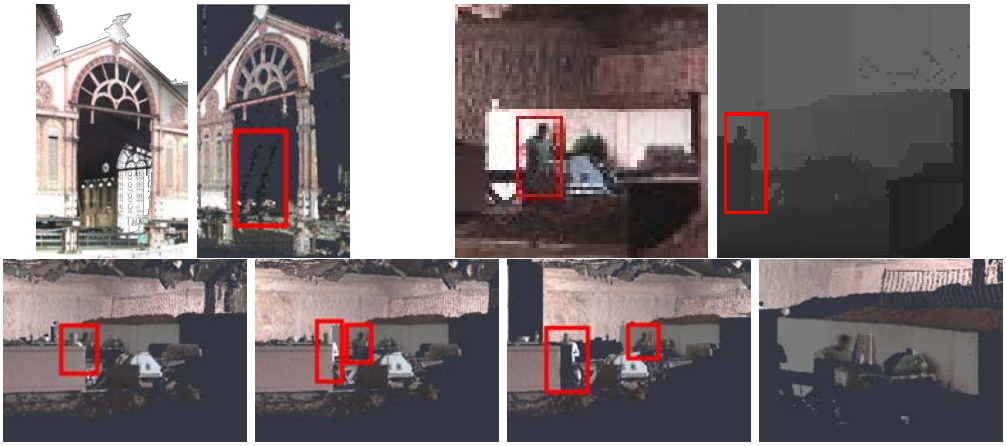


Figure 4.36: First row, from left to right: **(1,2)** Respectively, color image and render of the same cloud. In the render we can see a floating object that is not present in the color image. **(3,4)** Color and depth mismatch due to motion. Second row, from left to right: **(5,6,7)** Cloud renders from slightly different view points. The mismatch between geometry and color can be clearly seen. **(8)** Not a Bansky painting. This is a "false positives" artifact, a person captured only in the color images that has been *pasted* onto a wall.

One interesting fact is that the IR information captured by the sensor is much more robust to illumination conditions than color (Figure 4.33). Moreover, after radiometric calibration, it should mostly depend only on the reflected material. Hence, it should be similar across different scans for points representing the same surface.

We exploit this property and propose a simple yet powerful method for color enhancement. In order to improve the consistency between colors associated to points, we first compute the intensity $i_{rgb} = 0.2126 * r + 0.7152 * g + 0.0722 * b$ of the color and, then, we substitute this scan reported intensity i, e.g. the output red channel is computed as $r * i / i_{rgb}$.

4.3.4 Implementation Details

One efficient approach to implement the panorama generation is using the forward map f_{P_q}, in a vertex shader, to first convert 3D point coordinates to texture coordinates (s, t) (in the range $(0, 1)$) and, afterwards, to clip coordinates (in the range $(-1, 1)$). Conventional splatting techniques can be used to generate the final textures. We have implemented this in a GUI application (Figure 4.37), which allows us to interactively tune the parameters for methods that compute properties on top of the raw points (i.e., ambient luminance coefficients). Nevertheless, this tool also offers a command-line interface that can be used for offline generation of property maps for massive point clouds. In this case, we split these point clouds into cubic cells (voxelization), which are loaded and processed individually.

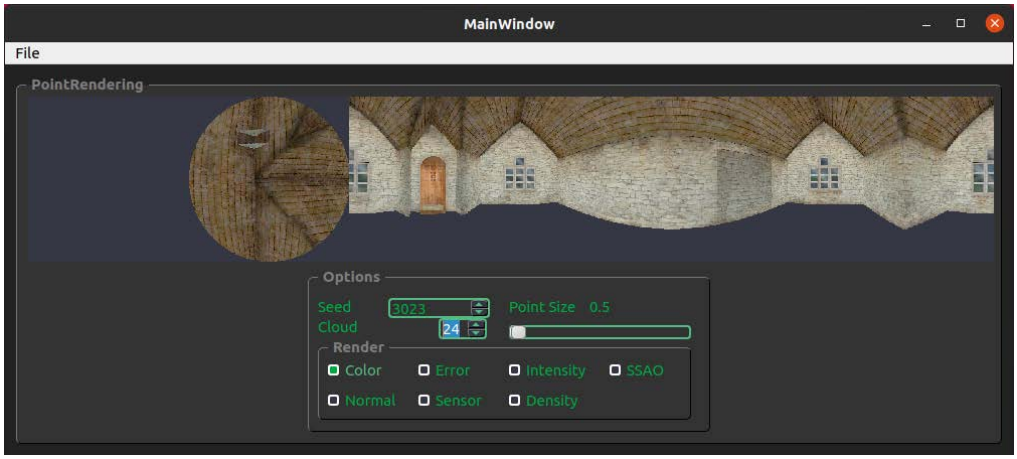


Figure 4.37: GUI application for the generation of stretch-invariant polar-capped maps from point clouds. Points are rendered normally and a vertex shared maps them to their corresponding texture coordinates (s, t), and these to clip coordinates.

We have estimated that, at maximum resolution, the Leica P20 can capture at most $2^{15} \times 2^{14}$ points, whereas we know that the Leica RTC 360 can capture at most $1.25 * 2^{14} \times 2^{13}$ points. Moreover, for the latter, we also know that the maximum color resolution is about $2^{13} \times 2^{12}$ pixels. Hence, these approximated texture sizes must be used to map exactly one point to one pixel. For instance, for a target resolution of $2^{13} \times 2^{12}$ pixels we would generate a stretch-invariant polar capped map of $1.5 * 2^{13} \times 2^{11}$ (see Figure 4.32). With modern hardware, it is feasible to generate textures of these sizes using frame buffer objects. Nonetheless, we have also developed a CPU-only algorithm, convenient when a dedicated GPU is not available.

To generate panoramas in the CPU, we first construct a kd-tree with the points from C mapped to the unit sphere centered at $\mathbf{q}$. Following, we traverse the pixels of the output texture and, for each of pixel with normalized coordinates (s,t) we compute its mapping to the unit sphere using $\mathbf{p}_{(s,t)} = f^{-1}_{P_\mathbf{q}}(s,t)$. We perform a radius search on the kd-tree around the point $\mathbf{p}_{(s,t)}$ using a radius inversely proportional to the output resolution. This process is analogous to performing a cone search on the original cloud. Also, by controlling the size of the search radius r, we control the points' spatial influence, which is analogous to using a splat radius. After retrieving the neighbors $\mathcal{N}_r(\mathbf{p}_{(s,t)})$, we perform a weighted average of these to compute the desired property. We use bilateral weights to avoid over-smoothing, and use the closest point $\mathbf{p} \in \mathcal{N}_r(\mathbf{p}_{(s,t)})$ to $\mathbf{p}_{(s,t)}$ as anchor.

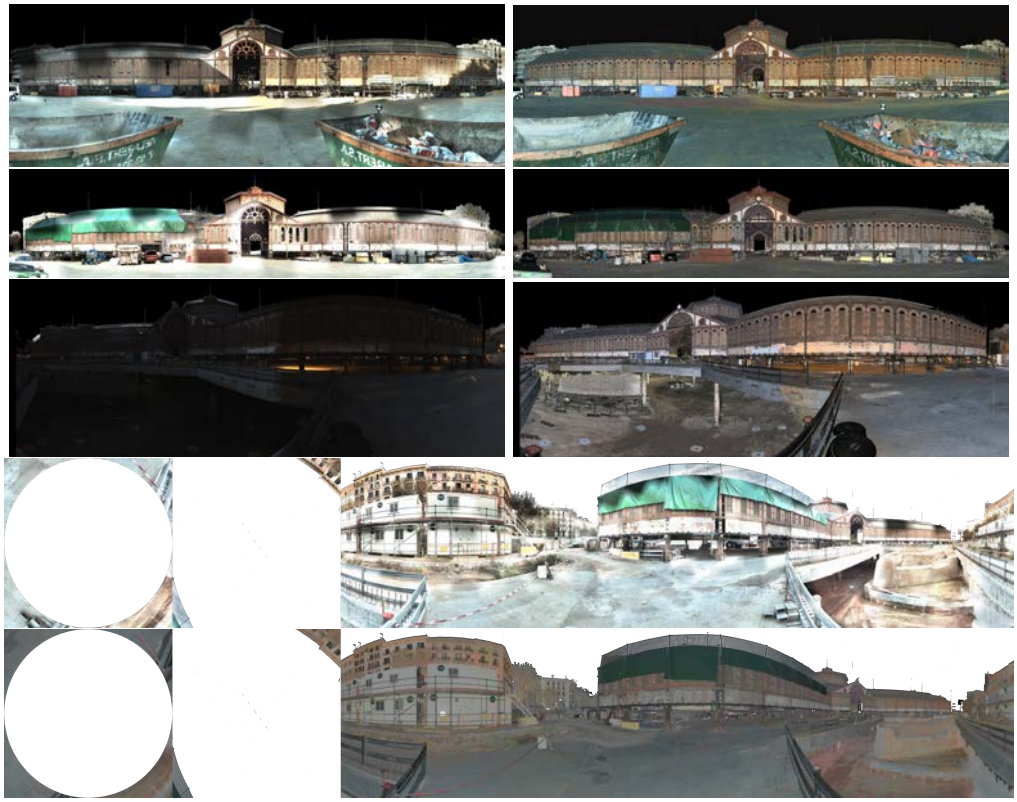


Figure 4.38: Showing the effects of our color enhancement strategy for different clouds of the same scene captured with a Leica P20. For the first three rows we see original color **(left)** and enhanced color **(right)**. The last two rows depict the same scan with original color **(top)** and enhanced color **(bottom)**.
Examples show how color becomes more consistent across a single scan and sharp color discontinuities are alleviated. Moreover, color also becomes more consistent across all the scans, even for those captured in very low-light conditions.

4.3.5 Results and Discussion

We implemented the two proposed applications using C++ and GLSL code. The GUI approach uses GPU acceleration and runs much faster than the CPU-only implementation. However, the former requires implementing a smart blending strategy for splats that map to the same pixel, whereas the averaging is trivial for the latter.

Following, we will present qualitative and quantitative results for the different desirable properties outlined at the beginning of the section. However, this part will mostly focus on the encoding of the properties. The practical use of these properties will be demonstrated later in this document.

Detail compression

One of the main advantages of encoding point cloud properties into textures is the availability of many techniques for this representation. We use already available image encoding to achieve significant compression ratios for this information. For geometric properties, we propose using lossless formats such as *PNG* or *TIFF* whereas for others, such as color, lossy formats such as *JPG* can be used. A summary of compression ratios obtained using this representation for colors is shown in Table 4.4.

Dataset	# points	Binary Encoding	PNG Encoding	PNG Savings	JPG Encoding	JPG Savings
Pedret Interior	1213 Mpts	3.4 GB	813.9 MB	76.6%	198.3 MB	94.3%
Pedret Exterior	1191 Mpts	3.3 GB	1137.2 MB	66.6%	148.9 MB	95.6%
Doma Interior	661 Mpts	1.9 GB	565.8 MB	70.1%	91.7 MB	95.2%
Doma Exterior	1026 Mpts	2.9 GB	270.1 MB	90.1%	42.6 MB	98.6%
Market	372 Mpts	1.0 GB	98.5 MB	90.8%	20.2 MB	98.1%

Table 4.4: Evaluation the storage savings achieved by encoding color information from point clouds into textures.

Color enhancement

We extensively evaluate the effect of our color enhancement strategy, namely replacing the raw intensity of the color with the IR intensity reported by the sensor. In Figures 4.2 and 4.421, we show how this strategy alleviates the sharp discontinuities and shadow caused by capturing the color for points using a mosaic of images recorded with different camera settings (Figure 4.33). As already discussed, the reported IR intensity is less sensitive to global illumination conditions, allowing us to remove shading and obtaining something close to the albedo of the material (Figure 4.39).

Figure 4.39: Showing the effects of our color enhancement strategy on a cloud captured with a Leica RTC 360. Because reflected IR intensity is less sensitive to illumination conditions our strategy lighting effects (shadows, highlights) making the depicted stone wall appear flat.

Reflected IR intensity depends on the distance to the scanned surface, the ray's incident angle, and the surface material. A process called *radiometric calibration* takes care of the first two variables. Hence, afterward, it should only depend on the material. This implies that an object captured from two different scan locations should have the same IR value. Thus, our color enhancement strategy makes the color more consistent within a single scan and between different scan locations. This is the case for the Leica P20 (Figure **4.38**) which suffers several color artifacts but also for the Leica RTC 360 (Figure **4.40**).

Another property of our method is that it can recover some information for clipped highlights. For some very bright regions of our scene, a long camera exposure time may cause different pixel sensors to become completely saturated. In these regions, color information is almost lost, resulting in very bright white tones. Our strategy recovers the shape of objects in clipped highlights (Figure **4.422**) because it replaces the saturated values with the IR values.

One shortcoming when processing highlights is that the values are not recoverable, and the resulting color is always a gray tone. Something similar happens in shadowed regions. In soft-shadowed regions, we may recover the color information partially, but, for darker ones, we can only recover shape (Figure **4.423**). Nevertheless, the biggest issue in shadowed regions is that, by brightening them, we also increase the noise level. One of the most significant sources of noise in imagery is called *Photon-shot noise*. A Poisson process can model this type of noise. Namely, its standard deviation is the square root of its mean. Hence, in low-light regions, the noise's variance is significant compared to the pixel's value. One interesting topic for future work would be using the IR image to remove the color image's noise since the former is very smooth.

Figure 4.40: Showing the effects of our color enhancement strategy for different clouds of the same scene captured with a Leica RTC 360. Rows **(1)** and **(3)** show original color and rows **(2)** and **(4)** show the corresponding enhanced color.
This scanner usually produces fewer color artifacts since it provides HDR images. Namely, it combines different image captures with different exposure settings for the same fragment of the scene. This scene was captured on a cloudy day. This is a usually good setting for outdoor scenes since diffuse illumination avoids the appearance of hard shadows and highlights. Nevertheless, we can see how one scan **(1)** appears clearly brighter than the other **(3)**. Our strategy can make their color more consistent.

Figure 4.41: Showing the color artifacts produced by the Leica RTC 360. Zooms are shown and discussed in Figure 4.42.

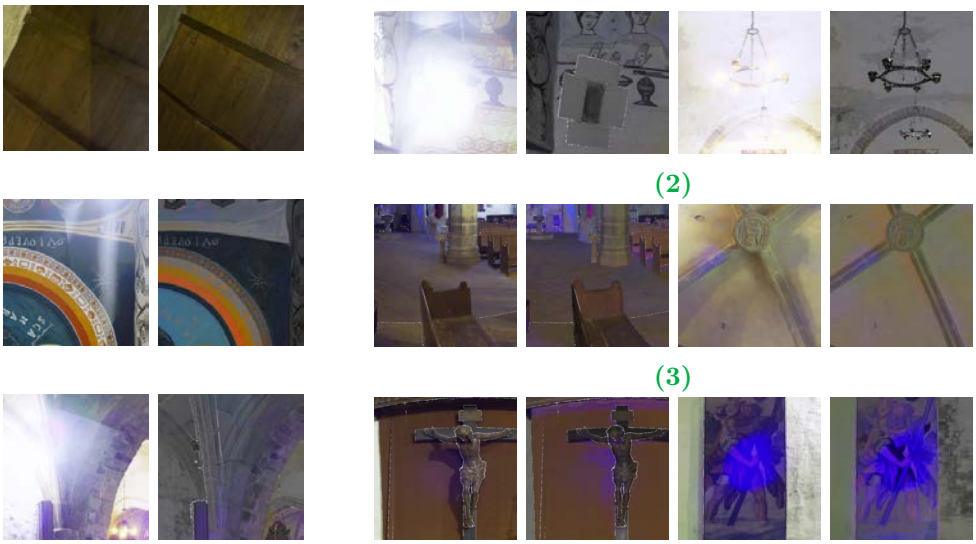


Figure 4.42: Showing the effects of our color enhancement strategy for different artifacts produced by the Leica RTC 360. This scanner usually produces fewer color artifacts since it provides HDR images. Namely, it combines different images captured with different exposure settings for the same fragment of the scene. However, in **(1)**, we can still see some color discontinuities (left), which are alleviated by our strategy (right).
In **(2)**, we can see the effect of our strategy on highlights. Even if we cannot recover color information (because this is completely lost on the original image), we can still recover the objects and paintings' shapes.
In **(3)**, we can see the effect of our strategy on shadows. In soft-shadowed regions, we can recover the color information (since this was not completely lost on the original image). However, by brightening these regions, we sometimes end up increasing the noise level.
In **(4)**, we can see how our strategy fails at fixing lens flares. Since the blue channel's contribution to our luma computation is small, these areas' intensity ends up being similar to other flare-less regions' intensity.

While we increase the noise level when brightening shadows and cannot recover color in highlights, we still achieve some color enhancement. However, in the case of lens flares, it is usually the opposite. In our case, the blue channel's contribution to the luma computation is meager (0.0722). Our lens flares appear as bluish stains (Figure **4.424**). However, the luma value computed on them remains very similar to other stainless regions because the blue channel contribution is low. Hence, replacing this luma value with the corresponding IR value does not remove the flare.

Texture editing We have seen how to benefit from existing compression algorithms for images. Similarly, by representing cloud properties using textures, we can benefit from multiple available image edition software (Figure **4.43**). This is especially useful because users are usually more familiar with 2D editing than with 3D editing. Indeed, modifying point cloud properties (such as color) on top of the 3D cloud can be tedious.

Figure 4.43: Illustrating an easy edition task on the color of a point cloud represented as a texture. From left to right: **(1)** The original cloud displays a graffiti on the wall. **(2)** Within minutes and using open-source image edition software, the graffiti has been removed and a feasible texture has been inpainted. **(3)** Custom logo addition on top of the texture.

Pre-computed properties

Color and IR intensities are two properties usually provided by the scanner and that we will usually want to encode using our panoramic textures. Another candidate for this encoding is normal vectors, which we have seen how to compute robustly in Section **4.2**. However, we are not limited to these. We can precompute and store a considerable diversity of features and properties on the original point clouds before simplification. Some of these are curvature, salience, or ambient luminance coefficients.

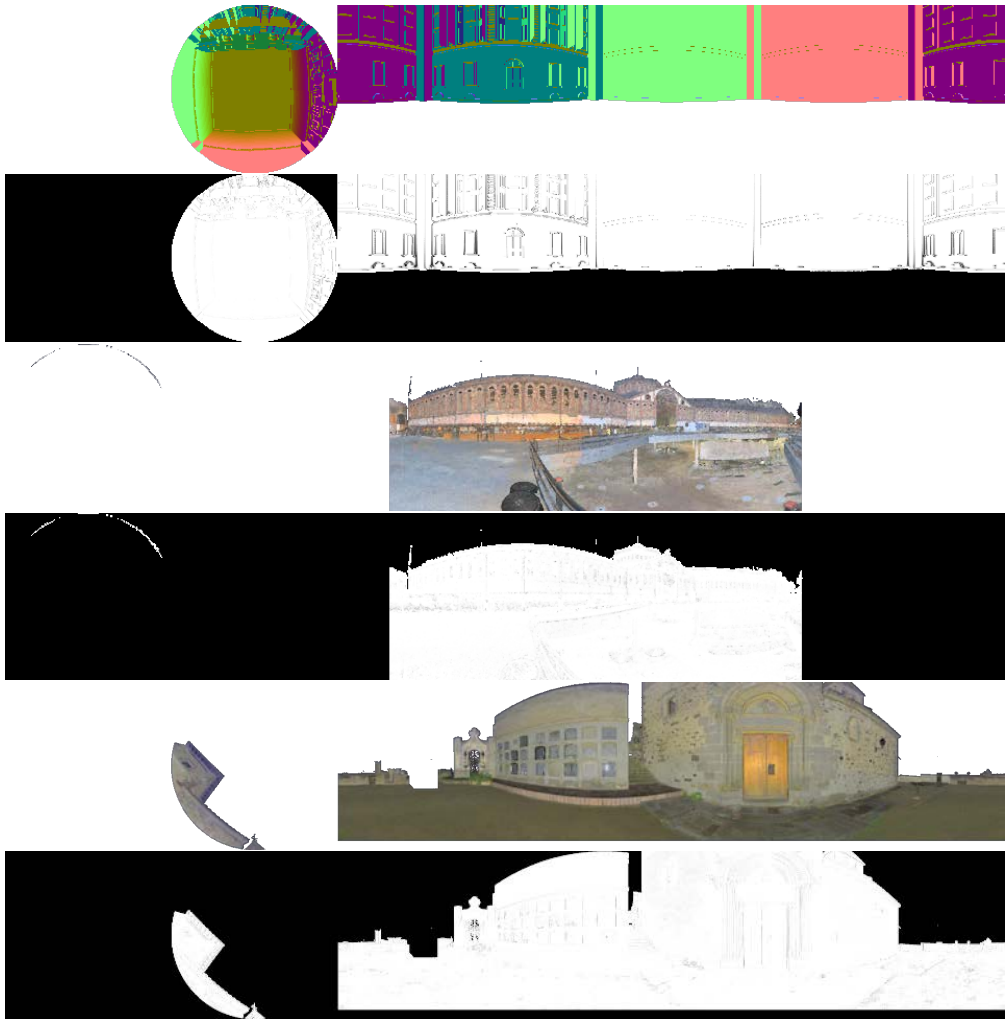


Figure 4.44: Illustrated ambient luminance coefficients computed on the original point clouds and stored using our panoramic representation. From top to bottom: **(1)** Normals for the *mansion* model. **(2)** Ambient luminance coefficients for the mansion model. **(3)** A cloud from the *market* model. **(4)** Ambient luminance coefficients for the cloud from the *Market* model. **(5)** A cloud from the *Doma exterior* model. **(6)** Ambient luminance coefficients for the cloud from the *Doma exterior* model.

