HASSO-PLATTNER-INSTITUT
Fachgebiet Computergrafische Systeme

Concepts and Techniques for Processing and Rendering of Massive 3D Point Clouds

Dissertation
zur Erlangung des akademischen Grades
"doctor rerum naturalium"
(Dr. rer. nat.)
in der Wissenschaftsdisziplin Informatik

eingereicht an der
Digital Engineering Fakultät
der Universität Potsdam

Rico Richter, M.Sc.

Potsdam
23. März 2018

Published online at the
Institutional Repository of the University of Potsdam:
https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-423304
https://doi.org/10.25932/publishup-42330

Abstract

Remote sensing technology, such as airborne, mobile, or terrestrial laser scanning, and photogrammetric techniques, are fundamental approaches for efficient, automatic creation of digital representations of spatial environments. For example, they allow us to generate 3D point clouds of landscapes, cities, infrastructure networks, and sites. As essential and universal category of geodata, 3D point clouds are used and processed by a growing number of applications, services, and systems such as in the domains of urban planning, landscape architecture, environmental monitoring, disaster management, virtual geographic environments as well as for spatial analysis and simulation.

While the acquisition processes for 3D point clouds become more and more reliable and widely-used, applications and systems are faced with more and more 3D point cloud data. In addition, 3D point clouds, by their very nature, are raw data, i.e., they do not contain any structural or semantics information. Many processing strategies common to GIS such as deriving polygon-based 3D models generally do not scale for billions of points. GIS typically reduce data density and precision of 3D point clouds to cope with the sheer amount of data, but that results in a significant loss of valuable information at the same time.

This thesis proposes concepts and techniques designed to efficiently store and process massive 3D point clouds. To this end, *object-class segmentation* approaches are presented to attribute semantics to 3D point clouds, used, for example, to identify building, vegetation, and ground structures and, thus, to enable processing, analyzing, and visualizing 3D point clouds in a more effective and efficient way. Similarly, *change detection and updating strategies* for 3D point clouds are introduced that allow for reducing storage requirements and incrementally updating 3D point cloud databases. In addition, this thesis presents *out-of-core, real-time rendering techniques* used to interactively explore 3D point clouds and related analysis results. All techniques have been implemented based on specialized spatial data structures, out-of-core algorithms, and GPU-based processing schemas to cope with massive 3D point clouds having billions of points.

All proposed techniques have been evaluated and demonstrated their applicability to the field of geospatial applications and systems, in particular for tasks such as classification, processing, and visualization. Case studies for 3D point clouds of entire cities with up to 80 billion points show that the presented approaches open up new ways to manage and apply large-scale, dense, and time-variant 3D point clouds as required by a rapidly growing number of applications and systems.

Zusammenfassung

Fernerkundungstechnologien wie luftgestütztes, mobiles oder terrestrisches Laserscanning und photogrammetrische Techniken sind grundlegende Ansätze für die effiziente, automatische Erstellung von digitalen Repräsentationen räumlicher Umgebungen. Sie ermöglichen uns zum Beispiel die Erzeugung von 3D-Punktwolken für Landschaften, Städte, Infrastrukturnetze und Standorte. 3D-Punktwolken werden als wesentliche und universelle Kategorie von Geodaten von einer wachsenden Anzahl an Anwendungen, Diensten und Systemen genutzt und verarbeitet, zum Beispiel in den Bereichen Stadtplanung, Landschaftsarchitektur, Umweltüberwachung, Katastrophenmanagement, virtuelle geographische Umgebungen sowie zur räumlichen Analyse und Simulation.

Da die Erfassungsprozesse für 3D-Punktwolken immer zuverlässiger und verbreiteter werden, sehen sich Anwendungen und Systeme mit immer größeren 3D-Punktwolken-Daten konfrontiert. Darüber hinaus enthalten 3D-Punktwolken als Rohdaten von ihrer Art her keine strukturellen oder semantischen Informationen. Viele GIS-übliche Verarbeitungsstrategien, wie die Ableitung polygonaler 3D-Modelle, skalieren in der Regel nicht für Milliarden von Punkten. GIS reduzieren typischerweise die Datendichte und Genauigkeit von 3D-Punktwolken, um mit der immensen Datenmenge umgehen zu können, was aber zugleich zu einem signifikanten Verlust wertvoller Informationen führt.

Diese Arbeit präsentiert Konzepte und Techniken, die entwickelt wurden, um massive 3D-Punktwolken effizient zu speichern und zu verarbeiten. Hierzu werden Ansätze für die *Objektklassen-Segmentierung* vorgestellt, um 3D-Punktwolken mit Semantik anzureichern; so lassen sich beispielsweise Gebäude-, Vegetations- und Bodenstrukturen identifizieren, wodurch die Verarbeitung, Analyse und Visualisierung von 3D-Punktwolken effektiver und effizienter durchführbar werden. Ebenso werden *Änderungserkennungs- und Aktualisierungsstrategien* für 3D-Punktwolken vorgestellt, mit denen Speicheranforderungen reduziert und Datenbanken für 3D-Punktwolken inkrementell aktualisiert werden können. Des Weiteren beschreibt diese Arbeit *Out-of-Core Echtzeit-Rendering-Techniken* zur interaktiven Exploration von 3D-Punktwolken und zugehöriger Analyseergebnisse. Alle Techniken wurden mit Hilfe spezialisierter räumlicher Datenstrukturen, Out-of-Core-Algorithmen und GPU-basierter Verarbeitungsschemata implementiert, um massiven 3D-Punktwolken mit Milliarden von Punkten gerecht werden zu können.

Alle vorgestellten Techniken wurden evaluiert und die Anwendbarkeit für Anwendungen und Systeme, die mit raumbezogenen Daten arbeiten, wurde insbesondere für Aufgaben wie Klassifizierung, Verarbeitung und Visualisierung demonstriert. Fallstudien für 3D-Punktwolken von ganzen Städten mit bis zu 80 Milliarden Punkten zeigen, dass die vorgestellten Ansätze neue Wege zur Verwaltung und Verwendung von großflächigen, dichten und zeitvarianten 3D-Punktwolken eröffnen, die von einer wachsenden Anzahl an Anwendungen und Systemen benötigt werden.

Acknowledgments

This dissertation is the result of my research work at the Department of Computer Graphics Systems at the Hasso-Plattner-Institut (University of Potsdam). I am very grateful to my adviser Prof. Dr. Jürgen Döllner for granting me this opportunity.

I owe sincere and earnest thankfulness to all the anonymous reviewers for their suggestions to improve my work. It is a great pleasure to thank everyone who supported me during writing this dissertation, who proofreaded my thesis and gave comments and ideas for improvement. This work has been partially funded by the Hasso-Plattner-Institut, the German Federal Ministry of Education and Research (BMBF) as part of the InnoProfile research group "3D Geoinformation" (www.3dgi.de) and "4D-nD GeoVis" (www.4dndvis.de).

Potsdam, Germany, March 23, 2018 *Rico Richter*

Contents

Chapter 1

Introduction

"Based on their fundamental simplicity, points have motivated a variety of research on topics such as shape modeling, object capturing, simplification, processing, rendering, and hybrid point-polygon methods."

— Sainz, Pajarola & Lario 2004

This chapter introduces basic terms and presents research objectives and challenges of this thesis. It outlines its main contributions for processing, analyzing, managing, and visualizing 3D point clouds.