4.4 PUBLICATIONS

Our contributions to LiDAR point cloud processing have led to three publications. The first one was published on the special issue on "Special Issue on Large-Scale 3D Modeling of Urban Indoor or Outdoor Scenes from Images and Range Scans" of the "Computer Vision and Image Understanding" Journal. It describes the Weighted Least Squares normal estimation method and part of our panorama generation and color correction schemes:

- M. Comino, C. Andujar, A. Chica, and P. Brunet. "Error-aware construction and rendering of multi-scan panoramas from massive point clouds". In: *Computer Vision and Image Understanding* 157 (2017). Large-Scale 3D Modeling of Urban Indoor or Outdoor Scenes from Images and Range Scans, pp. 43–54

The second one was presented at the "Symposium on Geometry Processing 2018" held in Paris. It also was awarded the Best Paper Award - Third Price, and later published in the journal "Computer Graphics Forum". It describes the noise-aware covariance correction approach, which is the basis for strategies **(S1)** to **(S2)**.

- M. Comino, C. Andujar, A. Chica, and P. Brunet. "Sensor-aware Normal Estimation for Point Clouds from 3D Range Scans". In: *Computer Graphics Forum* 37.5 (2018), pp. 233–243

Finally, the last one was presented at the local conference "CEIG - Spanish Computer Graphics Conference 2019". It contains further details on encoding cloud properties using the Low-stretch Panoramic Representation.

- M. Comino Trinidad, A. Chica Calaf, and C. Andújar Gran. "View-dependent Hierarchical Rendering of Massive Point Clouds through Textured Splats". In: *Spanish Computer Graphics Conference (CEIG)*. The Eurographics Association, 2019

We have also submitted our work on point cloud simplification, and it is currently under review.

5

Algorithms for the Improvement of Photogrammetric Point Cloud Data

We have seen that LiDAR devices perform very accurate measurements and yield very reliable point clouds. These characteristic makes them very suitable for urban digitization since it allows recovering fine geometric detail. However, they are very delicate equipment that cannot be handheld. Even if they have become much more affordable in recent years, their price is not accessible to the general public. Hence, they are not suitable for a casual scanning session or personal use.

As opposed to this, photogrammetric techniques only require a set of input images. Even if the final reconstruction's quality partly depends on the input image quality, decent models can be obtained with pictures taken from an affordable medium-quality camera. Consequently, photogrammetric software is broadly used, e.g., in cultural heritage due to its low cost and fast capture times.

The major challenge for photogrammetric techniques is related to the quality of the produced models. There is a vast diversity of factors that can impact this. Some factors are related to the capturing process (e.g., lighting conditions or camera settings), some are related to the captured content (e.g., texture-less or specular regions), and some are related to the reconstruction algorithms (e.g., registration mismatches).

In this Chapter, we study how to overcome these difficulties to improve the quality of the reconstructions. Our key idea is to incorporate semantic information to treat the scene objects differently during the reconstruction process. The rest of the Chapter is organized as follows:

1 **Overview of the Photogrammetry Pipeline:** In Section 5.1 we enumerate the different steps involved in a typical photogrammetry pipeline. Then, we identify and analyze the different challenges that appear at each step and discuss which ones can be improved using semantic information.

2 **Effective Visualization of Sparse Image-to-Image Correspondences:**

3 **Semantic-Aware Reconstruction:** Different artifact sources impact the quality of the produced reconstructions. Thanks to the extracted semantic information, in Section 5.3, we study how to apply domain knowledge to alleviate these noisy sources' effects on different parts of the photogrammetry pipeline.

5.1 OVERVIEW OF THE PHOTOGRAMMETRY PIPELINE

Photogrammetric reconstruction combines Structure from Motion (SfM) and Multi-View Stereo (MVS) techniques. SfM [SF16] estimates the location of a set of images in a global reference frame. First, this process computes a set of features for each image. The algorithm then matches these features between views, and an iterative refinement step estimates the intrinsic and extrinsic information of each image.

Instead, MVS [SZP+16] densely reconstructs the scene. A depth map is estimated for each input image by finding, for each pixel, the depth with minimum re-projection error into a set of neighboring views. The algorithm generates a dense reconstruction of the scene by finding consistent points across different depth maps.

Finally, we could use any available surface reconstruction algorithms to compute a mesh out of the estimated points. A popular choice is Poisson Surface reconstruction [KH13]. We can also obtain a high-resolution colorization of the mesh by projecting the original images onto it.

Schematically, this process can be split into the following steps:

1. **Image acquisition:** Involves taking several pictures of the object to reconstruct with significant overlap between them. Diverse viewpoints help to reconstruct detailed features.

2. **Feature detection and extraction:** Finding locally relevant points in the input images using local descriptors such as Sift [Low04].

3. **Feature matching between image pairs:** Finding correspondences between regions of different images.

4. **Sparse reconstruction:** Also known as *Bundle adjustment* is the process of jointly optimizing the 3D scene geometry and the camera poses and intrinsics.

5. **Image undistortion:** Correcting the lens distortion.

6. **Dense stereo reconstruction:** For each image, the depth at each pixel is estimated.

7. **Depth map fusion:** The estimated depths across multiple depth maps are checked for consistency to produce a point cloud.

8. **Meshing:** Converting the point cloud into a triangle mesh using a surface reconstruction algorithm.

9. **color projection:** color images are projected into meshes, and their color is averaged to obtain a high-resolution texture.

Major challenges in photogrammetric reconstructions have been studied extensively for general objects [SCD+06; BCM18]. Here we analyze the main issues, focusing on those especially relevant reconstructing urban models and facades. Furthermore, we discuss how they are related to the different steps of the photogrammetry pipeline.

We classify the primary sources of artifacts into three groups: those related to the capturing process, those inherent to the scanned content, and those caused by an algorithm shortcoming.

5.1.1 Challenges: Lighting conditions and camera settings

The surrounding lighting conditions greatly influence images, and cameras are designed with multiple features that try to adapt to these. Camera sensors record the number of incoming photons to determine an intensity value for each image pixel. A Poisson process can model the arrival process for these photons, and this explains one of the most important sources of noise in images, known as Photon-shot noise.

The standard deviation of the noise at each pixel is the squared root of the average number of photons (n) that arrive at it. Hence, the signal-to-noise ratio is $SNR = \sqrt{n}$. Consequently, when a few photons arrive at the sensor, the noise level is huge compared to the recorded value. To minimize the effect of noise, we would ideally want to capture as many photons as possible, which poses the problem of choosing the right exposure when taking a picture.

The amount of light that reaches the camera depends on three different settings: the aperture of the diaphragm, the shutter's speed, and the size of each pixel sensor. For a given camera, the latter is fixed. A larger diaphragm aperture allows capturing more photons and reduces the depth of field (the depth interval on which the image is in focus). Therefore, it is not advisable to use large values landscape photography. Consequently, the only setting that can be considered, for our case, is the shutter speed.

Choosing the right shutter speed is not a trivial task since slow speeds can cause the saturation of some pixels (clipped highlights Figure 5.12), and fast speeds can cause noisy shadowed regions. High-dynamic range scenes (Figure 5.11) are those with shadowed areas and areas where much light reaches, and choosing a single shutter speed that works for both is not possible. Moreover, slow speeds can also cause motion blur if the camera is handheld or in the presence of moving objects (Figure 5.13).

Other artifacts introduced by the cameras are lens flares (Figure 5.14), namely flares caused by the scattering and refraction of the light within the lens system.

Finally, light sources can also become a source of artifacts. For photogrammetry, the ideal setting would be having objects influenced only by perfect diffuse light. For instance, cloudy days are preferred over sunny ones for outdoor shootings. Moreover, strong directional light sources, such as the Sun, can cause the appearance of hard shadows (Figure 5.15), which influences the final color and may cause registration mismatches if these move across the different images.

(1) (2) (3) (4) (5)

Figure 5.1: Illustrating different artifact sources, which are related to the capturing process. From left to right: **(1)** High-dynamic range scene with both shadowed and bright areas. **(2)** Completely saturated regions appear on the side of a facade. **(3)** Motion blur on an image taken with a handheld camera. **(4)** Lens flares. **(5)** Strong shadows cast by the Sun.

5.1.2 Challenges: Ill-behaved Content

Most photogrammetry algorithms have been designed under the assumption that content is coherent between images. However, for urban scenarios, we can find regions where this assumption is violated. In this context, the main artifact sources are:

- Low-texture surfaces, which cause gaps and missing parts on the final reconstruction.
- Mirror-like surfaces, which also cause gaps and noise.
- Occluding objects, which result in missing facade parts and color pollution when re-projecting the images back onto the reconstructed model.
- High-frequency depth discontinuities (e.g., between balcony rails and the wall), which cannot be captured at standard resolutions.

Some artifacts can be alleviated during the reconstruction time by tuning SfM and MVS parameters. However, this is a tedious task that requires substantial knowledge of the field.

Texture-less parts

Photogrammetric-based pipelines strongly rely on image features to reconstruct a scene (unlike LiDAR-based scanners). These features are crucial for correctly registering images during sparse reconstruction and stereo patch matching

during dense reconstruction. Homogeneous or texture-less image regions lack these features. Hence, the algorithms may fail to reconstruct these regions leaving gaps. Some of these holes will remain even after the meshing step, causing a severe visual impact and requiring manual editing (see Figure 5.2). Building facades are especially prone to include planar texture-less parts. Meaning these objects are challenging for photogrammetric reconstruction.

Figure 5.2: Example of poor reconstruction of low-texture parts. The reconstructed dense point cloud (Colmap) has important missing parts which are not recovered through meshing.

Non-Lambertian surfaces

Pure Lambertian surfaces exhibit a fully matte appearance. Namely, they reflect the light in an entirely diffuse way. Most surfaces do not have this ideal behavior and are non-Lambertian to some degree. When light interacts with a strong non-Lambertian surface, we can observe different effects such as specular reflection, refraction, or subsurface scattering. Meaning, the final surface appearance will depend on the incident angle. These surfaces pose a severe problem for photogrammetric reconstruction as the same surface will not look coherently across different views. Consequently, feature and patch matching steps will fail, resulting in noisy or missing points in both sparse and dense reconstructions (see reconstructions in Figure 5.3).

In building facades, we can find one example of these surfaces in window panes. They are frequent objects which are usually poorly reconstructed. At near-grazing incidence, glass reflectivity (ratio of reflected power to incident power) is close to 1.0 (thus mirror-like), whereas at normal incidence is about 0.04 (poor reflector). Moreover, in daylight conditions, the environment reflection usually outshines the transmission from the building's interior. The use of solar reflectivity films can also modify this behavior (see Figure 5.3-left).

Figure 5.3: Glass window behavior goes from nearly refractive to nearly mirror-like depending on incident angle and glass treatment (e.g. reflective films). In this reconstruction example (Colmap), window panes appear noisy and sparse, and the resulting mesh has significant noise.

Occluding objects

Undesired occluding objects can induce two major types of artifacts during photogrammetric reconstruction. On the one hand, they hide parts-of-interest of the scenes, difficulting their reconstruction. For instance, by default, Colmap requires each element to appear in at least 5 views to pass consistency checks. If a given point is occluded in multiple views, it might not get reconstructed, resulting in missing parts.

On the other hand, occluding objects may cause severe color artifacts. At the end of the photogrammetry pipeline, the input images can be re-projected onto the final mesh to obtain high-detail color information. If the occluding object has not been correctly reconstructed (e.g., a passing bus that only appears in one view), these will be projected onto the background.

In an urban scene, occluding objects can appear both at street-level (e.g., pedestrians, vehicles, and urban furniture occluding the storefront) and at facade-level (e.g., trees and light poles also occluding the upper facade). Especially tree leaves and branches are usually poorly reconstructed, and thus not even depth comparisons are not enough to prevent color pollution (Figure 5.4).

High-frequency content

Repetitive thin structures, such as balusters and railing parts in balconies and terraces, are often poorly reconstructed. One reason for this is that street-level imagery often fails to capture enough detail on these very narrow elements.

Figure 5.4: Despite depth tests, colors from occluding objects such as tree leaves often contribute to facade textures. The contribution is specially noticeable in those parts captured from a few unoccluded views. From left to right: **(1)** Photo a building. **(2)** Reconstruction (Colmap + image reprojection); notice tree color on the facade. **(3)** Removing reconstructed trees before reprojection increases color pollution because of the lack of depth test rejections.

Nonetheless, even if high-resolution close-up views were available, it is hard to correctly match these repetitive patterns between views, causing uncertainty during the depth estimation process.

In urban facades, balconies are especially challenging to reconstruct due to their railing parts. To completely recover these thin elements, we would require very dense reconstructions, which are not practical. Therefore, most algorithms reconstruct them poorly either by filling the holes between them or leaving larger holes that reveal the background (see Figure **5.5**).

Figure 5.5: High-frequency facade elements such as balcony rails often cause sharp depth discontinuities between the rails and the wall. From left to right: **(1)** Photogrammetric reconstruction (Colmap) with reconstructed colors. **(2)** Same reconstruction using uniform color. An ambient occlusion term has been added to emphasize depth discontinuities.

5.1.3 Challenges: Algorithm Flaws

While some content is inherently challenging for photogrammetry algorithms, there are cases where their failures are not related to these. A typical example is associating two patches from two images, which are different, during feature matching or failing to correctly estimate the depth value for a pixel during dense reconstruction.

Figure 5.6: Feature matches between two different images. We show in red missmatches probably caused by the repetitive pattern and nor taking into account enough context.

5.2 EFFECTIVE VISUALIZATION OF SPARSE IMAGE-TO-IMAGE MATCHES

Feature detection and feature matching are fundamental operations in different Computer Vision applications. These include, but are not limited to, photogrammetry, structure-from-motion [SF16], multi-view stereo [SZP+16], image-based localization [LSX+15; NLH+19], content-based image retrieval [ZYT18] and motion field prediction [LYT11].

Different authors have proposed several keypoint detectors and feature descriptors such as SIFT, SURF, and ORB. More recently, learned CNN-based descriptors are gaining popularity (see [KPS17; ZYT18] for recent reviews). Most of these descriptors have been designed to be robust against image differences caused by affine transformations, intensity variations, or viewpoint changes. Nevertheless, they may diverge when faced with large differences causing wrong matches. Occluding and moving objects, specular surfaces, and self-similar structures are challenging elements that will difficult photogrammetric reconstruction. Other tasks, such as aligning images from different 3D scenes, may introduce even more challenges. For instance, tackling images containing different object instances, from different viewpoints, and at different locations [LYT11].

Feature mismatches can negatively influence the registration between two images, and this error can be carried over to the whole system. Ultimately, this can cause the whole reconstruction process to fail. In these cases, it is crucial to understand the causes behind these mismatches and correct them, if possible. The easiest way of detecting a feature mismatch is by visualizing it, and state-of-the-art photogrammetry pipelines already include tools for this task. Traditionally, keypoint matches are shown by placing the two images side-to-side and drawing link segments connecting pixels with matching features (Figure 5.7). Alternatively, segments can be drawn joining matched keypoint locations within each image [LYT11] instead of between images (Figure 5.7). This may be useful for illustrating, e.g., motion prediction [LYT11] or spotting potential outliers, but not for finding out an explanation for wrong matches.

Unfortunately, these approaches tend to produce cluttered images where individual matches are hard to identify (Figure 5.7-left), and outliers are entirely hidden. Since drawing all matches leads to cluttered images, a few approaches use multiple colors instead of a single one [KPS17]. However, these colors are assigned randomly, and thus some neighboring segments share similar or identical colors.

Figure 5.7: Traditional feature match visualization (VisualSFM), using a vertical layout. From left to right: **(1)** Matches represented as line segments joining matching keypoints. **(2)** Segments represent the displacement of the matches, within each individual image.

Taking into account these characteristics, we formulate the following desired properties for an image-to-image sparse correspondence visualization method:

- **Identifiable Context:** Users must be able to interpret the image contents surrounding each keypoint. For instance, users should be capable of identifying whether a match between two building windows corresponds to the same window instance (same floor, same column). Consequently, visual overlays should be as little invasive as possible.

  We propose using *hierarchical clustering* to group compatible matches and display fewer representatives. The general view will only show the top aggregated segments providing a good overview of the matches. The user can selectively explore matches at finer levels by clicking on a segment or selecting an image region. We use the line stroke thickness to encode the number of clustered matches, with more massive clusters having thicker strokes than smaller ones.

- **Identifiable Matches:** Users must be able to identify individual matches quickly, i.e., given an arbitrary keypoint on one image, finding the matching keypoint on the other image.

  We first propose computing the optimal arrangement of the images in each pair instead of using simple vertical or horizontal layouts. We compute the

image placement (relative 2D translation) that minimizes the overlaying of segments.

However, this is not enough. On images captured from close viewpoints, most link segments are approximately parallel. Consequently, there is a high overlap between them, which hinders visually tracing the segment between one keypoint and its matched counterpart.

Thus, we propose improving the visual traceability of segments by bending them using quadric curves, coloring them using high contrast palettes, and assigning unique colored glyphs to each matched keypoint pair.

Highly-contrasted color palettes have been employed in the context of transport maps. Similarly, Green et al. [Gre10] study using them to quickly identify the different routes on a map without risk of confusion. We use the palettes proposed by Kelly et al. [Kel65] by assigning different colors to neighbor edges.

- **Identifiable Outliers:** We have empirically found that isolated matches often correspond to outliers. Hence, we decided to give visibility priority to small clusters.

5.2.1 Visualization approach

Our algorithm takes as input an arbitrary pair of RGB images A and B. along their corresponding feature matches $\{(x_i, y_i) \in A\}$, $\{(x'_i, y'_i) \in B\}$. In our experiments, we used SIFT features and matched them using COLMAP, although our algorithm should work with features estimated using any other method.

In a typical case, the images in a pair will usually belong to similar scenes and have similar viewpoints. Our application interactively shows both images and a set of segments representing aggregated or individual matches.

Edge clustering

We perform the clustering of the matches aiming for the same goal as edge bundling, i.e., trying to reduce clutter. However, we use an edge compatibility criterion that does not depend on the two images' relative placement. Consequently, we only need to perform the clustering once before the layout optimization step.

$$||(x_i, y_i, x'_i, y'_i) - (x_j, y_j, x'_j, y'_j)|| \tag{5.1}$$

We represent a match by its endpoints $(x_i, y_i) \in A$ and $(x'_i, y'_i) \in B$ and use Euclidean distance to measure the distance between them. In particular, we use the distance between the 4D points $\{(x_i, y_i, x'_i, y'_i)\}$. This metric favors the clustering of segments with endpoints close to each other.

We cluster these matches using a bottom-up hierarchical strategy. The main advantage of the hierarchy is that, during inspection time, the user can refine a coarse overview into a detailed view by selecting clusters or regions of interest.

As initialization, we assign each segment to a different cluster. Clusters are iteratively merged until we reach the desired number of clusters. We propose three different metrics to measure the compatibility of two clusters S_1, S_2. More precisely, we use the *min*, *avg* or *max* distance between any match in S_1 and any match in S_2.