1.1 Background and Motivation

3D point clouds allow us to capture any type of 3D object, 3D environment and 3D spatial phenomenon (Leberl *et al.* 2010; Lafarge & Mallet 2012). To generate 3D point clouds for our physical surroundings, active and passive remote sensing approaches are frequently used and implemented by a number of reliable techniques. Remote sensing equipment can be mounted on a variety of platforms to capture data at different scales. 3D point clouds can also be synthesized as computational results by geometry processes of simulations.

1.1.1 Remote Sensing

"*Remote sensing is the science of deriving information about an object from measurements made at a distance from the object, i.e., without actually coming in contact with it*" (Campbell & Wynne 2011). **Active remote sensing techniques**, such as LiDAR or radar, emit a signal supplied by their own energy source, which is reflected by the target and captured by the sensor (Barrett & Curtis 1993). **Passive remote sensing techniques** operate in the visible, infrared, thermal infrared, and microwave portions of the electromagnetic spectrum and gather radiation that is emitted or reflected by the object or surrounding areas (NASA 2017). Most popular passive sensors are camera systems to capture photographs and videos as well as radiometers and spectrometers.

1.1.2 Remote Sensing Platforms

Remote sensing hardware is mounted on structures or vehicles to capture data. Main platform categories are **spaceborne, airborne, and ground based**. Data from spaceborne platforms, such as earth observation satellites, is beyond the scope of this thesis because the resulting data is not used to derive precise 3D point clouds.

Airborne Acquisition of 3D Point Clouds

Data acquisition from the air, i.e., airborne, is typically performed by airplanes, helicopters, and unmanned aerial vehicles (UAVs). Airplanes are typically used at high altitudes to capture large areas such as cities and landscapes. The main operational area of helicopters are mid altitudes to capture infrastructure networks such as powerlines, rails, and roads. UAVs are used at low altitudes to capture small sites and areas such as facilities. All platforms can be equipped with LiDAR and image-based sensors.

Airborne laser scanning, i.e., Light Detection and Ranging (LiDAR) supports the derivation of dense 3D point clouds and digital surface models from a *bird's eye view* (Leberl *et al.* 2010; Haala & Rothermel 2012b) for cities and landscapes (Figure 1.1 (a)). Filin & Pfeifer (2006) stated: "*airborne laser ranging as a leading technology for the extraction of information about physical surfaces*". These 3D point clouds are typically used to derive 3D geometric models, for example, city models, digital surface models (DSMs), digital terrain models (DTMs), as well as for monitoring of changes such as for buildings, vegetation, and terrain.

In contrast to LiDAR, **dense image matching** follows a different strategy: Rothermel & Haala (2011) stated "*the increasing quality of digital airborne cameras in combination with recent improvements in matching algorithms meanwhile allow for the automatic image based collection as a suitable alternative*". Photogrammetric methods such as semi-global matching (Gehrke *et al.* 2010) and computer vision algorithms use large sets of overlapping and high resolution aerial images to derive dense 3D point clouds (Haala 2013; Koutsoudis *et al.* 2013; Remondino *et al.* 2014). These 3D point clouds have typically a higher density than LiDAR data but exhibit several restrictions, for example, they can hardly cover the ground surface below vegetation (Figure 1.1 (b)).

Ground-based Acquisition of Point Clouds

The ground-based data acquisition can be categorized into mobile and static methods.

Mobile mapping systems are commonly used to capture infrastructure networks such as road (Figure 1.1 (c)) and railroad networks (Figure 1.1 (d)) with mobile vehicles such as cars or trains (Blug *et al.* 2007; Haala *et al.* 2008; Puente *et al.* 2013). As Nebiker, Bleisch & Christen (2010) explain "*The latest generation of mobile mapping systems is equipped with multiple laser scanners and is capable of dynamically acquiring vast amounts of 3D point clouds with a very high point density at the centimeter level and a point position accuracy of approximately 5 cm.*"

Terrestrial laser scanning is used to capture the physical environment from a *pedestrian's view* (e.g., building façades and streets (Nebiker, Bleisch & Christen 2010; Kang & Lu 2011)). Terrestrial laser scanners are typically mounted on a tripod and used to capture individual objects (e.g., buildings, constructions), structures and sites (e.g., cultural heritage) (Girardeau-Montaut *et al.* 2005), applied, for example, for documentation purposes (e.g., condition of facilities, crime scenes) of the exterior (Figure 1.1 (e)) and interior (Figure 1.1 (g)).

Recently, **dense image matching** based on digital photography allows even non-experts to generate 3D point clouds of objects and structures based on specialized apps and hand-held devices (e.g., using Autodesk's 123D Catch (Autodesk 2017) or Microsoft's Photosynth (Microsoft 2015)), i.e., this approach does not require specialized hardware

(Snavely *et al.* 2008). Consequently, devices, such as digital cameras and smartphones, can be effectively used to create 3D point clouds for indoor environments (Figure 1.1 (h)) as well as outdoor scenes (Figure 1.1 (f)). Kersten & Lindstaedt (2012) give an overview of automatic 3D object reconstruction with open-source systems for architectural, cultural heritage and archaeological applications. Wenzel *et al.* (2012) state that "*off-the-shelf industry cameras [...] provide high spatial resolution with low radiometric noise, which enables a one-shot solution and thus an efficient data acquisition while satisfying high accuracy requirements*".

1.1.3 Point Clouds as Universal Category of Geodata

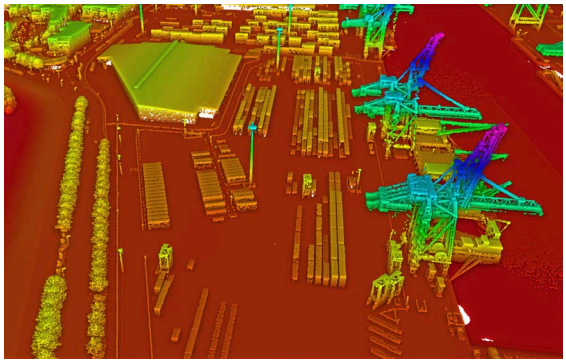
3D point clouds form "digital twins" of 3D objects, 3D environments and 3D spatial phenomena. They describe counterparts of the reality by means of finite sets of 3D points, which do not contain or imply any structure or order. 3D point clouds, the most elementary geometric primitives, can represent any spatial entity in a discrete, approximative way, which is always faced with incompleteness and ambiguousness. They can represent almost any type of physical object, site, landscape, geographic region, or infrastructure – at all scales and with any precision. Becoming an essential and universal category of geodata, 3D point clouds are used and processed by a rapidly growing number of applications, services, and systems such as in the domains of land surveying, urban planning, landscape architecture, environmental monitoring, disaster management, virtual geographic environments, as well as spatial analysis and simulation.

From an IT systems engineering perspective, the use of remote sensing devices or, alternatively, low-cost devices using dense image matching supports cost-efficient, area-wide, and semi-automatic data acquisition for our physical surroundings. The resulting 3D point clouds are commonly used to derive polygonal 3D models (Beutel, Mølhave & Agarwal 2010; Arikan *et al.* 2013) such as for sites, buildings, terrain, vegetation, objects, and indoor environments. However, the potential of 3D point clouds is not restricted to the process of creating polygonal 3D models. They also provide a direct data basis for documentation, analysis, examination, monitoring, and visualization tasks (Virtanen *et al.* 2017). This becomes important when the data amount increases. To process, analyze, inspect, and visualize large datasets such as massive 3D point clouds, out-of-core or external memory algorithms are required (Livny, Kogan & El-Sana 2009; Nebiker, Bleisch & Christen 2010; Ganovelli & Scopigno 2012; Rodríguez & Gobbetti 2013). Figure 1.1 shows 3D point clouds generated with LiDAR ((a), (c), (d), (e), (g)) and image-based ((b), (f), (h)) data acquisition for aerial ((a), (b)), mobile mapping ((c), (d)) and ground-based ((e), (f), (g), (h)) application.