Figure 5.8 shows the output for these strategies. The choice of the metric to use depends on the aim of the visualization. For instance, if we want to spot outliers easily, we would select a strategy that avoids getting them clustered with other matches. Assuming A and B have similar viewpoints, potential outliers are likely to correspond to 4D points with no nearby matches, whereas inliers will tend to form large dense regions. The *min* metric (also known as single linkage) is prone to generate large clusters representing inlier matches while keeping isolated matches (potential outliers) separated.

In contrast, the *max* metric (complete linkage) keeps clusters from growing too much in extent. Consequently, outlier matches are likely to be merged into other clusters, getting hidden in the final map. Finally, the *avg* metric provides a trade-off between inlier and outlier attention, which is our default option.

Independently of the chosen strategy, we represent all clusters using segments. The segment's endpoints are computed using the centroid of the clustered keypoints, and we represent them as circles. Moreover, the thickness of the segment is used to encode the size of the cluster.

Image layout optimization

Our problem lies in-between graph layout methods (where nodes are free to move) and origin-destination flow maps (where nodes are fixed). We consider two images A and B, with sizes $w_a \times h_a$ and $w_b \times h_b$, respectively, which have N matches with endpoints $(x_i, y_i) \in A$ and $(x'_i, y'_i) \in B$. We consider these matches to be fixed and optimize for the relative position of the images. We do not explore rotations, scales, and shears because they may hide the true transform relationship between the images.

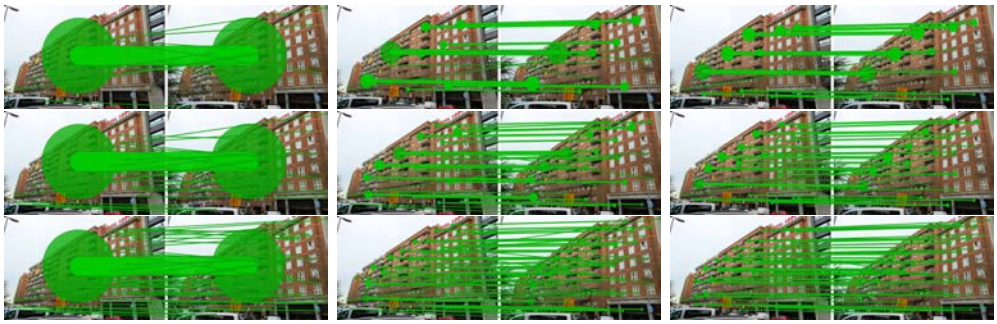


Figure 5.8: From left to right: **(1)** clusters computed using the *min* metric, **(2)** the *avg* metric and **(3)** the *max* metric. From top to bottom: **(1)** 12 clusters, **(2)** 25 clusters and **(3)** 50 clusters.
Compare to Figure **5.7**-left.

Without loss of generality, we consider the image A as fixed and translate image B by a vector $\mathbf{t} = (t_x, t_y)$. We use the sum of squared distances between matches as the optimization criterion. However, we restrict the relative positions between the images by adding constraints, as we do not want to overlap. The error function without constraints becomes:

$$\begin{aligned} E(\mathbf{t}) &= \sum_i \|a_i - b_i - \mathbf{t}\|^2 = \sum_i (a_i - b_i - \mathbf{t})^T (a_i - b_i - \mathbf{t}) \\ &= \sum_i (a_i - b_i)^T (a_i - b_i) + N\mathbf{t}^T\mathbf{t} - 2\sum_i (a_i - b_i)^T \mathbf{t} \end{aligned} \tag{5.2}$$

We have to find a vector $\mathbf{t}$ that minimizes $E(\mathbf{t})$, but the first term does not depend on $\mathbf{t}$, so:

$$\min_{\mathbf{t}} E(\mathbf{t}) = \min_{\mathbf{t}} \left[N\mathbf{t}^T\mathbf{t} - 2\sum_i (a_i - b_i)^T \mathbf{t} \right] \tag{5.3}$$

Now, to prevent any overlapping, we need to add a set of constraints. We will allow for two types of layouts: side-by-side or one image on top of the other. We will now address the side-by-side case as the other one is analogous. In this case, we will want to have $\mathbf{t} = (s, t_y)$, with s a fixed value. When $s = w_a$ image B will be to the right of image A, while when $s = -w_b$ image B will be to the left of A. Substituting in the previous equations and eliminating from the minimization those terms that do not depend on t_y:

$$\min_{\mathbf{t}} E(\mathbf{t}) = \min_{\mathbf{t}} \left[N(t_x t_x + t_y t_y) - 2 \sum_i t_x (x_i - x_i') + t_y (y_i - y_i') \right] \tag{5.4}$$

$$\min_{\mathbf{t}} E(\mathbf{t}) = \min_{\mathbf{t}} \left[N t_y t_y - 2 \sum_i t_y (y_i - y_i') \right] \tag{5.5}$$

Now we can derive and equal to zero:

$$\frac{\partial E(\mathbf{t})}{\partial \mathbf{t}} = 2N t_y - 2 \sum_i (y_i - y_i') = 0 \tag{5.6}$$

$$t_y = \frac{1}{N} \sum_i (y_i - y_i') \tag{5.7}$$

Hence, we have found that the optimal t_y is the mean of the differences between the matches' y components. This result is valid independently of the chosen s value. However, it is necessary to calculate $E(t)$ for $s = w_a$ and for $s = -w_b$ independently, and choose the option that results in the minimum error.

The case considering a vertical alignment is analogous, and the optimal t_x is the average of the matches' x components. The final alignment is chosen by computing the error measure for the two horizontal and the two vertical candidate layouts and choosing the arrangement with the minimum error value.

Figure 5.9 shows the optimal layout for different image pairs.

Figure 5.9: Optimal layout found for three different image pairs.

Color coding

To facilitate the visual tracing of segments, we propose using distinct colors for neighboring matches. This technique is especially helpful for distinguishing segments at crossings and cluttered parallel sections. To the best of our knowledge, existing approaches in the context of feature matching are limited to assigning random colors to edges [KPS17]. Instead, we use the high-contrast Kelly's [Kel65] color palette and assign different entries to neighboring segments.

The coloring of the segments should minimize collisions between segments that are close to each other. Consider images A and B are laid out horizontally. We compute the intersection of all segments with the vertical midway axis between A and B and use their order of intersection to assign colors. Let r_i be the rank order of the i-th segment. The color index c_i for such a segment is computed simply as $c_i = r_i \bmod N_c$, where N_c is the number of palette colors to be used. We perform analogously for the case where A and B are laid out vertically.

Figure 5.10 compares different color mappings. We verified that our simple approach assigns different colors to roughly parallel segments. However, it may assign similar colors to segments that are not parallel and intersect at larger angles. For the former case, parallel segments are hard to trace visually, and our coloring approach greatly eases this task. The latter case does not pose a severe visual challenge because our visualizations are usually uncluttered.

Figure 5.10: Different color mappings. From left to right: **(1)** Single color. **(2)** Random color. **(3)** Our approach with 22 Kelly's color. **(4)** Our approach with 9 Kelly's colors. These first 9 colors are maximally different for people with defective color vision as well as for people with normal vision [Gre10].

Segment bending

We offer the possibility of bending segments to improve our visualization. This tool moves nearly-parallel segments further apart and can be convenient

when images A and B have similar viewpoints, causing many segments to be roughly parallel (Figure 5.11). This process replaces straight line segments with a quadratic Bézier curve by adding a single control point. These points initially lay on the intersection between their corresponding segment and the midway axis between A and B. We use the (median-centered) rank order r_i to move the control points towards the image borders. The higher the rank, the higher the deviation, and thus the bending of the segment. Notice this behavior is similar to that of repulsion-force approaches, which bends nearly-parallel edges towards the map's sides.

Figure 5.11: Segments drawn with increasing bending factors.

Glyph assignment

Other authors have previously explored using colored glyphs for labeling matched images features (e.g., [LYT11]). We implement this technique by drawing lowercase and uppercase letters from the English alphabet as glyphs. In our experiments, this has proved to be enough since we usually visualize 25-50 segments. Next to the circles representing the features, we draw the letters using the same color as the corresponding segment (Figure 5.12). We also use the rank order r_i to determine the letter associated with each segment. Hence, adjacent nodes sharing the same letter are highly unlikely.

Figure 5.12: Final output. From left to right: **(1)** $N = 25$, **(2)** $N = 50$ and **(3)** $N = 100$ clusters.

5.2.2 Results and Discussion

We implemented our algorithm using Python and tested it on images of 10 to 25 Mpixels and 10^3 to 10^4 matches. All our executions were performed on a commodity PC and took less than one second for all steps, except image I/O.

Segments and glyphs might overlap, therefore, the order in which we draw them matters. We defined a prioritization strategy that tries to maximize the displayed information. We sort segments by their thickness (cluster size) and draw thinner segments on top of thicker ones. Nodes do not usually overlap. Consequently, glyphs are unlikely to hide each other, and drawing them the last avoids getting them occluded (Figure **5.12**).

Figure **5.13** compares our results against a baseline [SF16] approach enhanced to include our layout optimization, with green **(2,4)** or random color **(5)** lines. In our output images, aggregated segments are easy to follow, outlier matches are apparent, and the image content is mostly preserved. Thus, users can easily check matches. In contrast, the baseline approach hides a large part of the image content. Although some main directions are apparent, individual matches can hardly be followed, and only a few outlier matches can be distinguished.

In conclusion, our method can effectively reduce image clutter and make the displayed information easy to read. Thus, making this visualization useful for the viewer.

Figure 5.13: Results compared with a baseline approach. From left to right: **(1,3)** Results using our approach. **(2,4)** Baseline. **(5)** Baseline using random colors. With our approach, wrong matches 'x', 'y' are easy to spot and verify in our output, whereas in the baseline only 'x' is easy to follow. Random colors facilitate the detection of some outlier directions, but the image content is too occluded to allow for any checking.

Figures **5.14** and **5.15** compare different clustering strategies with a baseline approach. The best strategy depends on the user's interest in effectively identifying outlier or inlier matches.

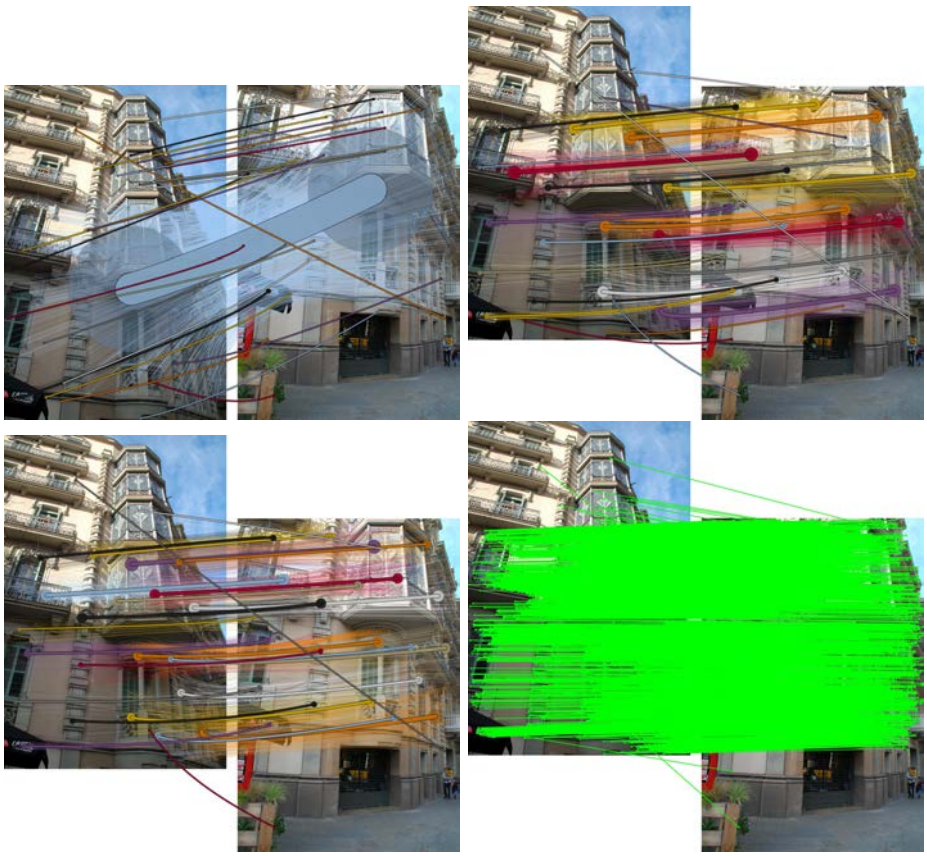
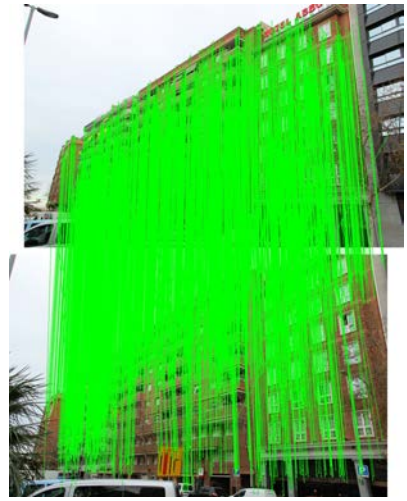


Figure 5.14: Aggregated segments using *min*, *avg* and *max* methods on a facade, compared with a baseline showing all matches. The resulting segments affect the optimized image layout, due to the varying aggregation of outlier directions.

(1) Aggregated segments using *min* clustering. We can observe outliers that stand out such as *h, e, i, w, y* and others more subtle such as *b, k, m, o, p, q, r, s, u.*

(2) Aggregated segments using *avg* clustering. We can observe outliers that stand out such as *m, t, x.*

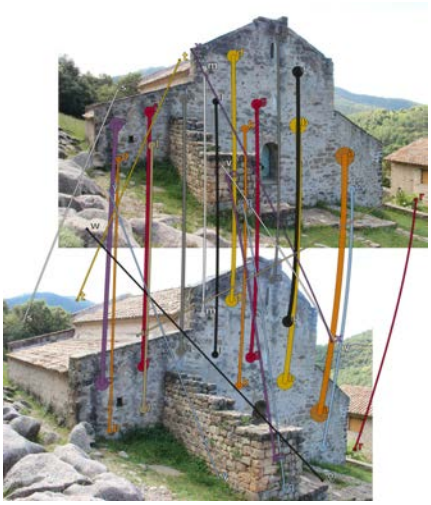
(3) Aggregated segments using *max* clustering. We can observe outliers that stand out such as *y, x.*



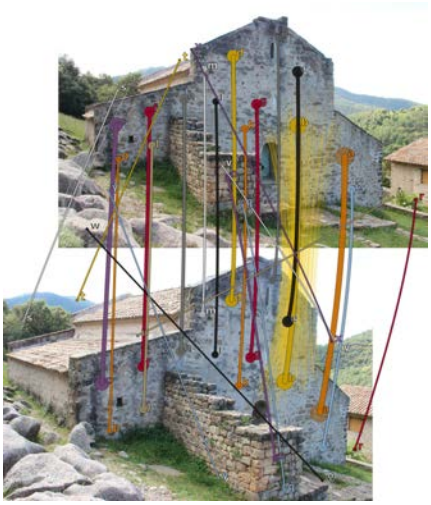
(4) Baseline approach. Just a few outliers can be distinguished.

Figure 5.15: Aggregated segments using *min*, *avg* and *max* methods on a facade, compared with a baseline showing all matches. Our output includes also glyphs to identify segments.

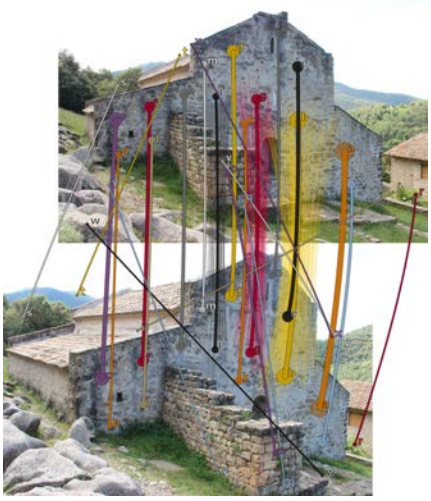
Finally, in Figure 5.16, we try to demonstrate the benefits of using a hierarchical clustering approach in an interactive application. The aggregated segments are refined upon selection to show the underlying matches.



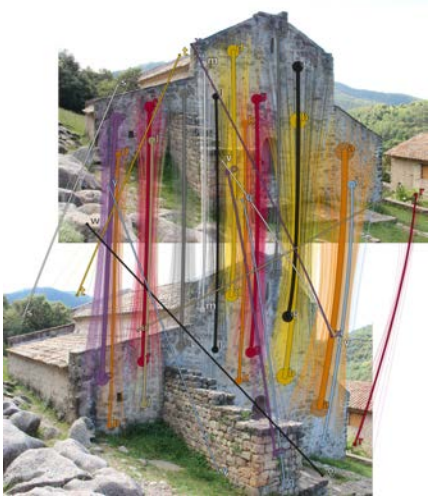
(1) Visualization generated with no uncollapsed segments.



(2) Visualization generated with a small amount of uncollapsed segments.



(3) Visualization generated with a moderate amount of uncollapsed segments.



(4) Visualization generated with a large amount of uncollapsed segments.

Figure 5.16: Benefits of hierarchical clustering: Pedret image pair with an increasing number of uncollapsed segments.

5.3 SEMANTIC-AWARE RECONSTRUCTION

When applied to building reconstruction, a significant limitation of photogrammetric techniques is that they require extensive manual editing to clean unwanted objects and improve the resulting reconstruction. We extend the traditional photogrammetric reconstruction pipeline by extensively exploiting semantic information for the task of facade reconstruction. In particular, we use a semantic segmentation of the facades into different classes. Typical classes in an urban scene include constructions (building, wall, and fences), flat objects (road, sidewalks), obstacles (persons, vehicles, poles, and traffic lights), nature (vegetation, terrain), and sky. Based on the class of each pixel, we modify its role throughout the pipeline. For instance, this information can be used to decide which image parts should be ignored (e.g., trees), where they should be ignored (static vs. dynamic objects), and how they should be reconstructed (e.g., applying planar priors on building facades).

In particular, the proposed pipeline uses semantic data for the following tasks:

- Extract features by minimizing the influence of moving objects (vehicles, pedestrians) to avoid inconsistent feature matches that could affect camera pose estimation.
- Repair depth/normal maps produced during the dense reconstruction stage according to class-dependent priors.
- Remove the contribution of unwanted occluding objects (e.g., trees) from the dense reconstruction.
- Generate facade texture data by preventing the re-projection of occluding objects.
- Improve the building silhouette by trimming those parts labeled as sky.

5.3.1 Semantic-aware reconstruction pipeline

Now we revisit the pipeline described in Section 5.1 by including new steps and improving others (Figure 5.17) in order to exploit semantic information. These are highlighted in green bold. The rest of the steps follow a standard SfM+MVS pipeline (see [SF16; SZP+16]) and are described more briefly. We claim no novelty in these later steps.

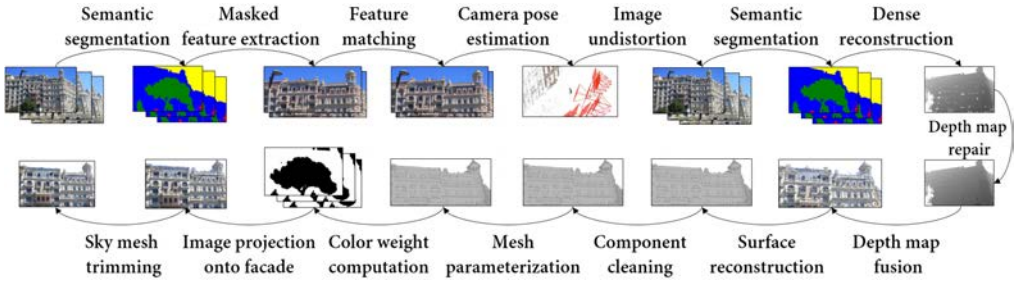


Figure 5.17: Overview of our proposed pipeline.

1 **Image acquisition:** Involves taking several pictures of the object to reconstruct with significant overlap between them. Diverse viewpoints help to reconstruct detailed features.

2 **Image segmentation:** This step is introduced to identify potential moving objects (vehicles and pedestrians).

3 **Feature detection and extraction:** We remove features corresponding to moving objects since their motion is not coherent with the rest of the scene.

4 **Feature matching between image pairs:** Finding correspondences between regions of different images.