1.2 Problem Statement, Challenges and Research Objectives

In the last decade, the availability, accuracy, density, and massivity of 3D point clouds has vastly increased. Reasons include:

- **Improved hardware**, i.e., LiDAR systems that can generate millions of points per second.
- **Dense image matching**, which facilitates a cost efficiently data acquisition for large areas in contrast to LiDAR (Gehrke *et al.* 2010).



(a) *Aerial LiDAR data of Rotterdam.*

(b) *3D point cloud from dense image matching of Berlin.*

(c) *Mobile mapping point cloud captured with a LiDAR scanner mounted on a vehicle.*

(d) *LiDAR point cloud captured by a train.*

(e) *Terrestrial laser scan of a building.*

(f) *Dense image matching point cloud generated with Autodesk's 123D Catch (Autodesk 2017).*

(g) *Indoor scan of a building captured with a terrestrial LiDAR scanner.*

(h) *Indoor scan of a building captured with a Microsoft Kinect.*

Figure 1.1: *3D point clouds generated with LiDAR ((a), (c), (d), (e), (g)) and image-based ((b), (f), (h)) acquisition techniques for aerial ((a), (b)), mobile mapping ((c), (d)) and terrestrial ((e), (f), (g), (h)) application.*

- **Digitalization of workflows based on 3D spatial models**, i.e., regular scans of cities, landscapes, and infrastructure.
- **Novel carrier systems and platforms** such as cars and unmanned aerial vehicles (UAVs).
- **Cheap and easy-to-use hardware and software solutions**, which increase the number of industries that can apply 3D point clouds and set up new applications and workflows.

1.2.1 Geographic Information Systems and 3D Point Clouds

Geographic information systems (GIS) are designed to integrate, store, and manage geodata and provide functions to analyze, process, edit, and present the data. Most common data categories are vector data (e.g., shapes), raster data (e.g., aerial images), and 2.5D data (e.g., DTMs). However, more and more applications and GIS users have the key technical requirement to store, manage, and process massive and dense 3D point clouds because they need a detailed representation of the captured site. GIS software is sometimes limited due to their processing strategies that generally do not scale. Reasons are compatibility restrictions because outdated hardware platforms and numerous target systems must be supported. ESRI, market leader for GIS software, gives the following hardware requirements for ArcGIS: a minimum of 64 MB GPU memory and OpenGL Version 2.0 (ESRI 2017). The engine for data handling and provision for data handling evolved over decades and was probably not designed to implement out-of-core concepts for 3D data such as 3D point clouds. As a remedy GIS frequently have to reduce the precision and density of 3D point clouds to handle the data. ArcGIS, for example, gives the following recommendation for loading 3D point clouds: "*A recommended file size is approximately 25 to 50 MB, and no larger than 100 MB. The LAS files should not contain more than three million points per file when used in a LAS dataset.*" (ESRI 2016). This statement shows impressively the gap between the amount of data that can be handled practically with today's GIS systems (i.e., a few million points) and the amount of data that needs to be handled (i.e., billions of points).

The need to efficiently handle massive 3D point clouds grows because many applications demand for frequent scans and simultaneous use of scans taken at different points in time. For example, in the context of urban planning, the continuation workflows applied to virtual 3D city models requires to identify differences between scans from different points in time. Hence, novel concepts and techniques for an efficient storage, management, and processing of the data are required.

1.2.2 Research Objectives

This thesis aims at the following research objectives:

- **Efficient processing and storage of massive 3D point clouds.** The ability to cope with massive 3D point clouds containing billions of points is a challenge for systems and applications because the data exceeds available memory capacities and processing resources. Existing solutions typically thin out or rasterize the 3D point cloud to reduce the data size (Van Gosliga, Lindenbergh & Pfeifer 2006), i.e., they cannot take full advantage of the available data. This, however, leads to a

loss of accuracy and makes it difficult to match the original unfiltered data with processing results. To overcome this limitation, specialized spatial data structures and out-of-core algorithms are developed to prepare and handle massive 3D point clouds for storage, analysis, and visualization.

- **Redundancy elimination for 3D point clouds.** A continuous data acquisition typically results in redundant, time varying 3D point clouds for large parts of the captured area. One challenge is to compare 3D point clouds from different points in time to identify differences with "reasonable" processing times. Highly parallel, GPU-based processing schemas are developed to perform fast change detection.

- **3D point cloud classification.** In general, 3D point clouds have no inherent semantics. Applications, however, commonly need only subsets of 3D point clouds belonging to specific object-classes. Hence, such information can be used to significantly reduce the amount of data actually processed. To this end, automatic object-class detection algorithms are developed.

- **Real-time visualization of massive 3D point clouds.** Interactive visualization requires real-time rendering techniques. Due to the massivity of the data, straightforward 3D rendering approaches are generally not feasible. This motivates out-of-core rendering techniques for massive 3D point clouds, which can contain billions of points. The main challenge is to guarantee interactive frame rates during user interaction and to achieve high rendering quality. In addition, rendering techniques can take into account object class, topology, and thematic information to improve visualization quality and expressiveness.

1.3 Contributions and Structure

This thesis presents concepts and techniques for processing, analyzing, and visualizing 3D point clouds. In particular, they focus on massive 3D point clouds using out-of-core strategies to achieve scalability. With these approaches – from an IT systems engineering point of view – 3D point clouds can be applied as fundamental, efficient data category by future IT applications, systems, and services.

Fundamentals of 3D point clouds are presented in Chapter 2. Key research questions are addressed in separate chapters including related work, implementation details, results, discussion, and future work. The main contributions of this thesis can be assigned to the subjects **Classification**, **Change Detection**, and **Rendering** of 3D point clouds:

- **Classification** denotes the process of computing and assigning category information to 3D points such as object-class categories. For example, object-class categories for aerial 3D point clouds include building, ground, vegetation, and water.

  "The classification of point clouds is an important step in the extraction of information. Whereas point cloud classification initially served to select points on the ground in the context of DTM production, the higher point densities obtained nowadays allow the extraction of various object types [...] like buildings, vegetation, vehicles, and water." (Vosselman 2013).

- **Change detection** denotes the process of identifying points and regions that have changed based on a comparison between a given 3D point cloud and a given 3D reference model.

  "The goal of the change detection process is to identify and mark all the points in the new scan that are most likely to be changes in the actual physical world and not merely changes in measurement." (Butkiewicz *et al.* 2008).

- **3D point cloud rendering** denotes the process of synthesizing images of 3D point clouds used for viewing and accessing the data.

  "point-based rendering has been shown to offer the potential to outperform traditional triangle based rendering both in speed and visual quality when it comes to processing highly complex models." (Botsch & Kobbelt 2003).

All presented techniques have been implemented and tested within a software framework for processing, analysis, and visualization of 3D point clouds (Chapter 6). The applicability is demonstrated with different case studies such as updating a 3D city model and monitoring railroad lines (Chapter 7).