5 **Sparse reconstruction:** Also known as *Bundle adjustment* is the process of jointly optimizing the 3D scene geometry and the camera poses and intrinsics.

6 **Image undistortion:** Correcting the lens distortion.

7 **Segmentation of undistorted images:** To identify non-facade pixels (e.g., sky, street, trees, furniture).

8 **Dense stereo reconstruction:** For each image, the depth at each pixel is estimated.

9 **Depth and normal map repairing:** By filling holes whose boundary pixels belong to the facade class.

10 **Depth map masking:** Cleaning by setting to null all pixels not having a facade label.

11 **Depth map fusion:** The estimated depths across multiple depth maps are checked for consistency to produce a point cloud.

12 **Meshing:** Converting the point cloud into a triangle mesh using a surface reconstruction algorithm.

13 **Color projection:** The segmentation mask is used to prevent the pixels from foreground objects from being pasted into the facades.

14 **Clipping building silhouettes:** By re-projecting all images again and trimming those parts labeled as sky.

Image segmentation

We extract semantic information from the input images using a state-of-the-art neural network (Xception-65 from DeepLab-v3+ [CPS+17]) trained using the ImageNet and Cityscapes [COR+16] datasets. This process must be applied before the photogrammetric pipeline's feature extraction step since it will use the resulting information. The classes used for segmentation are construction (building, wall, and fences), flat (road, sidewalks), obstacles (persons, vehicles, and objects such as poles and traffic lights), nature (vegetation, terrain), and sky.

Feature detection and extraction

We first considered segmenting the input images at the beginning of the photogrammetry pipeline and masking all non-facade pixels before extracting features. Implementing this option is straightforward and can be combined with any photogrammetric software (even closed-source) as a preprocessing step by, e.g., setting to a fixed color all non-facade pixels.

However, just masking pixels does not avoid (but reduces) features placed inside masked parts. It does not either totally avoid their reconstruction. This happens because most image features are computed considering some neighborhoods. Even more important is that feature matches between image pairs will solve for camera poses and intrinsic parameters. Therefore, masking large parts at this stage would result in less distinctive features and matches. This is especially serious if the overlap between image pairs happens within a masked region (Figure 5.18).

In conclusion, we decided not to mask *static* occluding objects (e.g., trees) before feature extraction to preserve the contribution of their features to the sparse reconstruction process.

Figure 5.18: These two images capture usable scene content from two slightly different viewpoints. Since the tree occupies most of the overlap region, masking it before feature extraction would result in no feature matches between them.

Nonetheless, *Moving* objects (vehicles, pedestrians) might interfere with camera pose estimation. Thus, we mask them at this stage (Figure **5.19**). In our pipeline, we implemented this step by providing a per-image mask corresponding to vehicles and pedestrians to the feature extraction process.

Figure 5.19: Moving objects such as vehicles lead to inconsistent feature matches between image pairs. We identify pixels belonging to vehicles and pedestrians and minimize their influence on image features.

We extract SIFT[Low04] features for all input images. SIFT keypoints correspond to local maxima/minima of the Difference of Gaussians at multiple scales.

Feature matching between image pairs

Feature matching and geometric verification find correspondences between feature points in image pairs. We do not assume having any prior information to guide the matching process. Hence we perform a brute-force matching checking all images against each other (exhaustive matching). This process has a quadratic cost but often generates versatile results.

Sparse reconstruction

The next step consists of reconstructing a sparse model starting from an initial image pair. The model is incrementally extended by registering new images and triangulating new points. We used Colmap for this task. The output includes camera parameters for all images.

Image undistortion

All images must be undistorted before dense reconstruction. In our experiments, we modeled all cameras using a simple radial model with one focal length and one radial distortion parameter. From this point, the remaining steps will all work with undistorted images.

Segmentation of undistorted images

We perform a second segmentation step (as described in Subsection 5.3.1) but, this time, on the undistorted images. The pixels on the undistorted segmentations will map one-to-one to the reconstructed depth map pixels. This property will allow removing unwanted objects directly on the depth maps or the reconstructed mesh.

Depth stereo reconstruction

The camera poses of the input images serve as the input to compute the depth and normal maps using stereo [SZP+16].

Masking images before dense reconstruction does not entirely prevent masked objects from being reconstructed (Figure 5.20), and thus is not completely effective. Therefore, we discarded this option.

Depth and normal map repairing

Unfortunately, the depth maps generated by MVS techniques usually contain some defects which are not desirable for our goal. These include unwanted occluding objects (such as trees and light poles) and missing parts on low-texture/mirror-like surfaces.

We remove the depth values that we identify as showing occluding objects (sky, trees, vehicles, pedestrians, furniture). While these objects are effectively removed from the final dense point clouds (Figure 5.20), they still appear on the original color images. Consequently, when re-projecting these images to colorize the reconstructed models, these undesired objects become pasted into the background.

In our pipeline, we implement this step by masking (setting to a null depth) all depth map pixels located within all non-facade regions.

Building facades are usually planar. We propose using this domain knowledge to repair the holes in the depth maps caused by poorly textured areas and windows. Pixels on the contour of a hole can provide information about the

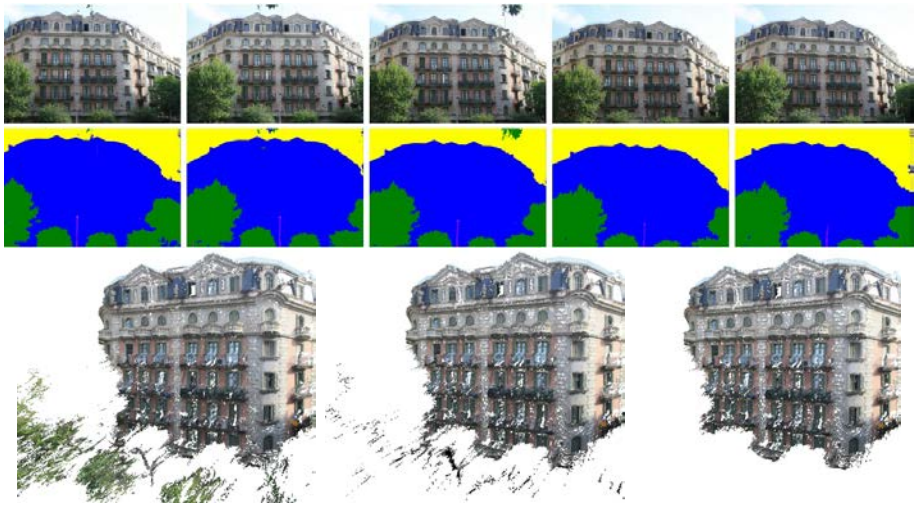


Figure 5.20: Masking occluding objects before dense reconstruction reduces but does not eliminate their impact on the dense point cloud. Masking the output depth maps do effectively remove masked objects. From top to bottom: **(1)** Input images. **(2)** Segmentations. **(3)** Reconstructions, from left to right: **(3.1)** With original images. **(3.2)** Masked images. **(3.3)** Masked depth maps.

underlying geometry. However, they may belong to either the facade or other occluding objects. Hence, we use the segmentation in this step to confine the repair of holes to pixels segmented as facade.

We first apply a border-following algorithm [SA85] to extract all the contours separating valid regions from non-valid ones. Contours are represented as a list of 2D points and might include other contours and define deep inclusion hierarchies.

Points belonging to a contour might be a combination of valid depth values and null values due to non-valid regions surrounding valid ones and vice-versa. We will repair a contour if:

1. It contains multiple points with valid depth values.
2. It encloses pixels labeled as facade in the segmented image.
3. The area of the bounded region is below some threshold.

We compute a regression plane using RANSAC on the 3D coordinates of the 4-neighbors of all contour boundary points. We only consider points with

valid depth values and labeled as facade. This plane is used to replace the contour's interior missing values. However, tiny contours, which have a smaller contribution to the overall quality, are handled separately using the depth average of valid neighboring points.

All depth map changes are propagated to the associated normal maps for consistency.

Depth map masking

We want to minimize the number of occluding and unwanted objects in our scene. In the semantic segmentation, these object usually belong to classes different than constructions. Examples o these are roads, sidewalks, obstacles, vegetation, and sky. However, we could allow the user to preserve a subset of these (e.g., vegetation for facades showing vertical gardens).

Depth map fusion

Valid depth map points are unprojected and checked for depth/normal consistency with points from other images. The consistency check is successful when n or more pixels contribute to a point. Thus, the point is written to the output point cloud. The final color and normals are the average of the contributing pixels.

Meshing and Mesh Trimming

We use Screened Poisson Surface reconstruction [KH13] to mesh the point cloud. This algorithm defines an indicator function and then obtains the reconstructed surface by extracting an appropriate isosurface. The implicit function is assumed to be smooth. Therefore, the algorithm may fail to recover detailed geometry features when the point density is low. Here, the reconstructed surface will show a curved appearance that overfits the available position and normal data. Another problem is that the smoothness assumption does not apply to building facades, where a small set of orientations are predominant. Fortunately, our repaired depth maps yield dense enough point clouds to capture most facade details.

Reconstruction methods based on an implicit function are robust against data noise. However, they usually attempt to create water-tight surfaces even when the point cloud only captures a few objects' sides. For building facades, photographs shot at street-level do not capture the building roof or the building's backside. Moreover, these parts around the facade's silhouette correspond to low-density regions with poor color reproduction. Hence, Poisson reconstruction wrongly tends to mesh them with smooth surfaces that must be trimmed.

Removing small components

We keep only the largest connected surface component, which we assume corresponds to the building facade.

Re-projecting images

Photogrammetry applications can generate a high-quality texture from the input images. First, they parameterize the reconstructed mesh, and, second, they re-project the images on top of it while filling the texture. Thanks to this, the reconstructed facades can be rendered using the high-resolution color encoded in the texture instead of with per-vertex color (which requires a much more complex mesh).

There is one problem that arises from the occluding objects which have been masked in the depth maps. Image re-projection uses depth checks to compute the color to assign to each triangle. Nevertheless, this information is no longer available for the objects masked during dense reconstruction. One solution would be using the original (unmasked) depth data. However, this is unreliable for thin structures (tree leaves and branches). Hence, we address this issue as a part of our color weighting scheme.

Besides classic weights (pixel view angle, pixel view distance, pixel distance to image borders), we use a weight mask to nullify non-facade pixels' contributions. This mask also allows a smooth transition by weighting down pixels close to the boundary between facade and non-facade pixels.

In particular, we use a radius r that controls the thickness of the transition zone. Non-facade pixels receive a zero weight while facade pixels at a distance greater than r from the border receive a weight of one. In the transition zone, we compute the weights applying a smoothstep function based on the distance to the border. A radius of 25 pixels (for over 3000-pixel images) has given good results in our experiments.

Clipping building silhouettes

We trim the surface corresponding to sky-labeled pixels. We re-project the segmented images onto the mesh and remove the vertices (and their incident faces) classified as sky.

5.3.2 Results and Discussion

We provide an implementation of our pipeline in Python, which uses the C++ implementation of Colmap. All tests were run on a commodity PC equipped with an Intel i7-8700 CPU and an NVIDIA TitanV GPU. We used Colmap for SfM (on CPU) and MVS (on GPU), TensorFlow (on GPU) for image segmentation, and Python (on CPU) for the rest of the tasks, including image masking, depth map repairing, and cleaning, and color weight mask generation.

Most of our input images represent building facades from European cities. We downsampled these to a resolution of about 3000×2000 pixels. Each dataset corresponds to a different facade and contains ten to fifty street-level photographs. Generally, the photographs' layout is similar to the Cityscapes images used to train the segmentation CNN.

The average running time for the standard Colmap pipeline (default parameters, maximum image size of 3000 pixels, 13 octree levels in Poisson Surface Reconstruction) was about 40 minutes (5' for SfM, 35' for MVS). Our pipeline (same parameters) added about 6 minutes to the process.

We tested major photogrammetry software on an extensive collection of European building facades. Figure 5.21 shows examples of depth maps automatically repaired with our algorithm. We successfully repair most missing parts in facades, whereas the pixels of other classes are not changed. Figure 5.22 shows our results after the fusion of the repaired depth maps. Windows and texture-less patches have been reconstructed successfully.

In Figure 5.23, we show how using our color weight masks avoids occluding objects from being pasted onto the background. Namely, these objects do not appear on the final mesh textures.

In Figures 5.24 and 5.25, we compare our results with a baseline Colamp reconstruction. Our pipeline improves the reconstructed model's geometric quality by producing fewer missing parts (thanks to depth map repairing) and removing removes most occluding objects such as trees. Moreover, we also achieve overall better color quality and remove unwanted objects from the facades' textures.

In Figure 5.26, we also compare our method against other state-of-the-art open-source packages such as OpenMVG and AliceVision Meshroom [MMM12; JP11] and some popular commercial solutions, such as Agisoft Metashape Pro, Zephyr3D [STM+19; TGF+15], Reality Capture, and Autodesk Recap Pro 2020. Our method can match the best color reproduction while reconstructing less sky, reconstructing missing parts, and removing unwanted objects like trees.

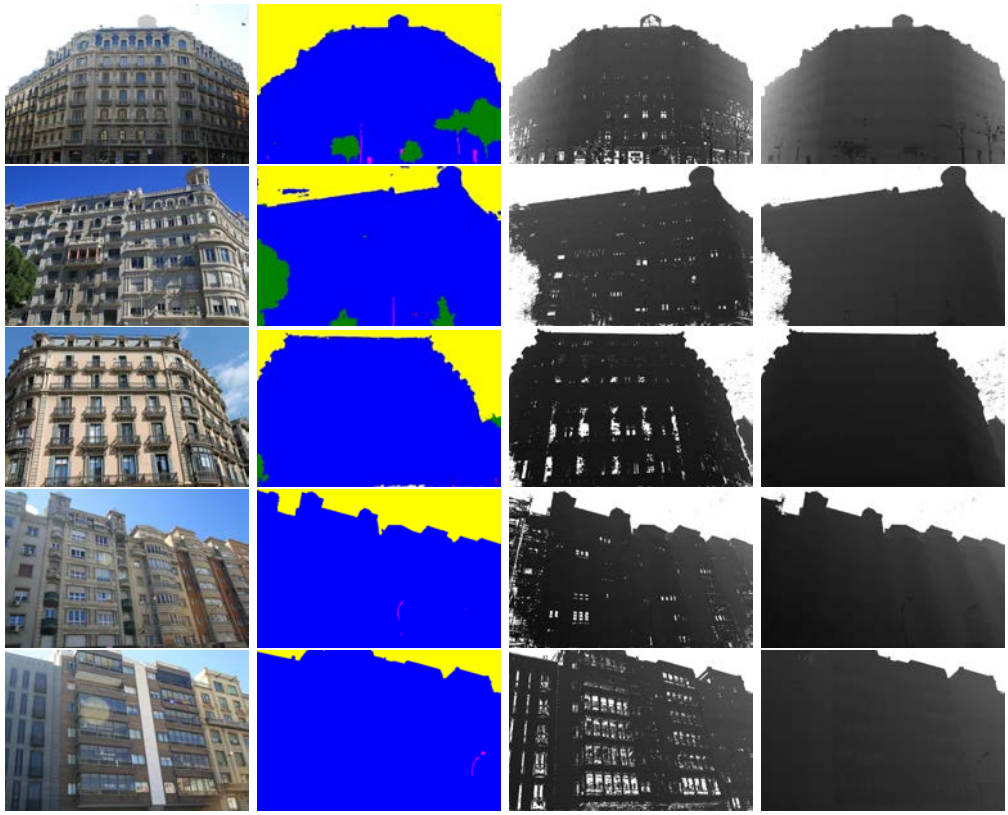


Figure 5.21: Repairing results. From left to right: **(1)** RGB image. **(2)** Segmentation. **(3)** Original depth map. **(4)** Repaired depth map.

Figure 5.22: Impact of depth map repairing. From left to right: **(1)** Point cloud from original depth maps. **(2)** Point cloud from repaired depth maps. **(3)** Reconstructed mesh. **(4)** Close-up view.

Figure 5.23: Impact of the semantic segmentation mask for weighting pixel contributions. From left to right: **(1)** Reconstructed model (Colmap) with per-vertex color. Color pollution on the facade near the tree is hardly visible. **(2)** The output of our pipeline but disabling pixel-weighting when generating the texture. Overall color reproduction is much better but the left side of the facade has serious color artifacts. **(3)** The output of our complete pipeline; despite trees were not reconstructed, tree pixel colors do not contribute to the facade.

Unfortunately, some unwanted objects like streetlights and some sky pixels were not correctly classified during the segmentation step. Consequently, they were not completely removed in the final reconstruction.

Some of the reconstruction improvements we achieve automatically by exploiting semantic information could be addressed with different workarounds and manual editing. For instance, moving objects could be outlined manually with an image editor. The resulting mask would be passed to the photogrammetry software to remove keypoints during feature extraction. Static occluding objects, such as trees, could also be masked manually. However, the resulting masks need to be reasonably accurate for model cleaning and image re-projection. Two masks are required for each image, one for the original and one for the undistorted versions. Hence, the amount of manual work would be considerable.

Various techniques can fill the gaps in low-texture regions and mirror-like surfaces (which we repair after depth map creation). Alternatively, we could also fine-tune some parameters during dense reconstruction. For example, we could increase the patch window radius (at the expense of performance). However, parameter tuning is a tedious task for end-users.

Poisson surface reconstruction can fill most of these gaps. However, the succeeding surface trimming step will remove some of them (trimming with low-density thresholds) or leave large faces around the model (trimming with large density thresholds). Standard hole-repair techniques can also handle gaps. However, these techniques applied to the reconstructed mesh are agnostic about the objects in the scene. Thus, they cannot distinguish missing parts from real holes nor use consistent shape priors to repair the model. Figure **5.27** shows a reconstruction where some missing parts (due to tree branches) do not form a closed loop and thus cannot be identified as holes.

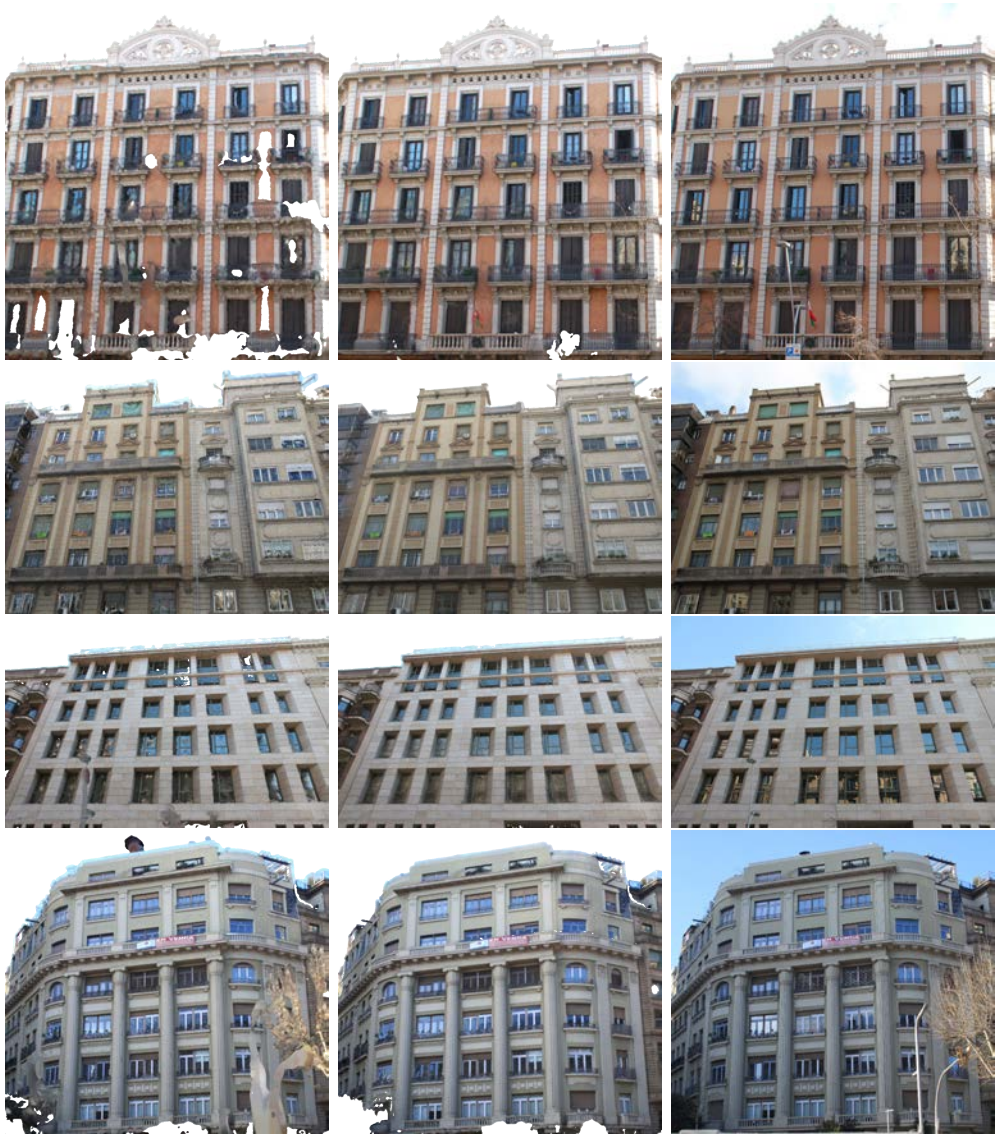


Figure 5.24: Effects of our semantically-aware segmentation on final reconstructed models. From left to right: **(1)** Results without using segmentation. **(2)** Our results. **(3)** An actual reference image of the building.