1.3.1 Results

Core results of this thesis can be summarized as follows:

- **Classification:** The classification and segmentation of 3D point clouds according to object classes, including terrain, building, vegetation, water, and infrastructure, relies on analyzing the 3D topology in 3D point clouds. The presented technique does not require per-point attributes or representative training data and, therefore, can be applied for 3D point clouds having different characteristics (e.g., LiDAR or image-matching 3D point clouds). The approach is based on an iterative multi-pass processing schema, where each pass focuses on different topological features and considers already detected object classes from previous passes. To achieve scalability, the implementation uses out-of-core spatial data structures and GPU accelerated algorithms. The results show that 3D point clouds with semantics can substantially improve analysis algorithms and applications as well as enhance the features of 3D point cloud visualization.

- **Change detection:** Repeated data acquisition for geographic entities, such as landscapes, cities, or infrastructures, results in redundant, time-variant 3D point clouds (Kang & Lu 2011). To reduce storage requirements and to accelerate processing and analysis, change detection is required to determine the differences between newly captured and already stored 3D point clouds. The change detection approach is based on an out-of-core spatial data structure that stores data acquired at different points in time. A GPU-based processing schema is used to efficiently attribute 3D points with change information that can be used to perform quality control, evaluate the acquired data and update 3D models and contents (Matikainen 2004; Gröger & Plümer 2009).

- **3D Point cloud rendering:** 3D point cloud visualization is essential for understanding spatial information as well as to communicate analysis and simulation

results (Kreylos, Bawden & Kellogg 2008; Bettio *et al.* 2009; Kim & Medioni 2010). Point-based rendering techniques can cope with massive 3D point clouds and enable an interactive visualization and exploration of the data (Wimmer & Scheiblauer 2006; Richter & Döllner 2010b) without the need to generate polygon-based 3D models. The developed 3D rendering system implements different point-based rendering techniques (e.g., photorealistic, non-photorealistic, or based on point attributes (Botsch *et al.* 2005)).

- **Case studies and applications:** The presented classification and change detection approaches have been evaluated with 3D point cloud data for urban areas of cities and landscapes including datasets with up to 80 billion points and densities between 5 and 400 points per square meter. Applications such as monitoring railroad lines based on data from mobile mapping and detection of individual trees in urban areas are presented as examples to exemplify the benefits for real-world applications. The implemented out-of-core, real-time rendering system for massive 3D point clouds is used to present 3D point clouds and analysis results in the application specific context.

Chapter 2

Fundamentals of 3D Point Clouds

In this work, the term *3D point cloud* refers to a set of points in the three-dimensional Euclidean space having no structure, order, or hierarchy. Each point represents a discrete sample of a surface and is defined by its three-dimensional coordinates x, y, and z. Per-point attributes can provide additional information (e.g., color, surface normal, and local density). In this thesis, the term *massive 3D point cloud* is used to refer to 3D point clouds that exceed main memory capacities, i.e., vast amounts of 3D points (e.g., billions).

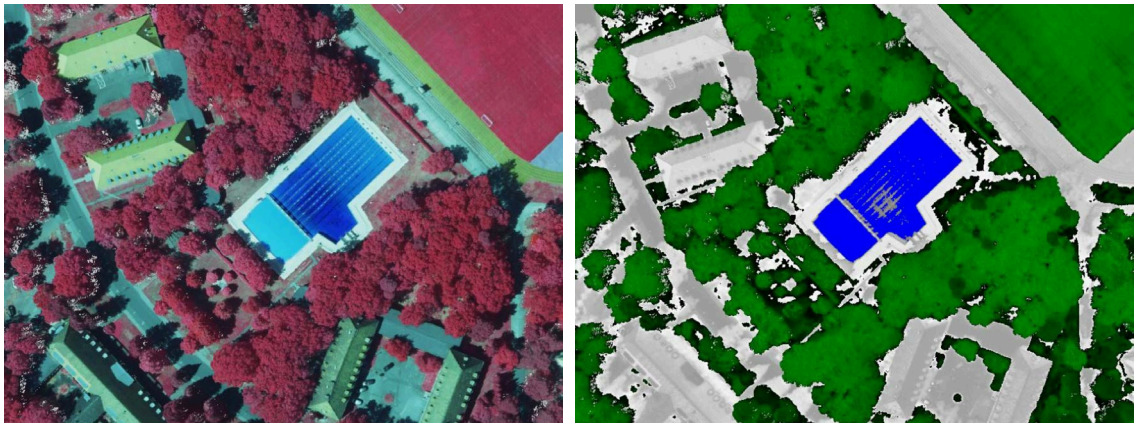
3D point clouds reflect the geometry of scanned targets by a discretized representation. In a growing number of applications, 3D point clouds can be used directly (e.g., simulation, analysis) or as base data for reconstructing 3D models. 3D point clouds are applied for planning, monitoring (Schneider 2006; Monserrat & Crosetto 2008; Trinder & Salah 2011; Kang & Lu 2011), and construction of 3D models (Zhou & Neumann 2008). In particular, 3D point clouds are used to automatically derive 3D models of buildings, surfaces, terrains, and vegetation to build up virtual 3D city models. Reconstruction methods allow for more and more automatic and fast generation of complex virtual geographic environments (Lafarge *et al.* 2010; Huang, Brenner & Sester 2013).

2.1 Capturing Methods - LiDAR vs. Dense Image Matching

This section gives an overview and comparison of LiDAR (Wehr & Lohr 1999) and image-based (Wenzel *et al.* 2012; Mayer, Sester & Vosselman 2013; Remondino *et al.* 2014) techniques to acquire 3D point clouds. Advantages and disadvantages of both capturing technologies are discussed with particular emphasis on 3D point clouds resulting from aerial data acquisition of cities and landscapes.

- **Capturing attributes**: LiDAR has typically several per-point attributes such as pulse and intensity information, which can be generally used to identify vegetation and surface materials (Lodha, Fitzpatrick & Helmbold 2007). Dense image matching data typically comes along with color information from aerial images such as RGB or RGBI (Gehrke *et al.* 2010).

- **Density**: The density of LiDAR-based 3D point clouds depends on the flight height, typically ranging between 4 and 25 points per m^2. Much higher densities can be achieved, for example, based on overlapping flight lines (Höfle, Hollaus & Hagenauer 2012). Dense image matching 3D point clouds are characterized by a higher density, which depends on the resolution of the input images. Most common aerial images have a ground sampling distance of 10 or 20 cm. This is the distance of pixel centers



(a) *Point cloud with color information from an aerial true orthophoto with CIR information. Red colors indicate near infrared, which are beyond the wavelengths for the color red.*

(b) *Point cloud with colors based on NDVI information from CIR images (green - vegetation; blue - water; gray - areas without vegetation).*

Figure 2.1: *Illustration based on a 3D point cloud with CIR data that can be used to compute the NDVI to separate surfaces belonging to organic, non-organic, and water structures.*

measured on the ground. Hence, 3D point clouds with a density of 100 points or 25 points per m^2 can be computed (Remondino *et al.* 2014).