Figure 5.25: Effects of our semantically-aware segmentation on final reconstructed models. From left to right: **(1)** Results without using segmentation. **(2)** Our results. **(3)** An actual reference image of the building.

Figure 5.26: Comparison between different photogrammetry software, such as Alicevision Meshroom [MMM12; JP11], Agisoft Metashape Pro 1.5.5 [1], Autodesk ReCap Photo 2020 [2], 3DFlow Zephyr [STM+19; TGF+15], VisualSFM [WAC+11; Wu13]+CMVS+Meshlab [CCC+08], Colmap [SF16; SZP+16], and our pipeline.

An additional benefit of depth map repairing is that classic depth and normal consistency checks during fusion prevent poorly repaired regions from becoming part of the dense point cloud. This happens because these faulty regions are highly unlikely to be consistent across multiple views.

Figure 5.27: Left: reconstruction of a facade using Colmap. Some missing parts (due to tree branches near the facade) extend up to the border of the model and do not form a closed loop. Right: our pipeline succeeds in generating a complete mesh thanks to the depth repairing step.

It is also possible to manually choose a subset of images for creating the facade texture. Although this task seems doable in a few minutes, the exclusion of some images (e.g., because they show a tree) also prevents usable facade pixels from being used.

In summary, we claim that our fully-automatic pipeline provides significant benefits for facade reconstruction, therefore, avoiding or minimizing editing efforts.

Limitations

Our tests were limited to facades from European cities (such as Barcelona, London, Paris, and Madrid) whose appearance resembles those in the Cityscapes dataset. Thus, we have not examined the generalization of the segmentation network and, consequently, of our pipeline to other facade styles.

Most photographs on our test sets were taken at street-level, also similar to Cityscapes images. Thus, we miss all the information from roofs or the upper

part of balconies and terraces. Moreover, we avoided capturing pedestrians and focused on upper-facades, which is needed to release the image dataset later. Thus, we also miss most of the information from storefronts. Consequently, we were unable to evaluate the reconstruction in these areas.

We also observed that some misclassified regions could affect the quality of the reconstructions. We noticed that some sky pixels near the image boundaries are often assigned the wrong class. Also, sometimes thin occluding objects, such as light poles, are not fully segmented. A part of these misclassified regions can appear in the reconstruction, which motivates the succeeding component-based cleaning.

5.4 PHOTOGRAPHY-TO-LIDAR REGISTRATION AND TEXTURING

Different acquisition approaches have different advantages. For instance, we have seen that LiDAR technology yields point clouds with high-quality geometric detail with low-quality color information. Moreover, these point clouds are redundant, unevenly sampled, and, usually, incomplete. Consequently, it is laborious to capture a moderately complex scene completely. One option is performing scans at different locations to increase coverage, but this will also increase the redundancy in some regions.

Better results may be achieved by combining LiDAR point clouds with those obtained from photogrammetry. Photogrammetry-based reconstructions only require a medium-quality camera. Therefore they are easier to capture. We could potentially use these to complete the missing parts of LiDAR point clouds or improve their color information. The only requirement would be having all of them in the same coordinate space.

Directly registering both types of point clouds presents several challenges. Uneven point densities, scale differences between datasets, noise, outliers, and missing data make this process difficult. One possibility would be using the Scale Iterative Closest Point (SICP) algorithm to register these datasets obtained from different sources [SPT15]. Another option would be using robust statistics to explore the space of similarity transformations [AGV+14]. It is also possible to match regions of them using a graph representation of their micro and macro structures [HZF+17]. Once a match is found, the registration can be refined using other techniques [HZW+17; HFW+19].

Instead of registering geometry to geometry or photography to geometry, we propose including the registration process directly in the photogrammetric pipeline.

5.4.1 Registration algorithm

We want to have the input photographs, the LiDAR model, and the photogrammetric model in the same coordinate frame. First, we will estimate each photograph's camera parameters matching the position on the LiDAR model. Second, the photogrammetric reconstruction on these images will directly yield a point cloud in the same frame.

We assume we have a point cloud C, which resulted from the union of a set registered scans from locations $Q \subset \mathbb{R}^3$, namely $C = \bigcup_{\mathbf{q} \in Q} C_{\mathbf{q}}$.

The first step consists of generating a cube map for each different scan location **q** within the 3D model (Figure 5.28). We sampled the cubemaps directly from the equirectangular panoramas provided by the laser scanner equipment. We did this by converting from spherical coordinates to 3D coordinates and then to cubemap coordinates. Alternatively, if the scanning technology does not provide these panoramas, they could be generated directly from the point clouds Section 4.3. Panoramas (and thus cubemaps) are already registered with the LiDAR point cloud.

Figure 5.28: Sample panorama and extracted cubemap images from the *museum* model. ("Sant Quirze de Pedret´´ mural paintings at "Museu Diocesà i Comarcal de Solsona´´). The image corresponding to the cube's bottom face is not used as it shows the scanner tripod.

The next step is to extract features for all the cubemap images. The feature extractor software (Colmap) is configured to use a simple pinhole camera model with known parameters. For a $W \times W$ cubemap image, its focal length is W/2, and the principal point is located at (W/2, W/2).

Although we know the ground truth extrinsic parameters of the cubemap images, we still use photogrammetric software to match features between the cubemap images and create a sparse representation of the model [SF16].

Next, we run the feature extraction and matching steps for each input photograph and then register them against the existing sparse model. Intrinsic and extrinsic parameters are estimated for each new image. Notice that, at this point,

we have registered all images with the LiDAR point cloud. Figure 5.29 shows the estimated camera poses for a collection of user-provided photos.

The rest of the pipeline follows the same steps described in Section 5.3 after sparse reconstruction (5). The resulting dense point cloud will be in the same coordinate frame and scale as the cameras. Hence it will also be in the same space as the LiDAR model.

5.4.2 Photography projection

Additionally, photographs made with high-quality cameras can have higher definition and sharpness than the mesh color extracted from the panoramas provided by the scanning technology. Hence, if enough photographs are taken, these can be projected onto the mesh to compute a high-detail colorization (Figures 5.30 and 5.31).

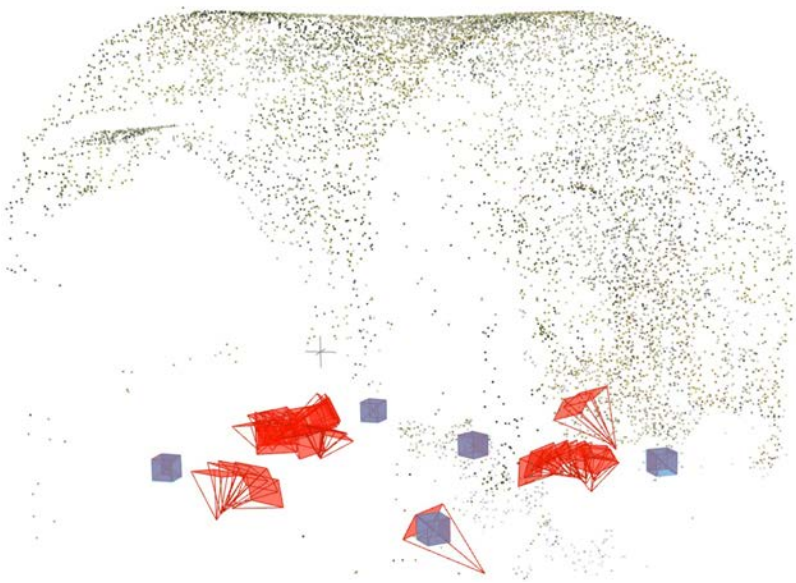


Figure 5.29: Sparse point cloud, cameras corresponding to cubemap images (in blue) and estimated camera poses for user-provided images (in red).

Figure 5.30: Small slice of the *museum* model (Museu Diocesà i Comarcal de Solsona). From top to bottom: **(1)** Original mesh color (extracted from the panoramas captured at scan time), **(2)** sample photograph registered to the geometry and **(3)** resulting color from projecting the different images onto the mesh.

Figure 5.31: Altarpiece from "La Doma" church at La Garriga (Barcelona). From top to bottom: **(1)** Original mesh color (extracted from the panoramas captured at scan time), **(2)** sample photograph registered to the geometry and **(3)** resulting color from projecting the different images onto the mesh.

5.5 PUBLICATIONS

Our contributions to photogrammetry reconstruction have led to three publications. The first one was presented at the local conference "CEIG - Spanish Computer Graphics Conference 2018". It contains some initial ideas on how to apply semantic information to improve the quality of the reconstructed models:

- C. Andújar, O. Argudo, I. Besora, P. Brunet, A. Chica, and M. Comino. "Depth Map Repairing for Building Reconstruction". In: *Spanish Computer Graphics Conference (CEIG)*. The Eurographics Association, 2018

The second one was presented at the "EuroVis 2020" conference. This conference was planned to be held in Norrköping but was held virtually due to the 2020 COVID-19 pandemic. It describes our method for improved visualization of correspondences between images:

- C. Andujar, A. Chica, and M. Comino. "Effective Visualization of Sparse Image-to-Image Correspondences". In: *EuroVis 2020 - Short Papers*. The Eurographics Association, 2020

The third one was presented at the "Eurographics Workshop on Graphics and Cultural Heritage". This conference was planned to be held in Granada but was held virtually due to the 2020 COVID-19 pandemic. It describes our method for registering high-quality pictures to a LiDAR model, which can be used to create short narratives:

- M. Comino, A. Chica, and C. Andujar. "Easy Authoring of Image-Supported Short Stories for 3D Scanned Cultural Heritage". In: *Eurographics Workshop on Graphics and Cultural Heritage*. The Eurographics Association, 2020

We also plan on publishing an extended version of the first conference paper as another article.

6

Interactive Visualization of Point Clouds

In previous chapters, we have discussed how point clouds captured by LiDAR devices cannot be easily handled due to its high-resolution and require simplification. We have also seen how to encode this high-frequency detail into textures before simplification. In this Chapter, we will show how to effectively render these simplified models while reproducing the high-frequency detail and ensuring an interactive experience.

Massive point clouds cannot be rendered and inspected without resorting to out-of-core acceleration techniques. The main reason behind this is their large memory footprint, which makes it impossible to fit them in main memory. Such techniques are only available in high-end point-based rendering tools [MVV+15].

Acceleration techniques commonly include the use of GPU for splat rendering [PJW12], visibility culling [RL00; KTB07], and hierarchical representations to quickly retrieve suitable points at different levels of details [RL00; GM04; GEM+13; RDD15; DRD18].

While these methods generally perform well (e.g., visualizing a single digitized statue), they may struggle for the particular case of exploring buildings and urban models. In this setting, interactive navigation usually consists of fast camera movements, requiring constant loading of a substantial amount of point data from the disk/network. Due to the associated latency, maintaining an ideal one-sample per pixel ratio can be challenging. Hence, large splats must be used to cover large screen areas. We have noticed that existing point-based rendering approaches almost always use a single color for each point (i.e., flat-colored splats). This causes a large part of high-frequency color detail to be

omitted until all data has moved to main memory, even if it is vital for many urban models.

Point clouds generated with Multi-view Stereo techniques are usually much smaller and will fit in memory. Nonetheless, they could also benefit from specially designed visualization techniques. On the one hand, we can usually not process our raw images at maximum resolution due to hardware constraints. On the other, these techniques may fail to reconstruct the geometry on specular surfaces and texture-less areas. Even if we repair these, the result is that the raw color information will usually have a higher frequency than the computed geometry.

In summary, the types of clouds we consider have the special characteristic of combining lower-frequency geometry as points with associated higher-frequency detail in the form of textures. In this Chapter, we study how to exploit these characteristics in interactive visualization algorithms. The rest of the Chapter is organized as follows:

1 **Rendering and Interactive Inspection of Panoramas:** Image-based rendering techniques are a powerful, well-known family of methods that synthesize images from new viewpoints by reprojecting the pixels of a set of input images. In Section **4.3**, we have seen how to encode point clouds properties into a set of textures, and in Section **6.1**, we study how to use these to produce high-quality interactive renderings.

2 **View-dependent Hierarchical Rendering through Textured Splats:** Instead of a fully-image-based method, in Section **6.2**, we study the benefits of a hybrid approach. In particular, we combine a hierarchical representation, built by simplifying the input cloud at different ratios, and the high-resolution detail encoded into textures.

6.1 RENDERING AND INTERACTIVE INSPECTION OF PANORAMAS

In this section we present a method inspired in image-based rendering techniques. For LiDAR point clouds, we first use the approach presented in Section 4.3 to encode normal, color and depth information into a different panoramic images. One triplet of images is generated for each scan location. More formally, consider a point cloud C which results from the union of a set registered scans from locations $Q \subset \mathbb{R}^3$, namely $C = \bigcup_{\mathbf{q} \in Q} C_{\mathbf{q}}$. For each scan location $\mathbf{q}$ we generated a set of textures $T_{\mathbf{q}} = \{T^n_{\mathbf{q}}, T^c_{\mathbf{q}}, T^d_{\mathbf{q}}\}$.

For Multi-view Stereo generated point clouds, we use the depth maps generated using our improved approach presented in Chapter 5, the color images at the original resolution, and the camera projection matrices calibrated using Colmap [SF16]. Moreover, we have exported an extra property for each point, indicating the id of its source image.

There is a whole family of existing methods for the rendering of RGB-D images. Shum et al. [SK00] and Gledhill et al. [GTT+03] present some older surveys for rendering single panoramas and multi-perspective panoramas.

This task is very closely related to the problem of *novel view synthesis* in the Computer Vision literature. One State-of-the-Art method presented by Penner and Zhang [PZ17] considers that the estimated depth maps associated with the input images define a 3D probability function that indicates where the real surface is located. One depth probability volume is computed for each input view, which, for each pixel, contains the probability of the surface it represents being at a certain depth. These volumes are computed by accumulating information from the depth maps of their corresponding neighboring views. Each depth map pixel provides information about where the surface is located and the free space in front of it. Then, the consensus for each voxel (probability of it belonging to a surface) is computed. They use the ratio between the number of views placing a surface on the voxel, and those that see it. Finally, these probabilities are used to reproject the color images obtaining a soft reconstruction.

However, newer approaches usually include Deep Learning elements. For instance, DeepStereo [FNP+16] learns to generate new views from a stack of plane-sweep volumes generated by reprojecting each input image using different depths. This algorithm is not suitable for interactive rendering (authors mention a cost of 12 minutes to generate an image patch of 512x512).

Instead of using a fully-learned system, other authors [ZTF+18; CGT+19] present strategies that combine traditional elements with learned pieces. Zhou et

al. [ZTF+18] and Choi et al. [CGT+19] use intermediate representations similar to depth probability volumes [PZ17]. Nonetheless, whereas the first one uses learning to generate such representation, the second generates it traditionally, and learning is used to refine views produced from it.

Zhou et al. [ZTF+18] propose a network architecture that predicts the scene geometry by mapping the input depth maps into a set of planes. This representation is generated once and then used to generate multiple views during interactive rendering. Each map has an associated transparency mask, which is used back-to-front alpha compositing.

Choi et al. [CGT+19] propose a system that focuses on extreme novel view synthesis. They compute a probability volume and generate a first approximation of the final image using traditional methods. Then, each pixel's depth probability functions are used to retrieve patches from the original image. These are then fed to a network to produce a refined result. However, their method is not suitable for interactive rendering since it takes over a minute to generate a single view.

These methods are not suitable for our case for mainly two reasons. The first one is that most of them are too demanding for interactive rendering. The second one is that they are designed to tackle the unreliable nature of depth maps estimated using Multi-view Stereo. For LiDAR point clouds, the depth values can be reliably used without resorting to probabilistic approximations.

Instead, for our Multi-view Stereo generated point clouds, some of these methods could be used to inspect them interactively. However, it is not clear how they would benefit from having higher color resolution than depth resolution. Furthermore, Colmap [SZP+16] also estimates a robust 3D position out of the depth probability distribution for each pixel. In particular, they take the median of the positions provided by different views of each 3D point.

6.1.1 Single Panorama Rendering Algorithm

Single panorama views are one of the multiple ways reconstructed panoramas can be interactively explored. In this setting, we constrain the camera to remain at the center of projection to avoid the appearance of gaps in non-sampled areas. With the help of depth maps and normal maps, this constrained visualization helps understand the local geometry (which cannot be done by directly looking at the plain color images). It can also help the testing of illumination techniques (since the rendering requirements are relatively inexpensive).

We allow for rotations and zooms and support different shading techniques

such as Phong lightning, real-time shadow maps [Wil78] (see Figure 6.1) and screen space ambient occlusions [SA07] (see Figure 6.2).

Figure 6.1: Panoramas rendered without (left) and with (right) shadow mapping (mesh-based rendering). For single panoramas, shadows are only accurate for light sources nearby the sensor location.

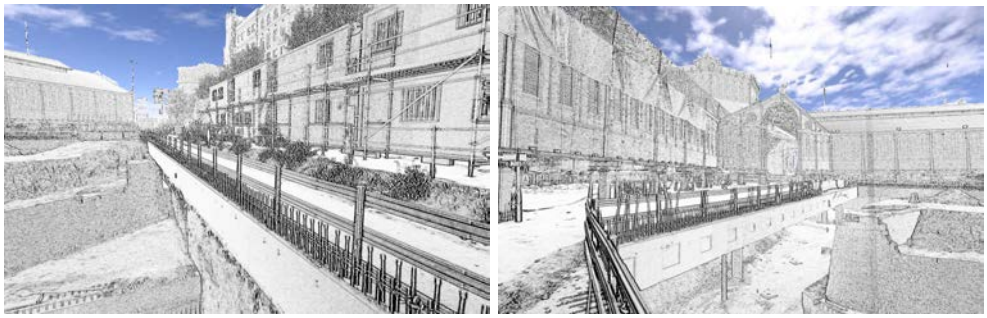


Figure 6.2: Panoramas rendered with an illustrative shader based on Screen-Space Ambient Occlusion with small radius. Robust normals are essential for getting good estimates of the ambient occlusion term.

6.1.2 Image-Based Rendering Algorithm

Allowing free camera movements can provide users richer experiences. For this exploration modality, we combine multiple panoramas captured from different locations. This combination provides a larger and more complete coverage of the scene (Figures 6.3 and 6.4).

Figure 6.3: From left to right: **(1)** A single panorama rendered from its center point **(2)** The same panorama rendered from a distant point **(3)** Adding additional panoramas to improve coverage and allow for free camera navigation. The original sensor locations are shown as color spheres.

Figure 6.4: Combining multiple panoramas to improve coverage. From left to right: **(1)** a single panorama **(2)** 2 panoramas **(3)** 5 panoramas.

Our OpenGL-based implementation feeds the graphics hardware with $V = u \times v$ vertices from a highly tessellated unit sphere, where u is the number of vertical segments (meridians) and v is the number of horizontal rings (parallels) on the sphere.

For a low-stretch panorama of $6h \times h$ pixels we would ideally use a sphere of $4h \times 2h$ vertices ($u = 4h$, $v = 2h$, $V = 8h^2$). Our usual panorama size is 12288×2048 pixels; hence, we would use a sphere of $V = 2^{25}$ vertices. The amount of GPU memory to store V vertices with their texture coordinates is $5 \times 4 \times V$ bytes, i.e., 640 MB. Using face indices, we would need to allocate $11 \times 4 \times V$ bytes, and, using triangle strips, we would need $10 \times 4 \times V$ bytes.

A more space-efficient solution is to insert in the VBO the vertices of a single ring, with attributes $(\theta, \sin\theta, \cos\theta)$. The resulting VBO takes $3 \times 4 \times 2u$ bytes (192 KB for the 12288×2048 panorama). In this case, the application draws the VBO v times per panorama, one for each ring. We send a total of six uniforms. The three values $(\psi, \sin\psi, \cos\psi)$ for each of the two rows of vertices corresponding to the ring being drawn. The *gl_VertexID* property can be used to pick the right set of values. This allows the vertex shader to recover the original unit-sphere vertices $(\bar{x}, \bar{y}, \bar{z}) = (\cos\theta\cos\psi, \sin\psi, \sin\theta\cos\psi)$ with no trigonometric function calls.

Texture coordinates must be computed at the vertex shader to retrieve the depth for each vertex. While these could also be recomputed for each fragment,

the interpolation process will introduce minimal errors for high tessellation rates. Moreover, this has the great advantage of not requiring any additional trigonometric computations. We can easily reformulate Equations (4.60) to (4.62) to only depend on ψ, θ and $\cos\psi$.

For the bottom cap ($\psi \leq -\pi/4$), $r_b = \frac{2}{\pi}(\pi + \psi)$:

$$f_{P_\mathbf{q}} = \left.\begin{cases} s = (\bar{x} r_b / \cos\psi + 0.5)/6 \\ t = (\bar{y} r_b / \cos\psi + 0.5)/6 \end{cases}\right\} \psi \leq -\pi/4 \tag{6.1}$$

Similarly, for the top cap ($\psi > \pi/4$), $r_t = \frac{2}{\pi}(\pi - \psi)$:

$$f_{P_\mathbf{q}} = \left.\begin{cases} s = (\bar{x} r_t / \cos\psi + 1.5)/6 \\ t = (\bar{y} r_t / \cos\psi + 0.5)/6 \end{cases}\right\} \psi > \pi/4 \tag{6.2}$$

Finally, for the equator:

$$f_{P_\mathbf{q}} = \left.\begin{cases} s = (\theta/(2\pi) + 0.5)/1.5 \\ t = (\psi + \pi/4)/(2\pi) \end{cases}\right\} -\pi/4 < \psi <= \pi/4 \tag{6.3}$$

This approach also offers flexibility to adjust the level-of-detail dynamically. The number of vertices V can be tuned to obtain a suitable trade-off between rendering speed and geometric detail. Moreover, a geometry shader can also perform interactive refinement. We could choose to render a coarser tessellation while interactively detecting when the viewpoint approaches a given surface. Upon getting closer than a certain distance to this surface, the geometry shader could be used to produce a finer tessellation.

In fact, for arbitrary viewpoints, a geometry shader is required. Faces with sharp depth changes appear extruded along a panorama's radial direction (rubber sheets). Let min, max be the min/max depth values sampled from the depth map at the triangle vertices. When $\max - \min$ is above some threshold ε_d, the depth of the three triangle vertices is set to min (see Figure **6.5** for ε_d values). This prevents large extruded surfaces while avoiding unnecessary gaps when rendering the panorama from the sensor location.

In the most basic scenario, the fragment shader samples the panorama's color and normal maps to perform lighting computations. Alternatively, the fragment shader may also use the available depth map to perform more advanced

Figure 6.5: Panoramas rendered from an offset from the sensor location, with varying depth thresholds ε_d. From left to right: **(1)** $\varepsilon_d = 50\,\text{cm}$ **(2)** $\varepsilon_d = 1\,\text{m}$ **(3)**$\varepsilon_d = 4\,\text{m}$ **(4)** $\varepsilon_d = 16\,\text{m}$ **(5)** $\varepsilon_d = 50\,\text{cm}$ (two panoramas).

illumination techniques such as ambient occlusion. Notice that our method is not very sensitive to using coarser tessellations (Figure **6.6**) since we use very high-resolution textures. Moreover, the detail can be enhanced by algorithms such as bump mapping.

Figure 6.6: Effect of reducing the number of vertices of the unit sphere used to render the panorama. From left to right: **(1)** 1024×512 **(2)** 512×256 **(3)** 256×128. In all cases we used a high resolution color map.

Panorama visualization also supports distance queries at any moment during the scene inspection. Upon user selection of two image pixels, the algorithm simply has to compute the two corresponding pixels q_1 and q_2 and their panoramas P_{q_1} and P_{q_2}. The required distance is the Euclidean distance between the 3D points associated with q_1 and q_2.

6.1.3 Results and Discussion

We carried out a performance study of our free-camera exploration tool based on combining multiple panoramas. We computed all reported times using a commodity PC equipped with an NVIDIA GTX 770 and a Full HD display. On Table **6.1**, we report point throughput and frame rates achieved for different unit sphere tesselation levels. These determine the number of depth values sampled from the depth maps.

We did not use any dynamic refinement strategy while producing these results. We found that we can approximately achieve a constant throughput of 500 M points (vertices) per second. This allowed rendering 4 panoramas at

N	2048×1024	4096×2048	8192×4096	16384×8192
1	480 (229)	495 (59)	503 (15)	537 (4)
2	486 (116)	503 (30)	536 (8)	537 (2)
3	491 (78)	503 (20)	503 (5)	402 (1)
4	495 (59)	503 (15)	536 (4)	536 (1)

Table 6.1: Performance of our rendering algorithm when rendering N panoramas at increasing sampling rates. The table shows both throughput (millions of points per second) and frame rate (fps, within parenthesis).

2048×1024 resolution while guaranteeing 60 fps. We observe that high-quality renders are generated from viewpoints that are far from the original sensor. Moreover, the visual quality does not severely degenerate when reducing the sampling rate. Hence, we theorize that we can increase the number of rendered panoramas by dynamically adjusting their rendering resolution. This way, we would roughly maintain the visual quality and frame rate while increasing the scene coverage.

6.2 VIEW-DEPENDENT HIERARCHICAL RENDERING THROUGH TEXTURED SPLATS

In the previous section, we have described a purely image-based method for rendering point clouds using panoramic images. Although it provides high-quality results, it has two main problems: on the one hand, a parameter ε_d must be carefully tuned in order to remove rubber sheets and, on the other, using a geometry shader for this task and the refinement of geometry becomes a bottleneck that strongly limits the performance.

This section presents a hierarchical point-based rendering method that employs textured splats at varying resolutions as rendering primitive. This allows us to produce good quality results while significantly improving the performance. The representation used for the geometry is our point clouds (simplified at different ratios), while the detail information (e.g., color, normals) is encoded in textures. The use of textured splats for point-based rendering is not new. However, to the best of our knowledge, the existing approaches using color textures [SSL+13; BLM+18] do not target real-time rendering, but the off-line creation of high-quality views for image-based localization.

Our work is probably closer to Gunnip et al. [GLY04], and Yang et al. [YGW06]. However, while they use projective texture mapping to color for their splats, our approach's key ingredient is keeping a representation of the color data for each scan (see Section **4.3**). This way, we can identify each point's source and associate a single texture to each of them. In Subsection **4.3.3** we have seen how to achieve consistent color across multiple views; hence, we only require one color texture access for each fragment.

Our approach combines a hierarchical representation of the point cloud geometry and textured splats to allow the efficient and scalable rendering of these clouds. We use textured splats to render much less but larger primitives. Nevertheless, we still obtain high-quality views with high-resolution image details. In summary, we can perform a more aggressive level-of-detail simplification with little impact on the final image quality. Notice that texture coordinates can be computed on a fragment shader using only the point sample coordinates and the scan location. Consequently, our approach can complement most hierarchical point-rendering methods.

6.2.1 Hierarchical Textured-Splat Rendering Algorithm

The core of most existing point-based hierarchical rendering algorithms consists of traversing out-of-core hierarchical structures. The level-of-detail of each element (point) is chosen based on its overall contribution to the final render. This contribution is usually computed as the screen-projected area of each element. When there is a high coherence between consecutive views (e.g., for small camera motions), these algorithms usually perform well (e.g., when visualizing a statue). However, we may perform swift camera movements when interactively exploring large urban models. In this setting, a large amount of points is continuously loaded and unloaded from the GPU. Therefore, rendering one element (raw point or a higher-level representative) for each screen pixel can become too expensive.

We designed our algorithm taking inspiration from the field of botanical rendering. Rendering leaves in botanical trees is strongly connected to rendering points. Both elements are unorganized, rendered in large amounts, and significantly impact the realism of a scene. There is a family of methods [CHP+07; NPD+11], which aim at reducing the rendering cost of vegetation leaves by *pruning* part of them and *scaling* the rest to preserve the overall appearance.

In hierarchical point-based rendering, the counterpart of *pruning* would be replacing a set of points with a higher-level representative. The counterpart of *scaling* would be enlarging the splats when the fragments generated from the resulting set of elements is not enough to cover the whole surface. Nevertheless, using enlarged flat splats produces blocky low-quality renders because the splat properties (color, normal) appear constant across the splat surface (Figure 4.31). We propose using our simplified point clouds (with different decimation factors) together with our texture-encoded high-resolution detail in order to improve any of the existing hierarchical rendering approaches and to allow more aggressive pruning of the cloud.

We draw each point $\mathbf{p} \in C_{\mathbf{q}}$ as an oriented quad. Our fragment shader receives the interpolated fragment position and the scan location $\mathbf{q}$. Then, it computes the per-fragment texture coordinates using the Equations (4.60) to (4.62). We use these coordinates to retrieve the fragment color and normal from the panoramic textures. The alpha channel encodes which fragments are outside of the model's silhouettes, so we discard those. Finally, the shader uses the distance from the fragment position to the center of the quad to generate either circular or ellipsoidal splats.

6.3 RESULTS AND DISCUSSION

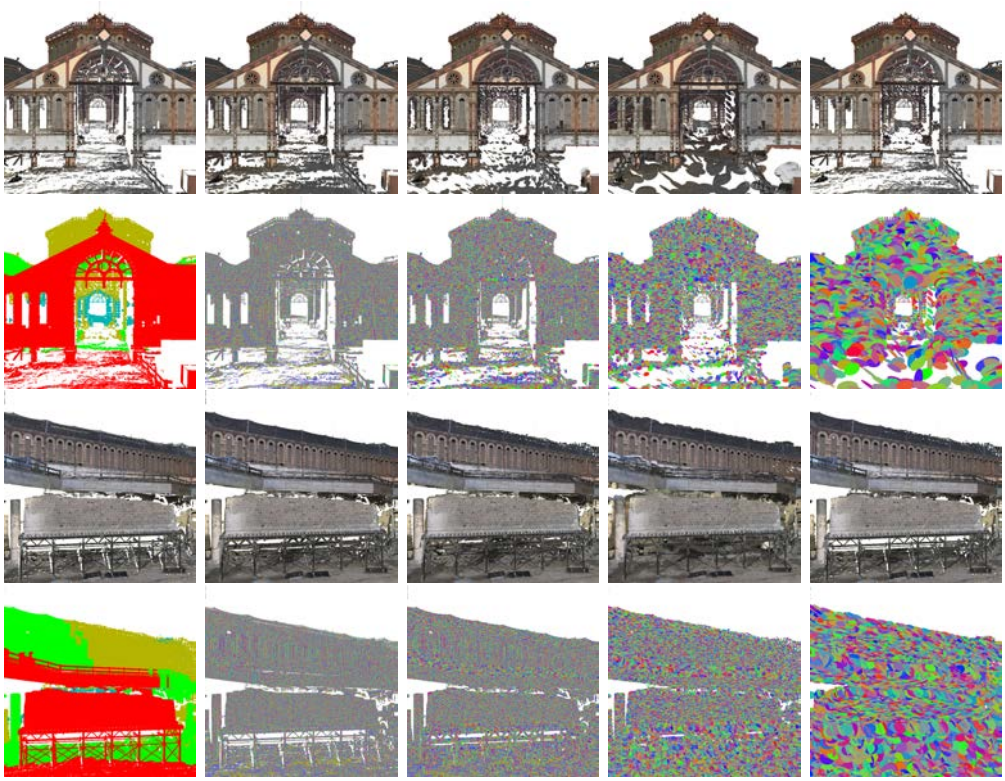


Figure 6.7: Renders of point clouds using multiple levels of detail. From left to right: **(1)** Render using our approach, the color scheme indicates original points (red) and decreasing levels of detail (green, yellow, blue). In **(2,3,4,5)** we show the results of rendering the same scenes choosing a single level-of-detail. In **(1)** we can see a render with only the original points. In **(3)** we show a render with a tenth of the original points, in **(4)** we use a hundredth and in **(5)** a thousandth.

We validated our model by implementing a straightforward point-based hierarchical rendering algorithm. We first generate a voxelization of the point cloud and store it out-of-core. We use a regular grid with cubic cells of $10^3 m^3$. We apply our simplification algorithm (Section **4.1**) to produce three levels of resolution for the cloud for decimation factors $\lambda = \{0.1, 0.01, 0.001\}$. These three levels of resolution usually fit in main memory (for our test clouds), whereas the original one does not, and its voxels are loaded on demand.

When the user interacts with the application, we render the cells that overlap with the view frustum in front-to-back order, which benefits from early depth culling. We use the distance from the cell center to the viewpoint to select the

most appropriate level-of-detail for the cell dynamically. The distances between the closest points within a resolution level are similar. Consequently, we use this distance as the scaling factor for the splats at each level-of-detail.

We render a textured splat for each point, and we use the alpha channel of the texture to preserve its original silhouette. When the interaction is stopped, and the viewpoint is held still for a few seconds, we refine the rendered view. We do this by loading from disk and displaying the cells containing the points at the original resolution.

We have tested our algorithm on a commodity PC equipped with an Intel Core i7-4790K CPU, 32GB of RAM, 16GB of swap memory, and a 2TB Toshiba DT01ACA200 HDD running Ubuntu 20.04. The test dataset included 31 different 3D scans from a singular XIX century market. This market has an extent of 5,214 m^2 on a city block of about 15,876 m^2. A Leica ScanStation P20 was used for digitizing key parts of the building. The raw dataset included 31 ASCII files containing information on 3.487.095.733 points and requiring a total of 157.3 GB. We used the point clouds acquired from 4 different outdoor locations for our tests, amounting to roughly 150 million points. We chose the outdoor scans to illustrate the preservation of fine silhouette details when large splats are trimmed with the alpha mask. Using the interactive inspection mode, we achieve frame rates of up to 600 fps for far-away views and from 300 to 600 fps for closer ones. When the interaction is stopped, it can take several seconds to load and display the finest level-of-detail. Fortunately, the use of textured splats makes this last refinement less critical than with classical single-colored splats.

Figure 6.7 shows multiple renders of this scene, with different resolution levels. Note that even for the level with the most aggressive pruning (keeping one point out of one hundred samples), the final render quality is suitable for areas with an approximately uniform sampling density.

6.4 PUBLICATIONS

Our contributions to point cloud visualization have led to two publications. The first one was published on the special issue on "Special Issue on Large-Scale 3D Modeling of Urban Indoor or Outdoor Scenes from Images and Range Scans" of the Computer Vision and Image Understanding Journal. It describes our image-based rendering approach using panoramic images:

- M. Comino, C. Andujar, A. Chica, and P. Brunet. "Error-aware construction and rendering of multi-scan panoramas from massive point clouds". In: *Computer Vision and Image Understanding* 157 (2017). Large-Scale 3D Modeling of Urban Indoor or Outdoor Scenes from Images and Range Scans, pp. 43–54

The second one was presented at the local conference "CEIG - Spanish Computer Graphics Conference 2019". It describes our rendering approach using textured splats:

- M. Comino Trinidad, A. Chica Calaf, and C. Andújar Gran. "View-dependent Hierarchical Rendering of Massive Point Clouds through Textured Splats". In: *Spanish Computer Graphics Conference (CEIG)*. The Eurographics Association, 2019

We are currently working on improving our work on rendering using textured splats, and we would like to publish it as another article.

7

Conclusions and Future Work

7.1 CONCLUSIONS

When we first devised this thesis, our primary goal was to digitize urban scenes and overcome the shortcomings of the techniques used for this digitization. We wanted to make the captured models as useful as possible. In particular, our definition of "useful" comprises making these models suitable for various tasks ranging from interactive inspection for cultural heritage to virtual tourism. Other examples could be performing reliable measurements to keep records of public works or reconstructing a mesh suitable for physical phenomena simulation. We believe that the different contributions explained through this document provide tools and insights towards this goal.

We have presented and evaluated an effective strategy for LiDAR point cloud simplification. This solution allows the out-of-core processing of clouds of any size and removes points in regions with high redundancy while preserving those in sparse areas. The sampled points are a subset of the original ones. Hence, we maintain the confidence and reliability determined by the manufacturer of the sensors. This property is essential if measurements between points need explicit guarantees, e.g., for forensic applications. Our strategy also allows taking into account other factors during the simplification, for instance, to preserve more points around features or to favor the removing of noisy points. For this, ideally, reliable normal vectors are needed. These are also key in many tasks, such as surface reconstruction or rendering. We presented a noise-aware normal estimation algorithm specifically designed for point cloud data. Our algorithm adaptively selects the size of the point neighborhood taken into account for PCA

estimation. Therefore, in low-noise regions, smaller radii are selected, allowing the preservation of features. In contrast, larger radii are selected in noisy regions to filter the noise effects. Finally, we have proposed encoding high-resolution point cloud detail information into textures. This way, even if the clouds are simplified, we can later recover this detail, e.g., at render time.

In general, the color data reported by the LiDAR scanners suffers from different artifacts. Much more reliable color can be obtained from medium and high-quality cameras. A dense set of pictures can also be used to estimate a dense reconstruction of a scene. More specifically, a depth map can be estimated for each image, and these can be later fused into a point cloud. Nevertheless, these reconstructions suffer from several artifacts due to challenging content or algorithm shortcomings. We have introduced a pipeline specifically adapted to the reconstruction of buildings. The proposed system's main strength comes from the information derived from the input photographs' semantic segmentation. We use this information in several steps of the pipeline to reduce the typical artifact affecting facade reconstruction. The last step of the pipeline involves re-projecting the input images into the generated meshes using semantic-aware averaging weights. If an available registration exists between a LiDAR model and a photogrammetric model, this technique can also improve the former's color.

Finally, we have also studied how to visualize the massive point clouds obtained from these sources effectively. The key idea is exploiting high-resolution detail encoded into textures with simpler geometry. We proposed one image-based rendering method, which takes a set of depth, color, and normal maps and generates a tessellation that allows for interactive inspection. While high-quality results are obtained, one shortcoming is the limited number of scans visualized at interactive rates. Our second proposed method uses point geometry and texture detail and adaptively renders textured splats. Performance can easily be guaranteed by decreasing the quality during the interactive inspection. Once a specific view has been selected, full-resolution refinement is allowed.

In conclusion, point cloud processing is a fascinating research topic because it allows converting the natural output of scanning technologies into suitable representations. While there is still much room for improvement, we believe our contributions have helped to get closer to our original goal.

7.2 FUTURE WORK

There are multiple exciting lines of work to explore and that we would like to approach as follow-up work.

We have studied how to use additional cost terms to favor preserving samples around regions with high curvature during simplification. However, although our intuition says this should improve the resulting reconstructions' quality, we have not experimentally validated this. It would be interesting to measure this technique's impact in the final reconstruction to understand the ideal preservation ratios.

Our normal estimation approach's main shortcoming is that we do not specifically detect or treat sharp edges. We think we should be able to detect these using covariance analysis on our robust covariance matrix estimations. After an edge is detected, it would be interesting to detect which points lay on either side. Perhaps a RANSAC-based or a Neural-based approach would work best for these cases.

We have also noticed that the scans' IR intensity values are much less noisy than the color information. We want to study if this can be used to modulate the color intensity and remove its noise. In this sense, guided filters are an interesting path to explore. Moreover, our current approach only focuses on a single scan. We should have to produce more coherent color maps across multiple scans by exchanging information between them.

Our improved multi-view stereo pipeline is currently limited to only repairing holes. Currently, we only differentiate facade pixels from the rest, but being able to differentiate between the different elements on a facade (such as balconies, windows, or pillars) could be useful to apply further domain knowledge. Moreover, if relations can be built between these elements, we could potentially be able to exploit symmetry.

Finally, regarding visualization, we want to conduct an in-depth evaluation with other methods to determine the scenarios where our methods excel. Moreover, for the splat-based approach, we want to explore more complex shapes for the splats. By clustering points together, we could extract larger homogeneous surface patches, improving this method's performance.