- **Surface structure**: LiDAR data provides more precise 3D geometry, particularly for vegetation. Reasons are multiple returns of LiDAR rays that penetrate vegetation structures and hit the covered ground surfaces. In contrast, 3D point clouds from dense image matching tend to exhibit a 2.5 dimensional characteristic similar to digital surface models (Gehrke *et al.* 2010). LiDAR points have an irregular point distribution in contrast to the raster-based layout for x and y coordinates of dense image matching 3D point clouds. Dense image matching approaches operating on oblique airborne images, which represent well facades and other vertical objects, can be used to generate 3D point clouds (Cavegn *et al.* 2014), too. However, the availability of 3D point clouds from dense image matching is not a common practice. Figure 2.2 compares a LiDAR and dense image matching 3D point cloud for Berlin with different coloration schemas. The LiDAR 3D point cloud has an irregular structure, ground points below trees, and a density of 7-8 points per m^2. The dense image matching 3D point cloud shows a regular structure with a 2.5 dimensional characteristic and a density of 100 points per m^2.

- **Color information**: Color information can be added to 3D point clouds in a post-processing step by mapping aerial images to the data. However, a precise mapping for LiDAR data is not possible because LiDAR data and images are generally not captured at the same point in time (Richter & Döllner 2011b). In contrast, 3D point clouds from dense image matching can be attributed with accurate color information if input images from dense image matching computations are used. In addition to RGB images, RGBI or CIR images include the Normalized Differenced Vegetation Index (NDVI) that enable to separate living green vegetation from other structures and objects (Tucker 1979). Figure 2.1 shows a 3D point cloud colored with CIR and NDVI information.

- **Moving objects**: Non-static entities such as vehicles or trains can be captured well with LiDAR. Dense image matching approaches use multiple images from different points in time. Hence, the structure of the 3D point cloud for non-static entities could be incorrect.

- **Water surfaces**: Water surfaces can be captured only partly with LiDAR because the water surface does poorly reflect LiDAR rays. Dense image matching approaches are not capable to provide correct 3D point clouds for water surfaces (D'Angelo 2010). Reasons are the limited number of varying features for the captured water surfaces to estimate the correct height per pixel. Hence, 3D point cloud parts for water surfaces frequently show structures similar to vegetation or canyons.

- **Availability**: LiDAR has been used for decades to scan the Earth's surface; many workflows in the scope of administration and government define LiDAR scans at regular intervals. In contrast to collecting image data, it is much more expensive due to lower flight altitudes and denser flight stripes. Aerial images are captured typically once or twice a year for urban areas to produce and provide orthoimages. Aerial images from these acquisition campaigns can be used to generate 3D point clouds as additional product, which improves the availability of up-to-date 3D point clouds significantly.

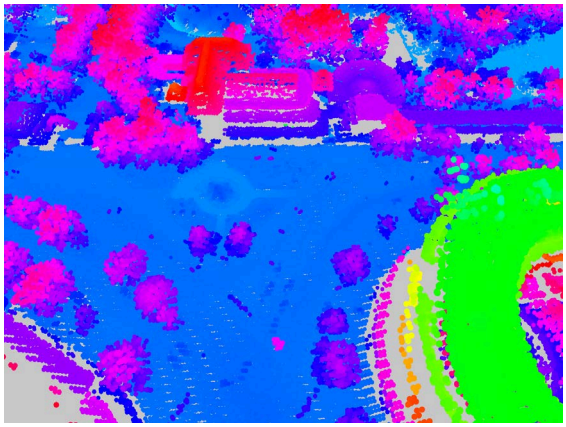
Both, LiDAR and dense image matching have their advantages and disadvantages. The relevance and usability does strongly depend on the data acquisition purpose and application context (Mayer, Sester & Vosselman 2013; Haala 2013). *"Several recent publications have compared ranging and imaging techniques based on factors such as accuracy, resolution and dense 3D reconstructions [..]. The choice between the two techniques nowadays depends primarily upon project constraints and requirements, budget and experience, and rather less on the geometric properties of point density and accuracy."* (Remondino *et al.* 2014). Most popular data acquisition techniques and systems are illustrated in Figure 2.3. They are categorized into LiDAR and image based techniques and arranged according to the scale of capturing area and acquisition costs. The illustration of acquisition cost and scale in Figure 2.3 is not linear and used to show the basic relation between capturing technologies.

2.2 Spatial Data Structures

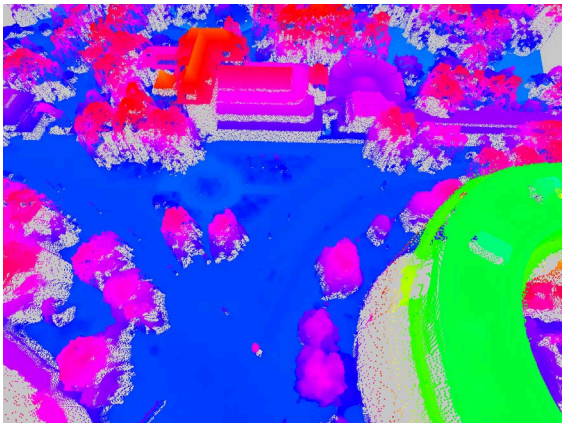
Spatial data structures are essential to partition and manage data. Popular data structures for triangles and polygons are R-trees (Guttman 1984; Beckmann *et al.* 1990; Zhu, Gong & Zhang 2007). They are used to partition a two-, three-, or even n-dimensional space for a fast spatial access, e.g., required for visibility computation (Stein, Limper & Kuijper 2014). Most popular data structures for 3D point clouds are quadtree, octree, and k-d tree derivations. Figure 2.4 illustrates the structure of a quadtree, octree, and k-d tree storing points in their leaf nodes. The construction is typically performed in a preprocessing step to optimize access times for processing and rendering tasks (Rusinkiewicz & Levoy 2000; Gobbetti & Marton 2004b; Wimmer & Scheiblauer 2006; Richter & Döllner 2010b; Goswami *et al.* 2013). This chapter introduces spatial data structures that have been implemented, evaluated, and used for processing, analysis, and visualization tasks presented in this thesis.

(a) *LiDAR scan of Berlin with colors from aerial images.*

(b) *Dense image matching scan of Berlin with colors from aerial images.*



(c) *LiDAR scan of Berlin with a color gradient.*



(d) *Dense image matching scan of Berlin with a color gradient.*

Figure 2.2: *Comparison of 3D point clouds from LiDAR and dense image matching of the same area.*

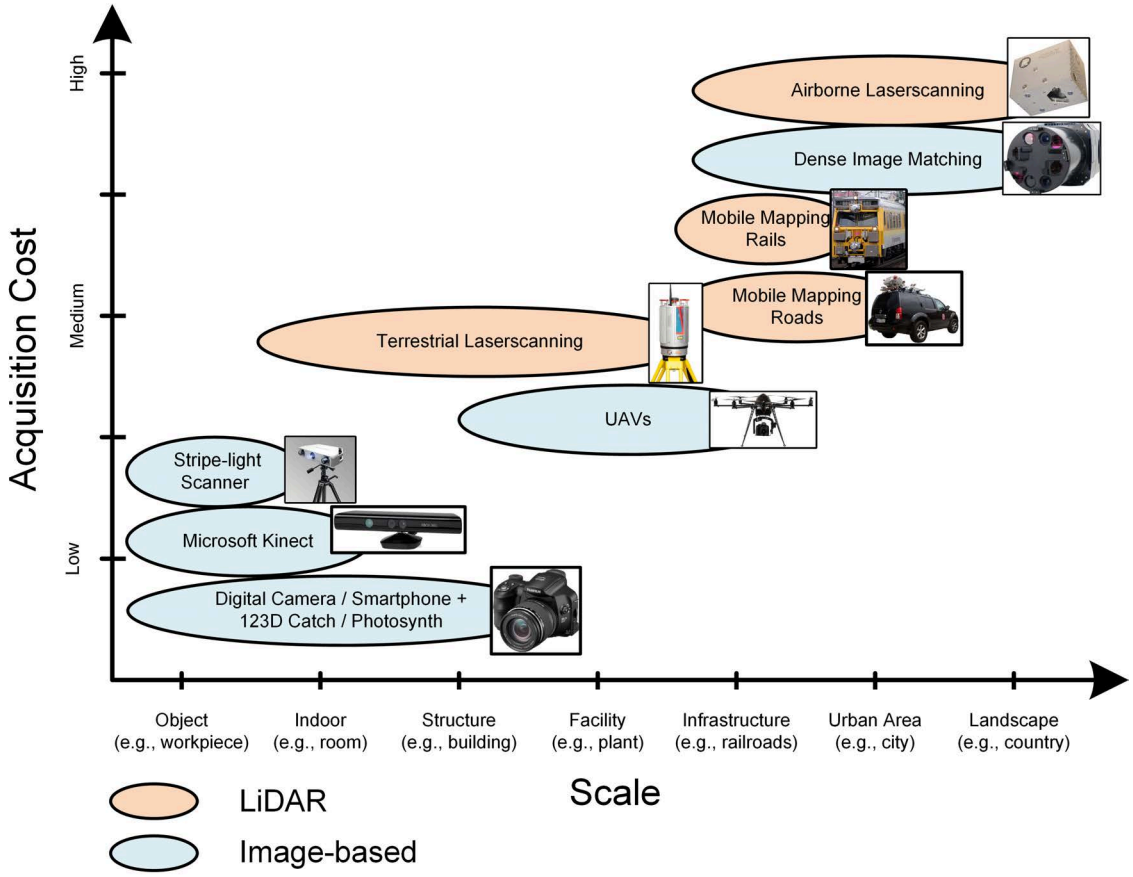


Figure 2.3: *Illustration of most common LiDAR and image-based capturing technologies arranged in terms of acquisition cost and application scale. It shows the basic relations between capturing technologies and not a linear scale.*

Spatial data structures presented in this thesis are basically composed of a hierarchy of nodes. The root node is derived from the spatial distribution of the data and has a bounding box (typically axis-aligned) that spans all points. Each node in the hierarchy has a spatial extend and is either an inner node with a defined number of child nodes or a leaf node without child nodes. Points of the 3D point cloud are assigned to nodes depending on the position in space. The number of points per node is typically limited depending on the application. The depth of the tree structure can be predefined (e.g., 8 levels) or adaptive depending on the spatial distribution of points (e.g., 1024 points per node). If the number of points for a spatial area exceeds this limit, a further subdivision of the related node is necessary. The subdivision can be performed adaptive or non-adaptive to the geometry. "*In some acceleration structures the location of subdivision planes is chosen so as to adapt to the geometry in the scene (e.g., a k-d tree), whereas in other acceleration structures the locations of bounding planes are predetermined, without looking at the geometry in the scene (e.g., a grid or octree)*" (Wald, Mark & Günther 2009).

2.2.1 Quadtree

Quadtrees partition two-dimensional data (e.g., 2D images data) in a recursive way, splitting a given rectangular area into four quadrants (Finkel & Bentley 1974; Zhang, Chengyuan and Zhang, Ying and Zhang, Wenjie and Lin 2013). In the scope of 3D point clouds, they can be applied with respect to a chosen reference plane (e.g., 2D geo-coordinate system). Each quadtree node is either an inner node with up to four child nodes, or a leaf node containing a subset of points (Shaffer & Samet 1987). The bounding box of the root node corresponds to the bounding box of the 3D point cloud. Quadtrees can be created in an adaptive way, taking into account the point density in the corresponding spatial area of a node. An inner node can also store a representative subset of the points of its child nodes providing a level-of-detail representation as required by many applications (e.g., real-time rendering) (Martinez-Rubi *et al.* 2015; van Oosterom *et al.* 2015). Quadtrees are also used as global index to organize local spatial areas with other data structures (Jian 2014).

2.2.2 Octree

Octrees, similar to quadtrees, recursively divide and partition spatial data within the three dimensional space (Meagher 1982). Each node represents a rectangular 3D subspace and an inner node can have up to eight child nodes (Figure 2.4). With respect to 3D point clouds, octrees compared to quadtrees provide more efficient handling and management because the data has a predominant three-dimensional extension and distribution (e.g., terrestrial scans of facilities and buildings). In particular, they support spatial partitioning, compression, and search operations on 3D point clouds. Octrees can be quickly constructed as required by time-efficient processing algorithms (Richter, Kyprianidis & Döllner 2013). Figure 2.5 shows an adaptive octree illustrated by bounding spheres for each node's subspace.

2.2.3 *k*-d tree

k-d trees, a specialized variation of the binary space-partitioning tree, are designed to manage points in a *k*-dimensional space and to support spatial queries (Bentley 1975). An important feature of a *k*-d tree is the balanced tree structure for irregularly distributed data (Goswami *et al.* 2010; Richter, Discher & Döllner 2015). A non-regular subdivision is

Table 2.1: *Comparison of quadtree, octree, and k-d tree properties. + indicates a good suitability, o means neutral, and - indicates that a property is not fulfilled or only partly applicable.*

Property	**Quadtree**	**Octree**	***k*-d tree**
Construction time	+	+	o
Equal number points per node	-	-	+
Balanced tree structure	-	-	+
Equal spatial extend of nodes per tree level	+	+	-
Speed of data excess	+	+	+
Ability to store level-of-details	+	+	+
Applicability for terrestrial data	o	+	+
Applicability for mobile mapping data	+	+	+
Applicability for airborne data	+	+	+
Applicability for multi-temporal data	+	+	+

used to divide the space to get equal-sized partitions in contrast to quadtrees and octrees. Hence, a more time-consuming preprocessing of the data is required due to the need to sort the points along the relevant dimension to determine the splitting plane (Goswami *et al.* 2010). The advantage of *k*-d trees are the balanced tree structure and almost equal-sized data chunks. This is important to implement efficient caching and memory swapping mechanism required by various applications such as real-time rendering (Preiner, Jeschke & Wimmer 2012; Richter, Discher & Döllner 2015).

2.2.4 Design Principles

In this work, the spatial data structures that have been implemented can be used to store and manage 3D point clouds based on the following design principles:

- Simple storage: Store points only in leave nodes (Elseberg, Borrmann & Nüchter 2011).
- Simple level-of-detail: Store points in leave nodes and a generalized or abstract representation (e.g., average position of child nodes) in inner nodes (Wand *et al.* 2007).
- Multi-resolution: Store points in leave nodes and a representative subset also in inner nodes to provide a level-of-detail representation.
- Adaptive multi-resolution: Store points either in inner nodes or leave nodes to reduce the required memory. Thus, all nodes together represent the entire data and are equal to the input 3D point cloud (Gobbetti & Marton 2004b).

Features and criteria of all presented spatial data structures, relevant for applications such as real-time rendering, processing, analysis, and management, are listed in Table 2.1.