References

[AAB+18] C. Andújar, O. Argudo, I. Besora, P. Brunet, A. Chica, and M. Comino. "Depth Map Repairing for Building Reconstruction". In: *Spanish Computer Graphics Conference (CEIG)*. The Eurographics Association, 2018.

[AB99] N. Amenta and M. Bern. "Surface reconstruction by Voronoi filtering". In: *Discrete & Computational Geometry* 22.4 (1999), pp. 481–504.

[ABC+01] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. "Point set surfaces". In: *Proceedings Visualization, 2001. VIS 2001.* 2001, pp. 21–29, 537.

[ABC+03] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. "Computing and rendering point set surfaces". In: *IEEE Transactions on Visualization and Computer Graphics* 9.1 (2003), pp. 3–15.

[ACC20] C. Andujar, A. Chica, and M. Comino. "Effective Visualization of Sparse Image-to-Image Correspondences". In: *EuroVis 2020 - Short Papers*. The Eurographics Association, 2020.

[ACT+07] P. Alliez, D. Cohen-Steiner, Y. Tong, and M. Desbrun. "Voronoi-Based Variational Reconstruction of Unoriented Point Sets". In: *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*. SGP '07. Barcelona, Spain: Eurographics Association, 2007, pp. 39–48.

[AFM+06] A. Akbarzadeh, J.-M. Frahm, P. Mordohai, B. Clipp, C. Engels, D. Gallup, P. Merrell, M. Phelps, S. Sinha, B. Talton, et al. "Towards urban 3d reconstruction from video". In: *Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06)*. IEEE. 2006, pp. 1–8.

[AGP+04a] M. Alexa, M. Gross, M. Pauly, H. Pfister, M. Stamminger, and M. Zwicker. "Point-Based Computer Graphics". In: *ACM SIGGRAPH 2004 Course Notes*. SIGGRAPH '04. Los Angeles, CA: Association for Computing Machinery, 2004, 7–es.

[AGP+04b] M. Andersson, J. Giesen, M. Pauly, and B. Speckmann. "Bounds on the K-Neighborhood for Locally Uniformly Sampled Surfaces". In: *Proceedings of the First Eurographics Conference on Point-Based Graphics*. SPBG'04. Switzerland: Eurographics Association, 2004, pp. 167–171.

[AGV+14] M. Agus, E. Gobbetti, A. J. Villanueva, C. Mura, and R. Pajarola. "SOAR: Stochastic optimization for affine global point set registration". In: *Vision, Modeling and Visualization 2014*. The Eurographics Association, 2014.

[AMC08] D. Aiger, N. J. Mitra, and D. Cohen-Or. "4-Points Congruent Sets for Robust Pairwise Surface Registration". In: *ACM SIGGRAPH 2008 Papers*. SIGGRAPH '08. Los Angeles, California: Association for Computing Machinery, 2008.

[AML18] M. Atzmon, H. Maron, and Y. Lipman. "Point Convolutional Neural Networks by Extension Operators". In: *ACM Trans. Graph.* 37.4 (2018).

[APS+14] M. Arikan, R. Preiner, C. Scheiblauer, S. Jeschke, and M. Wimmer. "Large-scale point-cloud visualization through localized textured surface reconstruction". In: *IEEE transactions on Visualization and Computer Graphics* 20.9 (2014), pp. 1280–1292.

[ASS+18] E. Ahmed, A. Saint, A. E. R. Shabayek, K. Cherenkova, R. Das, G. Gusev, D. Aouada, and B. Ottersten. "A survey on deep learning advances on different 3D data representations". In: *arXiv preprint arXiv:1808.01462* (2018).

[BCM18] S. Bianco, G. Ciocca, and D. Marelli. "Evaluating the performance of structure from motion pipelines". In: *Journal of Imaging* 4.8 (2018), p. 98.

[BKC17] V. Badrinarayanan, A. Kendall, and R. Cipolla. "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.12 (2017), pp. 2481–2495.

[BLM+18] G. Bui, T. Le, B. Morago, and Y. Duan. "Point-based rendering enhancement via deep learning". In: *The Visual Computer* 34.6-8 (2018), pp. 829–841.

[BM12] A. Boulch and R. Marlet. "Fast and Robust Normal Estimation for Point Clouds with Sharp Features". In: *Computer Graphics Forum* 31.5 (2012), pp. 1765–1774. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2012.03181.x`.

[BM16] A. Boulch and R. Marlet. "Deep Learning for Robust Normal Estimation in Unstructured Point Clouds". In: *Computer Graphics Forum* 35.5 (2016), pp. 281–290. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12983`.

[BM92] P. J. Besl and N. D. McKay. "Method for registration of 3-D shapes". In: *Sensor Fusion IV: Control Paradigms and Data Structures*. Vol. 1611. International Society for Optics and Photonics. SPIE, 1992, pp. 586–606.

[BRV15] A. Bodis-Szomoru, H. Riemenschneider, and L. Van Gool. "Superpixel Meshes for Fast Edge-Preserving Surface Reconstruction". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[BSB+14] B. Bellekens, V. Spruyt, R. Berkvens, and M. Weyn. "A survey of rigid 3d pointcloud registration algorithms". In: *AMBIENT 2014: the Fourth International Conference on Ambient Computing, Applications, Services and Technologies, August 24-28, 2014, Rome, Italy*. 2014, pp. 8–13.

[BTS+14] M. Berger, A. Tagliasacchi, L. M. Seversky, P. Alliez, J. A. Levine, A. Sharf, and C. T. Silva. "State of the Art in Surface Reconstruction from Point Clouds". In: *Eurographics 2014 - State of the Art Reports*. The Eurographics Association, 2014.

[BTS+17] M. Berger, A. Tagliasacchi, L. M. Seversky, P. Alliez, G. Guennebaud, J. A. Levine, A. Sharf, and C. T. Silva. "A Survey of Surface Reconstruction from Point Clouds". In: *Computer Graphics Forum* 36.1 (2017), pp. 301–329. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12802`.

[CAC+17] M. Comino, C. Andujar, A. Chica, and P. Brunet. "Error-aware construction and rendering of multi-scan panoramas from massive point clouds". In: *Computer Vision and Image Understanding* 157 (2017). Large-Scale 3D Modeling of Urban Indoor or Outdoor Scenes from Images and Range Scans, pp. 43–54.

[CAC+18] M. Comino, C. Andujar, A. Chica, and P. Brunet. "Sensor-aware Normal Estimation for Point Clouds from 3D Range Scans". In: *Computer Graphics Forum* 37.5 (2018), pp. 233–243.

[CB14] S. Calderon and T. Boubekeur. "Point Morphology". In: *ACM Trans. Graph.* 33.4 (2014).

[CCA19] M. Comino Trinidad, A. Chica Calaf, and C. Andújar Gran. "View-dependent Hierarchical Rendering of Massive Point Clouds through Textured Splats". In: *Spanish Computer Graphics Conference (CEIG)*. The Eurographics Association, 2019.

[CCA20] M. Comino, A. Chica, and C. Andujar. "Easy Authoring of Image-Supported Short Stories for 3D Scanned Cultural Heritage". In: *Eurographics Workshop on Graphics and Cultural Heritage*. The Eurographics Association, 2020.

[CCC+08] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. "MeshLab: an Open-Source Mesh Processing Tool". In: *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008.

[CCL+18] L. Cheng, S. Chen, X. Liu, H. Xu, Y. Wu, M. Li, and Y. Chen. "Registration of laser scanning point clouds: A review". In: *Sensors* 18.5 (2018), p. 1641.

[CCS12] M. Corsini, P. Cignoni, and R. Scopigno. "Efficient and Flexible Sampling with Blue Noise Properties of Triangular Meshes". In: *IEEE Transactions on Visualization and Computer Graphics* 18.6 (2012), pp. 914–924.

[CCZ+18] J. Cao, H. Chen, J. Zhang, Y. Li, X. Liu, and C. Zou. "Normal estimation via shifted neighborhood for point cloud". In: *Journal of Computational and Applied Mathematics* 329 (2018), pp. 57–67.

[CGA+13] R. Campos, R. Garcia, P. Alliez, and M. Yvinec. "Splat-based surface reconstruction from defect-laden point sets". In: *Graphical Models* 75.6 (2013), pp. 346–361.

[CGT+19] I. Choi, O. Gallo, A. Troccoli, and J. Kautz. "Extreme View Synthesis". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 7780–7789.

[Che95] S. E. Chen. "QuickTime VR: An Image-Based Approach to Virtual Environment Navigation". In: *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '95. Association for Computing Machinery, 1995, pp. 29–38.

[CHP+07] R. L. Cook, J. Halstead, M. Planck, and D. Ryu. "Stochastic Simplification of Aggregate Detail". In: *ACM Trans. Graph.* 26.3 (2007), pp. 79–87.

[CL96] B. Curless and M. Levoy. "A Volumetric Method for Building Complex Models from Range Images". In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. Association for Computing Machinery, 1996, pp. 303–312.

[CLZ13] E. Castillo, J. Liang, and H. Zhao. "Point Cloud Segmentation and Denoising via Constrained Nonlinear Least Squares Normal Estimates". In: *Innovations for Shape Analysis: Models and Algorithms*. Springer Berlin Heidelberg, 2013, pp. 283–299.

[CMZ+14] D. Ceylan, N. J. Mitra, Y. Zheng, and M. Pauly. "Coupled Structure-from-Motion and 3D Symmetry Detection for Urban Facades". In: *ACM Trans. Graph.* 33.1 (2014).

[Col96] R. T. Collins. "A space-sweep approach to true multi-image matching". In: *Proceedings CVPR IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE. 1996, pp. 358–363.

[COR+16] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 3213–3223.

[CP05] F. Cazals and M. Pouget. "Estimating differential quantities using polynomial fitting of osculating jets". In: *Computer Aided Geometric Design* 22.2 (2005), pp. 121–146.

[CPS+17] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam. "Rethinking atrous convolution for semantic image segmentation". In: *arXiv preprint arXiv:1706.05587* (2017).

[CVH+08] N. D. F. Campbell, G. Vogiatzis, C. Hernández, and R. Cipolla. "Using Multiple Hypotheses to Improve Depth-Maps for Multi-View Stereo". In: *Computer Vision - ECCV 2008*. Springer Berlin Heidelberg, 2008, pp. 766–779.

[DGB+14] M. Di Benedetto, F. Ganovelli, M. Balsa Rodriguez, A. Jaspe Villanueva, R. Scopigno, and E. Gobbetti. "ExploreMaps: Efficient construction and ubiquitous exploration of panoramic view graphs of complex 3D environments". In: *Computer Graphics Forum* 33.2 (2014), pp. 459–468. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12334`.

[DLS05] T. K. Dey, G. Li, and J. Sun. "Normal estimation for point clouds: a comparison study for a Voronoi based method". In: *Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics, 2005.* 2005, pp. 39–46.

[DRD18] S. Discher, R. Richter, and J. Döllner. "A Scalable WebGL-Based Approach for Visualizing Massive 3D Point Clouds Using Semantics-Dependent Rendering Techniques". In: *Proceedings of the 23rd International ACM Conference on 3D Web Technology*. Web3D '18. Poznaundefined, Poland: Association for Computing Machinery, 2018.

[DS06] T. K. Dey and J. Sun. "Normal and Feature Approximations from Noisy Point Clouds". In: *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science*. Springer Berlin Heidelberg, 2006, pp. 21–32.

[DYK07] X. Du, B. Yin, and D. Kong. "Adaptive Out-of-Core Simplification of Large Point Clouds". In: *2007 IEEE International Conference on Multimedia and Expo*. 2007, pp. 1439–1442.

[FG14] S. Fuhrmann and M. Goesele. "Floating Scale Surface Reconstruction". In: *ACM Trans. Graph.* 33.4 (2014).

[FHK+04] A. Frome, D. Huber, R. Kolluri, T. Bülow, and J. Malik. "Recognizing Objects in Range Data Using Regional Point Descriptors". In: *Computer Vision - ECCV 2004*. Springer Berlin Heidelberg, 2004, pp. 224–237.

[FK18] R. Frohlich and Z. Kato. "Simultaneous Multi-view Relative Pose Estimation and 3D Reconstruction from Planar Regions". In: *Asian Conference on Computer Vision*. Springer. 2018, pp. 467–483.

[FNP+16] J. Flynn, I. Neulander, J. Philbin, and N. Snavely. "Deep Stereo: Learning to Predict New Views from the World's Imagery". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 5515–5524.

[FPM+18] J. Femiani, W. R. Para, N. Mitra, and P. Wonka. "Facade segmentation in the wild". In: *arXiv preprint arXiv:1805.08634* (2018).

[GCA13] S. Giraudot, D. Cohen-Steiner, and P. Alliez. "Noise-Adaptive Shape Reconstruction from Raw Point Sets". In: *Computer Graphics Forum* 32.5 (2013), pp. 229–238. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12189`.

[GEM+13] P. Goswami, F. Erol, R. Mukhi, R. Pajarola, and E. Gobbetti. "An efficient multi-resolution framework for high quality interactive rendering of massive point clouds using multi-way kd-trees". In: *The Visual Computer* 29.1 (2013), pp. 69–83.

[GG07] G. Guennebaud and M. Gross. "Algebraic Point Set Surfaces". In: *ACM SIGGRAPH 2007 Papers*. SIGGRAPH '07. San Diego, California: Association for Computing Machinery, 2007, pp. 23–33.

[GKO+18] P. Guerrero, Y. Kleiman, M. Ovsjanikov, and N. J. Mitra. "PCPNet Learning Local Shape Properties from Raw Point Clouds". In: *Computer Graphics Forum* 37.2 (2018), pp. 75–85. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13343`.

[GLY04] D. T. Guinnip, S. Lai, and R. Yang. "View-Dependent Textured Splatting for Rendering Live Scenes". In: *ACM SIGGRAPH 2004 Posters*. SIGGRAPH '04. Los Angeles, California: Association for Computing Machinery, 2004, p. 51.

[GM04] E. Gobbetti and F. Marton. "Layered Point Clouds". In: *Proceedings of the First Eurographics Conference on Point-Based Graphics*. SPBG'04. Switzerland: Eurographics Association, 2004, pp. 113–120.

[Gre10] P. Green-Armytage. "A colour alphabet and the limits of colour coding". In: *JAIC-Journal of the International Colour Association* 5 (2010).

[GTT+03] D. Gledhill, G. Y. Tian, D. Taylor, and D. Clarke. "Panoramic imaging—a review". In: *Computers & Graphics* 27.3 (2003), pp. 435–445.

[GUM01] S. GUMHOLD. "Feature Extraction from Point Clouds". In: *Proc. of 10th International Meshing Roundtable, 2001* (2001), pp. 293–305.

[GWH+19] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun. "Deep Learning for 3D Point Clouds: A Survey". In: *arXiv preprint arXiv:1912.12033* (2019).

[HDD+92] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. "Surface Reconstruction from Unorganized Points". In: *SIGGRAPH Comput. Graph.* 26.2 (1992), pp. 71–78.

[HFW+19] X. Huang, L. Fan, Q. Wu, J. Zhang, and C. Yuan. "Fast registration for cross-source point clouds by using weak regional affinity and pixel-wise refinement". In: *2019 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE. 2019, pp. 1552–1557.

[HJW+17] X.-F. Han, J. S. Jin, M.-J. Wang, W. Jiang, L. Gao, and L. Xiao. "A review of algorithms for filtering the 3D point cloud". In: *Signal Processing: Image Communication* 57 (2017), pp. 103–112.

[HLZ+09] H. Huang, D. Li, H. Zhang, U. Ascher, and D. Cohen-Or. "Consolidation of Unorganized Point Clouds for Surface Reconstruction". In: *ACM Trans. Graph.* 28.5 (2009), pp. 1–7.

[HMF+18] T. Holzmann, M. Maurer, F. Fraundorfer, and H. Bischof. "Semantically Aware Urban 3D Reconstruction with Plane-Based Regularization". In: *The European Conference on Computer Vision (ECCV)*. 2018.

[HRV+18] P. Hermosilla, T. Ritschel, P.-P. Vázquez, À. Vinacua, and T. Ropinski. "Monte Carlo Convolution for Learning on Non-Uniformly Sampled Point Clouds". In: *ACM Trans. Graph.* 37.6 (2018).

[HWG+13] H. Huang, S. Wu, M. Gong, D. Cohen-Or, U. Ascher, and H. (Zhang. "Edge-Aware Point Set Resampling". In: *ACM Trans. Graph.* 32.1 (2013).

[HWS16] T. Hackel, J. D. Wegner, and K. Schindler. "Contour detection in unstructured 3d point clouds". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 1610–1618.

[HZC+13] C. Hane, C. Zach, A. Cohen, R. Angst, and M. Pollefeys. "Joint 3D Scene Reconstruction and Class Segmentation". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2013.

[HZF+17] X. Huang, J. Zhang, L. Fan, Q. Wu, and C. Yuan. "A systematic approach for cross-source point cloud registration by preserving macro and micro structures". In: *IEEE Transactions on Image Processing* 26.7 (2017), pp. 3261–3276.

[HZW+17] X. Huang, J. Zhang, Q. Wu, L. Fan, and C. Yuan. "A coarse-to-fine algorithm for matching and registration in 3D cross-source point clouds". In: *IEEE Transactions on Circuits and Systems for Video Technology* 28.10 (2017), pp. 2965–2977.

[IZZ+17] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. "Image-To-Image Translation With Conditional Adversarial Networks". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[JP11] M. Jancosek and T. Pajdla. "Multi-view reconstruction preserving weakly-supported surfaces". In: *CVPR 2011*. IEEE. 2011, pp. 3121–3128.

[KB04] L. Kobbelt and M. Botsch. "A survey of point-based techniques in computer graphics". In: *Computers & Graphics* 28.6 (2004), pp. 801–814.

[KBH06] M. Kazhdan, M. Bolitho, and H. Hoppe. "Poisson Surface Reconstruction". In: *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*. SGP '06. Cagliari, Sardinia, Italy: Eurographics Association, 2006, pp. 61–70.

[Kel65] K. L. Kelly. "Twenty-two colors of maximum contrast". In: *Color Engineering* 3.26 (1965), pp. 26–27.

[KH13] M. Kazhdan and H. Hoppe. "Screened Poisson Surface Reconstruction". In: *ACM Trans. Graph.* 32.3 (2013).

[KL17] A. Khaloo and D. Lattanzi. "Robust normal estimation and region growing segmentation of infrastructure 3D point cloud models". In: *Advanced Engineering Informatics* 34 (2017), pp. 1–16.

[KPS17] E. Karami, S. Prasad, and M. Shehata. "Image matching using SIFT, SURF, BRIEF and ORB: performance comparison for distorted images". In: *arXiv preprint arXiv:1710.02726* (2017).

[Kru14] J. Kruschke. *Doing Bayesian Data Analysis*. 2nd. Academic Press, 2014.

[KTB07] S. Katz, A. Tal, and R. Basri. "Direct Visibility of Point Sets". In: *ACM SIGGRAPH 2007 Papers*. SIGGRAPH '07. San Diego, California: Association for Computing Machinery, 2007, pp. 24–36.

[KZK17] M. Khoury, Q.-Y. Zhou, and V. Koltun. "Learning Compact Geometric Features". In: *The IEEE International Conference on Computer Vision (ICCV)*. 2017.

[LCL+07] Y. Lipman, D. Cohen-Or, D. Levin, and H. Tal-Ezer. "Parameterization-Free Projection for Geometry Reconstruction". In: *ACM Trans. Graph.* 26.3 (2007), 22–es.

[LCL06] Y. Lipman, D. Cohen-Or, and D. Levin. "Error Bounds and Optimal Neighborhoods for MLS Approximation". In: *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*. SGP '06. Cagliari, Sardinia, Italy: Eurographics Association, 2006, pp. 71–80.

[Leo08] A. Leon-Garcia. *Probability, statistics, and random processes for electrical engineering*. Pearson/Prentice Hall Upper Saddle River, NJ, 2008.

[Lin01] L. Linsen. *Point cloud representation*. Univ., Fak. für Informatik, Bibliothek Technical Report, Faculty of Computer Science, 2001.

[Low04] D. G. Lowe. "Distinctive image features from scale-invariant keypoints". In: *International journal of computer vision* 60.2 (2004), pp. 91–110.