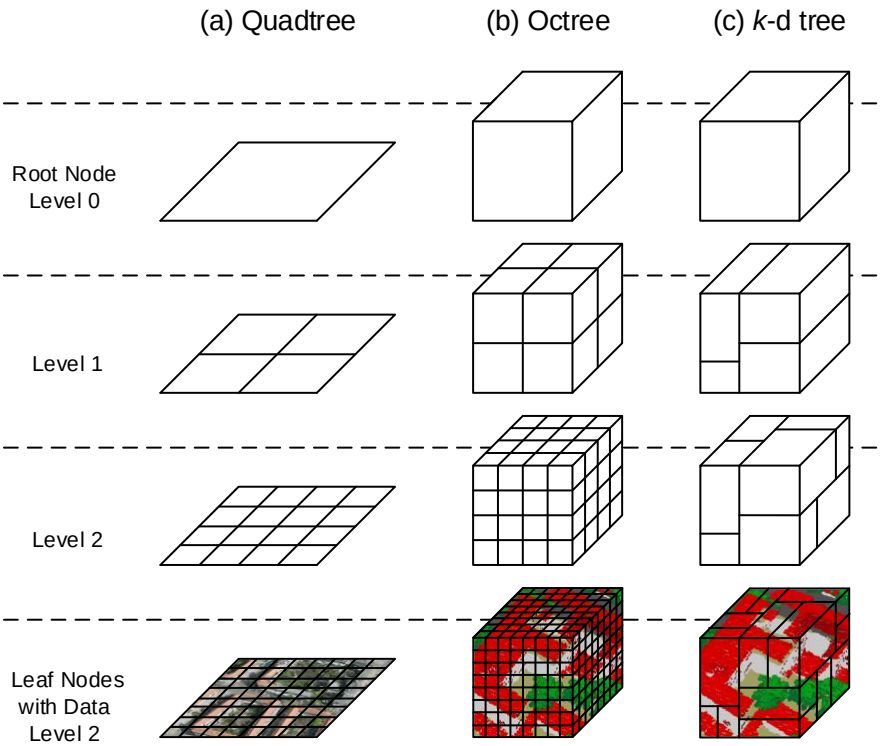


Figure 2.4: *Illustration of the quadtree, octree, and k-d tree structure used to arrange 3D point clouds by storing them in leave nodes.*

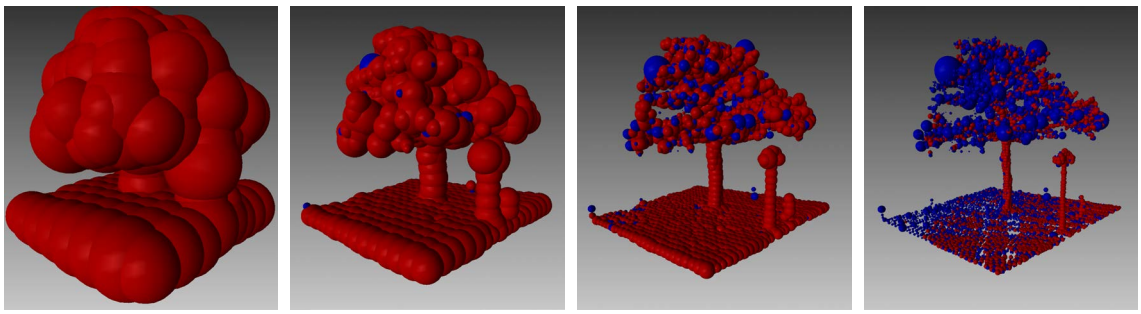


Figure 2.5: *Illustration of an octree structure with bounding spheres for tree levels 3–6. Each sphere represents a tree node and encloses all child nodes of the presented node. Inner nodes are illustrated with red and leaf nodes with blue color.*

2.2.5 Out-of-Core Construction

One challenge addressed in this thesis is the handling of massive amounts of data. 3D point clouds to be handled by today's and future applications exceed available main memory capacities and, therefore, require out-of-core or external memory algorithms. The general concept for the preparation of out-of-core spatial data structure can be divided into to three preprocessing steps. First, a division of the input data into subsets fitting into available main memory. The hierarchy and partition criteria of the first tree levels are used to define spatial areas for subsets. The input 3D point cloud is divided and each point is assigned to a subset on secondary storage. Second, all subsets are processed in a sequential order and serialized to secondary storage. The preprocessing of chunks in memory takes into account all data structure specific construction rules, e.g., subdivision criteria, node size, and LoD concept. The number of points, which can be stored and processed in main memory, is typically up to 100 million points on standard PC equipment with 16 GB main memory. Third, all preprocessed subsets are merged to the final tree structure on secondary storage. In general, the total size and capacity of the out-of-core tree structure is only limited by the available secondary storage capacity (e.g., hard disk, network drive).

2.2.6 GPU-based Processing

Most analysis and classification algorithms for 3D point clouds require to characterize and evaluate the proximity of each point. Hence, a large number of data queries are necessary. The number of queries is typically a multiple of the total number of points. Hence, these queries are typically time-consuming tasks and become a bottleneck in case of massive 3D point clouds. A solution to increase the processing speed are GPU-based processing schemas to parallelize time-consuming tasks, i.e., queries.

"*Creating an optimal GPU implementation not only requires redesigning serial algorithms into parallel ones but, more importantly, requires careful balancing of the GPU resources of registers, shared memory, and threads, and understanding the bottlenecks and tradeoffs caused by memory latency and code execution*"(Lee *et al.* 2012).

CUDA, OpenCL, and compute shader provide a flexible programming API for general purpose computations on GPUs. They enable the parallelization of algorithms by dividing the input data into chunks that are processed in parallel by a collection of threads (Farber 2011). The GPU facilitates a massive parallel processing using multiple processors and dedicated memory directly on the device. Similar to the out-of-core construction approach for spatial data structures, subdivisions of 3D point clouds and sequential processing of tiles on the GPU is required. A combination of spatial data structures and GPU-based processing schemas can be used to significantly increase the speed of processing tasks (Richter, Kyprianidis & Döllner 2013). In this thesis, all presented GPU-based processing schemas are based on NVIDIA's Compute Unified Device Architecture (NVIDIA Corporation 2011).

2.3 Used Datasets

3D point clouds captured with different technologies and at different scales are used to evaluate and proof processing and visualization techniques. All data sets presented in this thesis are listed in Table 2.2.

Table 2.2: *Characteristics of different 3D point clouds from aerial, terrestrial, and mobile data acquisition used to study and evaluate presented research objectives.*

Name	**Type**	**Size**	**Density**	**Vendor**
Berlin2008	Aerial LiDAR	4.7 bln	5-10 pts/m^2	virtualcitySYSTEMS
Berlin2013	Image Matching	80 bln	100 pts/m^2	Berlin Partner
Frankfurt2005	Aerial LiDAR	1.9 bln	7-8 pts/m^2	LVA Frankfurt
Frankfurt2009	Aerial LiDAR	7.1 bln	28 pts/m^2	LVA Frankfurt
Mansion	Terrestrial LiDAR	5.1 bln	1000 pts/m^2	Uni Colone
Railroads	Mobile Mapping	200 m	12 mln pts/m^2	DB Netz AG
Salzburg	Image Matching	400 pts/km^2	200 mln	virtualcitySYSTEMS
Kleinwalsertal	Image Matching	3.2 bln	25 pts/m^2	virtualcitySYSTEMS
Bournemouth2006	Aerial LiDAR	275 mln	38 pts/m^2	Ordnance Survey
Bournemouth2008	Aerial LiDAR	95 mln	21 pts/m^2	Ordnance Survey
Bournemouth2006	Image Matching	2.2 bln	153 pts/m^2	Ordnance Survey
Bournemouth2010	Image Matching	3.5 bln	220 pts/m^2	Ordnance Survey
Johannesburg2006	Aerial LiDAR	12.4 mln	1 pts/m^2	City of Johannesburg
Johannesburg2012	Aerial LiDAR	23.8 mln	2 pts/m^2	City of Johannesburg
Rotterdam2009	Aerial LiDAR	13 bln	60 pts/m^2	City of Rotterdam
Poland	Aerial LiDAR	24.1 mln	20 pts/m^2	SHH