[LSK+10] B. Li, R. Schnabel, R. Klein, Z. Cheng, G. Dang, and S. Jin. "Robust normal estimation for point clouds with sharp features". In: *Computers Graphics* 34.2 (2010), pp. 94–106.

[LSX+15] G. Lu, N. Sebe, C. Xu, and C. Kambhamettu. "Memory efficient large-scale image-based localization". In: *Multimedia Tools and Applications* 74.2 (2015), pp. 479–503.

[LY15] Z. Liying and D. Yong. "A Robust Normal Estimation Algorithm Based on Statistical Distance". In: *2015 Fifth International Conference on Instrumentation and Measurement, Computer, Communication and Control (IMCCC)*. 2015, pp. 1290–1293.

[LYT11] C. Liu, J. Yuen, and A. Torralba. "SIFT Flow: Dense Correspondence across Scenes and Its Applications". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.5 (2011), pp. 978–994.

[LZC+15] X. Liu, J. Zhang, J. Cao, B. Li, and L. Liu. "Quality point cloud normal estimation by guided least squares representation". In: *Computers & Graphics* 51 (2015). International Conference Shape Modeling International, pp. 106–116.

[LZS+11] Y. Li, Q. Zheng, A. Sharf, D. Cohen-Or, B. Chen, and N. J. Mitra. "2D-3D fusion for layer decomposition of urban facades". In: *2011 International Conference on Computer Vision*. IEEE. 2011, pp. 882–889.

[LZZ+17] H. Liu, J. Zhang, J. Zhu, and S. C. H. Hoi. "DeepFacade: A Deep Learning Approach to Facade Parsing". In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. IJCAI'17. Melbourne, Australia: AAAI Press, 2017, pp. 2301–2307.

[MAM14] N. Mellado, D. Aiger, and N. J. Mitra. "Super 4PCS Fast Global Pointcloud Registration via Smart Indexing". In: *Computer Graphics Forum* 33.5 (2014), pp. 205–215. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12446`.

[MD03] C. Moenning and N. A. Dodgson. "A new point cloud simplification algorithm". In: *Proc. Int. Conf. on Visualization, Imaging and Image Processing*. 2003, pp. 1027–1033.

[MGG17] B. Maiseli, Y. Gu, and H. Gao. "Recent developments and trends in point set registration methods". In: *Journal of Visual Communication and Image Representation* 46 (2017), pp. 95–106.

[MK10] B. Mičušík and J. Košecká. "Multi-view superpixel stereo in urban environments". In: *International journal of computer vision* 89.1 (2010), pp. 106–119.

[MKG+19] N. J. Mitra, I. Kokkinos, P. Guerrero, N. Thuerey, V. Kim, and L. Guibas. "CreativeAI: Deep Learning for Graphics". In: *SIGGRAPH 2019 Courses*. Siggraph 2019. Los Angeles, 2019.

[MMB+15] A. Monszpart, N. Mellado, G. J. Brostow, and N. J. Mitra. "RAPter: Rebuilding Man-Made Scenes with Regular Arrangements of Planes". In: *ACM Trans. Graph.* 34.4 (2015).

[MMM12] P. Moulon, P. Monasse, and R. Marlet. "Adaptive structure from motion with a contrario model estimation". In: *Asian Conference on Computer Vision*. Springer. 2012, pp. 257–270.

[MMP+16] P. Moulon, P. Monasse, R. Perrot, and R. Marlet. "Openmvg: Open multiple view geometry". In: *International Workshop on Reproducible Research in Pattern Recognition*. Springer. 2016, pp. 60–74.

[MN03] N. J. Mitra and A. Nguyen. "Estimating Surface Normals in Noisy Point Cloud Data". In: *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*. SCG '03. San Diego, California, USA: Association for Computing Machinery, 2003, pp. 322–328.

[MOG11] Q. Mérigot, M. Ovsjanikov, and L. J. Guibas. "Voronoi-Based Curvature and Feature Estimation from Point Clouds". In: *IEEE Transactions on Visualization and Computer Graphics* 17.6 (2011), pp. 743–756.

[MPF09] Y. Miao, R. Pajarola, and J. Feng. "Curvature-aware adaptive resampling for point-sampled geometry". In: *Computer-Aided Design* 41.6 (2009), pp. 395–403.

[MS15] D. Maturana and S. Scherer. "VoxNet: A 3D Convolutional Neural Network for real-time object recognition". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 922–928.

[MVV+15] O. Martinez-Rubi, S. Verhoeven, M. Van Meersbergen, P. Van Oosterom, R. GonÁalves, T. Tijssen, et al. "Taming the beast: Free and open-source massive point cloud web visualization". In: *Capturing Reality Forum 2015, 23-25 November 2015, Salzburg, Austria*. The Servey Association. 2015.

[MWA+13] P. Musialski, P. Wonka, D. G. Aliaga, M. Wimmer, L. van Gool, and W. Purgathofer. "A Survey of Urban Reconstruction". In: *Computer Graphics Forum* 32.6 (2013), pp. 146–177. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12077.

[NBW14] A. Nurunnabi, D. Belton, and G. West. "Robust statistical approaches for local planar surface fitting in 3D laser scanning data". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 96 (2014), pp. 106–122.

[NLH+19] Q. Niu, M. Li, S. He, C. Gao, S. .-.-H. Gary Chan, and X. Luo. "Resource-Efficient and Automated Image-Based Indoor Localization". In: *ACM Trans. Sen. Netw.* 15.2 (2019).

[NPD+11] B. Neubert, S. Pirk, O. Deussen, and C. Dachsbacher. "Improved Model- and View-Dependent Pruning of Large Botanical Scenes". In: *Computer Graphics Forum* 30.6 (2011), pp. 1708–1718. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2011.01897.x.

[NW17] L. Nan and P. Wonka. "PolyFit: Polygonal Surface Reconstruction From Point Clouds". In: *The IEEE International Conference on Computer Vision (ICCV)*. 2017.

[NWB15] A. Nurunnabi, G. West, and D. Belton. "Outlier detection and robust normal-curvature estimation in mobile laser scanning 3D point cloud data". In: *Pattern Recognition* 48.4 (2015), pp. 1404–1419.

[OKY15] F. Okura, M. Kanbara, and N. Yokoya. "Mixed-Reality World Exploration Using Image-Based Rendering". In: *J. Comput. Cult. Herit.* 8.2 (2015).

[PGK02] M. Pauly, M. Gross, and L. Kobbelt. "Efficient simplification of point-sampled surfaces". In: *IEEE Visualization, 2002. VIS 2002.* 2002, pp. 163–170.

[PJW12] R. Preiner, S. Jeschke, and M. Wimmer. "Auto Splats: Dynamic Point Cloud Visualization on the GPU". English. In: *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization.* 2012, pp. 139–148.

[PKG03] M. Pauly, R. Keiser, and M. Gross. "Multi-scale Feature Extraction on Point-Sampled Surfaces". In: *Computer Graphics Forum* 22.3 (2003), pp. 281–289. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-8659.00675.

[PKK+03] M. Pauly, R. Keiser, L. P. Kobbelt, and M. Gross. "Shape Modeling with Point-Sampled Geometry". In: *ACM Trans. Graph.* 22.3 (2003), pp. 641–650.

[PMW+08] M. Pauly, N. J. Mitra, J. Wallner, H. Pottmann, and L. J. Guibas. "Discovering Structural Regularity in 3D Geometry". In: *ACM SIGGRAPH 2008 Papers*. SIGGRAPH '08. Los Angeles, California: Association for Computing Machinery, 2008.

[PNF+08] M. Pollefeys, D. Nistér, J.-M. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S.-J. Kim, P. Merrell, et al. "Detailed real-time urban 3d reconstruction from video". In: *International Journal of Computer Vision* 78.2-3 (2008), pp. 143–167.

[PPM+09] M. Przemyslaw, W. Peter, R. Meinrad, M. Stefan, and P. Werner. "Symmetry-Based Façade Repair". In: *14th International Workshop on Vision, Modeling, and Visualization, VMV 2009, November 16-18, 2009, Braunschweig, Germany*. DNB, 2009, pp. 3–10.

[PVV+04] M. Pollefeys, L. Van Gool, M. Vergauwen, F. Verbiest, K. Cornelis, J. Tops, and R. Koch. "Visual modeling with a hand-held camera". In: *International Journal of Computer Vision* 59.3 (2004), pp. 207–232.

[PZ17] E. Penner and L. Zhang. "Soft 3D Reconstruction for View Synthesis". In: *ACM Trans. Graph.* 36.6 (2017).

[QHG19] J. Qi, W. Hu, and Z. Guo. "Feature Preserving and Uniformity-Controllable Point Cloud Simplification on Graph". In: *2019 IEEE International Conference on Multimedia and Expo (ICME)*. 2019, pp. 284–289.

[QSM+17] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[QYS+17] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 5099–5108.

[RDD15] R. Richter, S. Discher, and J. Döllner. "Out-of-core visualization of classified 3d point clouds". In: *3D Geoinformation Science*. Springer, 2015, pp. 227–242.

[RL00] S. Rusinkiewicz and M. Levoy. "QSplat: A Multiresolution Point Rendering System for Large Meshes". In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '00. ACM Press/Addison-Wesley Publishing Co., 2000, pp. 343–352.

[RL01] S. Rusinkiewicz and M. Levoy. "Efficient variants of the ICP algorithm". In: *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*. IEEE. 2001, pp. 145–152.

[RL08] P. Rosenthal and L. Linsen. "Image-space Point Cloud Rendering". In: *Proceedings of Computer Graphics International*. 2008, pp. 136–143. published.

[SA07] P. Shanmugam and O. Arikan. "Hardware Accelerated Ambient Occlusion Techniques on GPUs". In: *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*. I3D '07. Seattle, Washington: Association for Computing Machinery, 2007, pp. 73–80.

[SA85] S. Suzuki and K. Abe. "Topological structural analysis of digitized binary images by border following". In: *Computer Vision, Graphics, and Image Processing* 30.1 (1985), pp. 32–46.

[SCD+06] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. "A comparison and evaluation of multi-view stereo reconstruction algorithms". In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 1. IEEE. 2006, pp. 519–528.

[Sch16] M. Schütz. "Potree: Rendering Large Point Clouds in Web Browsers". MA thesis. Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2016.

[SDC+20] J. Sanchez, F. Denis, D. Coeurjolly, F. Dupont, L. Trassoudaine, and P. Checchin. "Robust normal vector estimation in 3D point clouds through iterative principal component analysis". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 163 (2020), pp. 18–35.

[SF09] H. Song and H.-Y. Feng. "A progressive point cloud simplification algorithm with preserved sharp edge data". In: *The International Journal of Advanced Manufacturing Technology* 45.5-6 (2009), pp. 583–592.

[SF16] J. L. Schönberger and J.-M. Frahm. "Structure-from-Motion Revisited". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[SHD11] T. Schlömer, D. Heck, and O. Deussen. "Farthest-point optimized point sets with maximized minimum distance". In: *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*. 2011, pp. 135–142.

[SK00] H. Shum and S. B. Kang. "Review of image-based rendering techniques". In: *Visual Communications and Image Processing 2000*. Vol. 4067. International Society for Optics and Photonics. SPIE, 2000, pp. 2–13.

[SK17] M. Simonovsky and N. Komodakis. "Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[SM01] J. Snyder and D. Mitchell. "Sampling-efficient mapping of spherical images". In: *Polar* 19 (2001), pp. 1–29.

[SM16] M. Schmitz and H. Mayer. "A Convolutional Network for Semantic Facade Segmentation and Interpretation". In: *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences* 41 (2016).

[SMK+15] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. "Multi-View Convolutional Neural Networks for 3D Shape Recognition". In: *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*. ICCV '15. IEEE Computer Society, 2015, pp. 945–953.

[SMO+20] M. Schütz, G. Mandlburger, J. Otepka, and M. Wimmer. "Progressive Real-Time Rendering of One Billion Points Without Hierarchical Acceleration Structures". In: *Computer Graphics Forum* 39.2 (2020), pp. 51–64.

[SPT15] C. G. Serna, R. Pillay, and A. Trémeau. "Data fusion of objects using techniques such as Laser Scanning, Structured Light and Photogrammetry for Cultural Heritage Applications". In: *International Workshop on Computational Color Imaging*. Springer. 2015, pp. 208–224.

[SSL+13] D. Sibbing, T. Sattler, B. Leibe, and L. Kobbelt. "SIFT-Realistic Rendering". In: *2013 International Conference on 3D Vision - 3DV 2013*. 2013, pp. 56–63.

[STM+19] Y. Singh, R. Toldo, L. MAGRI, S. Fantoni, and A. Fusiello. *Method for 3D modelling based on structure from motion processing of sparse 2D images*. US Patent App. 10/198,858. 2019.

[SYZ+17] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser. "Semantic Scene Completion From a Single Depth Image". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[SZP+16] J. L. Schönberger, E. Zheng, M. Pollefeys, and J.-M. Frahm. "Pixelwise View Selection for Unstructured Multi-View Stereo". In: *European Conference on Computer Vision (ECCV)*. 2016.

[TCL+13] G. K. Tam, Z.-Q. Cheng, Y.-K. Lai, F. C. Langbein, Y. Liu, D. Marshall, R. R. Martin, X.-F. Sun, and P. L. Rosin. "Registration of 3D point clouds and meshes: A survey from rigid to nonrigid". In: *IEEE transactions on visualization and computer graphics* 19.7 (2013), pp. 1199–1217.

[TF09] V. Todorov and P. Filzmoser. "An Object-Oriented Framework for Robust Multivariate Analysis". In: *Journal of Statistical Software* 32.3 (2009), pp. 1–47.

[TGF+15] R. Toldo, R. Gherardi, M. Farenzena, and A. Fusiello. "Hierarchical structure-and-motion recovery from uncalibrated images". In: *Computer Vision and Image Understanding* 140 (2015), pp. 127–143.

[TPK+18] M. Tatarchenko, J. Park, V. Koltun, and Q.-Y. Zhou. "Tangent Convolutions for Dense Prediction in 3D". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[TSK+10] O. Teboul, L. Simon, P. Koutsourakis, and N. Paragios. "Segmentation of building facades using procedural shape priors". In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE. 2010, pp. 3105–3112.

[WAC+11] C. Wu, S. Agarwal, B. Curless, and S. M. Seitz. "Multicore bundle adjustment". In: *CVPR 2011*. IEEE. 2011, pp. 3057–3064.

[Wal15] G. Walsh. *Leica ScanStation P-Series. Details that matter*. White paper, Leica Geosystems AG. 2015.

[WBB+07] M. Wand, A. Berner, M. Bokeloh, A. Fleck, M. Hoffmann, P. Jenke, B. Maier, D. Staneker, and A. Schilling. "Interactive Editing of Large Point Clouds". In: *Eurographics Symposium on Point-Based Graphics*. The Eurographics Association, 2007.

[Wil78] L. Williams. "Casting Curved Shadows on Curved Surfaces". In: *SIGGRAPH Comput. Graph.* 12.3 (1978), pp. 270–274.

[WK04] J. Wu and L. Kobbelt. "Optimized Sub-Sampling of Point Sets for Surface Splatting". In: *Computer Graphics Forum* 23.3 (2004), pp. 643–652.

[WS06] M. Wimmer and C. Scheiblauer. "Instant Points: Fast Rendering of Unprocessed Point Clouds". In: *Proceedings of the 3rd Eurographics / IEEE VGTC Conference on Point-Based Graphics*. SPBG'06. Boston, Massachusetts: Eurographics Association, 2006, pp. 129–137.

[WSK+15] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. "3D ShapeNets: A Deep Representation for Volumetric Shapes". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[Wu13] C. Wu. "Towards Linear-Time Incremental Structure from Motion". In: *2013 International Conference on 3D Vision - 3DV 2013*. 2013, pp. 127–134.

[YGJ+14] D.-M. Yan, J. Guo, X. Jia, X. Zhang, and P. Wonka. "Blue-Noise Remeshing with Farthest Point Optimization". In: *Computer Graphics Forum* 33.5 (2014), pp. 167–176.

[YGW+15] D.-M. Yan, J.-W. Guo, B. Wang, X.-P. Zhang, and P. Wonka. "A survey of blue-noise sampling and its applications". In: *Journal of Computer Science and Technology* 30.3 (2015), pp. 439–452.

[YGW06] R. Yang, D. Guinnip, and L. Wang. "View-dependent textured splatting". In: *The Visual Computer* 22.7 (2006), pp. 456–467.

[Yuk15] C. Yuksel. "Sample Elimination for Generating Poisson Disk Sample Sets". In: *Computer Graphics Forum* 34.2 (2015), pp. 25–32.

[ZCL+13] J. Zhang, J. Cao, X. Liu, J. Wang, J. Liu, and X. Shi. "Point cloud normal estimation via low-rank subspace clustering". In: *Computers & Graphics* 37.6 (2013). Shape Modeling International (SMI) Conference 2013, pp. 697–706.

[ZPB+01] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. "Surface Splatting". In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. Association for Computing Machinery, 2001, pp. 371–378.

[ZPK+02] M. Zwicker, M. Pauly, O. Knoll, and M. Gross. "Pointshop 3D: An Interactive System for Point-Based Surface Editing". In: *ACM Trans. Graph.* 21.3 (2002), pp. 322–329.

[ZPL+19] R. Zhao, M. Pang, C. Liu, and Y. Zhang. "Robust Normal Estimation for 3D LiDAR Point Clouds in Urban Environments". In: *Sensors* 19.5 (2019), p. 1248.

[ZSW+10] Q. Zheng, A. Sharf, G. Wan, Y. Li, N. J. Mitra, D. Cohen-Or, and B. Chen. "Non-Local Scan Consolidation for 3D Urban Scenes". In: *ACM Trans. Graph.* 29.4 (2010).

[ZTF+18] T. Zhou, R. Tucker, J. Flynn, G. Fyffe, and N. Snavely. "Stereo Magnification: Learning View Synthesis Using Multiplane Images". In: *ACM Trans. Graph.* 37.4 (2018).

[ZYT18] L. Zheng, Y. Yang, and Q. Tian. "SIFT Meets CNN: A Decade Survey of Instance Retrieval". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.5 (2018), pp. 1224–1244.

Websites

[1] *AgiSoft Metashape Professional Edition (Version 1.5.5) (Software). (2019*).* https://www.agisoft.com/downloads/installer.

[2] *Autodesk Recap Photo 2020.* https://www.autodesk.com/products/recap/overview.

[3] *Google Earth.* https://earth.google.com/web/.

[4] *Institut Cartogràfic i Geològic de Catalunya.* https://www.icgc.cat/.

[5] *Institut Cartogràfic i Geològic de Catalunya, Barcelona Model.* https://betaportal.icgc.cat/wordpress/sagrada_familia_eixample_3d/.

[6] *Institut Cartogràfic i Geològic de Catalunya, Girona Model.* https://betaportal.icgc.cat/wordpress/model-3d-de-girona/.

[7] *Institut Cartogràfic i Geològic de Catalunya, LiDAR Data.* https://www.icgc.cat/Descarregues/Elevacions/Dades-lidar.

[8] *Leica P20.* https://w3.leica-geosystems.com/downloads123/hds/hds/scanstation_p20/brochures/leica_scanstation_p20_bro_es.pdf.

[9] *Leica RTC 360.* https://leica-geosystems.com/en-us/products/laser-scanners/scanners/leica-rtc360.

[10] *New York University Spatial Data Repository.* https://geo.nyu.edu/.

[11] *Open Heritage.* https://www.openheritage3d.org/.

[12] *Robotic 3D Scan Repository, Universität Osnabrück.* http://kos.informatik.uni-osnabrueck.de/3Dscans/.

[13] P. Alliez, S. Giraudot, C. Jamin, F. Lafarge, Q. Mérigot, J. Meyron, L. Saboret, N. Salman, and S. Wu. "Point Set Processing". In: *CGAL User and Reference Manual*. 5.0.2. CGAL Editorial Board, 2020.

[14] T. C. Project. *CGAL User and Reference Manual*. 5.0.2. CGAL Editorial Board, 2020.

[15] T. Schöps, J. L. Schönberger, S. Galliani, T. Sattler, K. Schindler, M. Pollefeys, and A. Geiger. "A Multi-View Stereo Benchmark with High-Resolution Images and Multi-Camera Videos". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[16] R. Tyleček and R. Šára. "Spatial Pattern Templates for Recognition of Objects with Regular Structure". In: *Proc. GCPR*. 2013.

All the referenced websites have been correctly accessed on November 20th, 2020.