Chapter 3

Point Cloud Classification

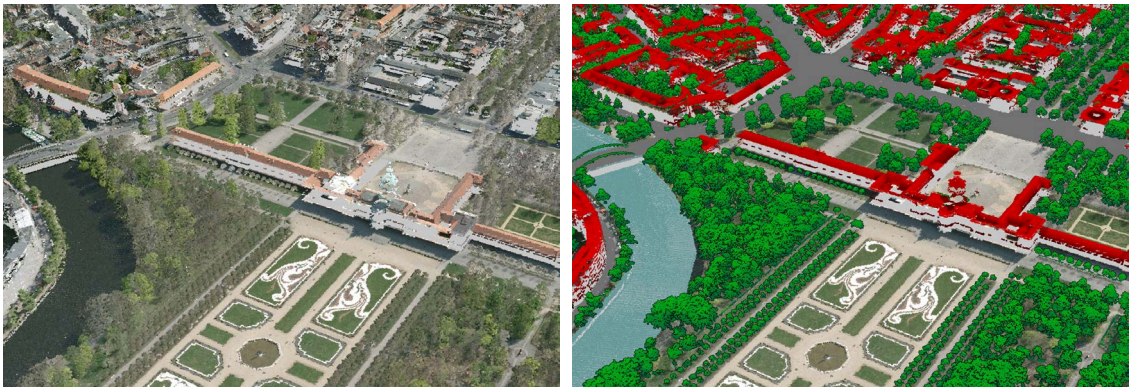
This chapter introduces concepts and techniques to classify 3D point clouds from airborne LiDAR scans and dense image matching into object classes. This *object-class segmentation* splits a massive 3D point cloud into 3D point subclouds according to the estimated surface category such as terrain, building, vegetation, water, and infrastructure. It relies on analyzing the point cloud topology and does not require per-point attributes or representative training data (Richter, Behrens & Döllner 2013). Object-class segmented 3D point clouds are important to improve analysis algorithms and applications as well as enhance visualization techniques (Figure 3.1) as they enable to treat each category independently. This chapter is partially based on the author's scientific publications in Richter and Döllner (2012), Richter, Behrens and Döllner (2013), and Richter and Döllner (2014).

3.1 Potentials and Usage of Classification

The classification of 3D point clouds is an essential processing step for applications, systems, and workflows (Lodha, Fitzpatrick & Helmbold 2007; Carlberg *et al.* 2009; Samadzadegan, Bigdeli & Ramzi 2010a; Grilli, Menna & Remondino 2017). In general, 3D point clouds provide only geometric data (i.e., x, y, z coordinates). The classification is used to add semantics information to the data and is required in a large number of applications such as reconstruction, modeling, and monitoring of buildings (Zhou & Neumann 2008; Meixner, Leberl & Brédif 2011; Yang *et al.* 2017a), terrain (Meng, Currit & Zhao 2010), and vegetation (Rutzinger *et al.* 2008). Point cloud classification provides a way to segment a massive 3D point cloud into disjoint subsets belonging to different surface categories, i.e., object classes such as building (Jochem *et al.* 2012), vegetation (Höfle, Hollaus & Hagenauer 2012), ground (Meng, Currit & Zhao 2010), city furniture (Golovinskiy, Kim & Funkhouser 2009), and infrastructure network (Clode, Kootsookos & Rottensteiner 2004). The process of enhancing 3D points by this kind of semantics is also called object-class segmentation due to the partitioning of the data according to object classes that can be found in urban areas (Richter, Behrens & Döllner 2013).

The classification can be used to optimize and enhance algorithms operating on massive 3D point clouds:

- Most importantly, a classified 3D point cloud can significantly reduce the required amount of data and improve the performance and accuracy of algorithms. For example, building reconstruction, infrastructure monitoring, or flood simulations can operate on a subset of the entire 3D point cloud belonging to object classes that are relevant for the domain of application.



(a) *3D point cloud with color information.* **(b)** *3D point cloud with object-class information.*

Figure 3.1: *Illustration of a 3D point cloud before and after object-class segmentation. The resulting object classes are illustrated with their respective colors: buildings - red, vegetation - green, water - blue, infrastructure - gray, terrain - aerial image colors.*

- Object-class enriched 3D point clouds can improve the visualization by applying different 3D rendering techniques to different object classes or by providing different interaction techniques for them.
- 3D point clouds with object-class information from different points in time enable a more specific detection of changes, e.g., after natural disasters to estimate damages.

The following requirements for classification approaches are important to provide added values for a variety of applications:

- Reliable classification for urban areas with different surface characteristics such as downtown, suburban, tree-covered, and hilly areas.
- Applicable for data captured with different remote sensing technologies, such as LiDAR and dense image matching with a 3D or 2.5D characteristic.
- No need for remote sensing technologies specific per-point attributes such as intensity, pulse, LiDAR return number, or RGB color.
- No need for training data, e.g., required for machine learning approaches, to avoid a time-consuming manual classification.
- Density adaptive classification to handle 3D point clouds with common densities from 5 to 400 points/m^2.

Existing object-class segmentation approaches frequently require additional per-point attributes (e.g., pulse or intensity information) or operate on manually classified training data sets that are used for machine learning approaches (Lodha, Fitzpatrick & Helmbold 2007; Jiang, Zhang & Ming 2008; Samadzadegan, Bigdeli & Ramzi 2010a).

The presented approach classifies massive 3D point clouds according to principal object classes found in urban areas. It is based on analyzing the *3D point cloud topology*, i.e., geometric relationships between points and segments, such as connectivity, local flatness, smoothness, and orientation. These attributes can be calculated in a preprocessing step and take into account only the position of each point (i.e., x, y, z). Resulting per-point attributes

are used as input for the segmentation as well as feature- and segment-based algorithms to determine object classes. The approach can be subdivided into the following steps: At first, per-point attributes are calculated by analyzing the proximity of each point. Second, an adaptive segmentation is performed taking into account previously calculated per-point attributes to determine patches (i.e., segments) of points with a similar characteristic. Third, all ground points are detected by analyzing the horizontal relationship of segments. Fourth, large segments with a characteristic topology for vegetation and building structures are determined. Sixth, all unclassified segments are analyzed within multiple passes taking already classified points into account. Finally, a classification of ground points is performed using geospatial data with semantics information (e.g., thematic maps). This determines specific object classes for ground points, such as terrain, water, and infrastructure that cannot be derived from the 3D point cloud topology in general.

3.2 Related Work

One category of related approaches for point cloud classification is based on additional attributes per point such as color (Charaniya, Manduchi & Lodha 2004), intensity (Charaniya, Manduchi & Lodha 2004; Lodha, Fitzpatrick & Helmbold 2007; Samadzadegan, Bigdeli & Ramzi 2010a), scan line (Sithole & Vosselman 2005; Douillard *et al.* 2011; Che & J.Olsen 2017), amplitude (Alexander *et al.* 2011), spectral (Matikainen *et al.* 2017; Morsy, Shaker & El-rabbany 2017), and pulse information (Charaniya, Manduchi & Lodha 2004; Clode & Rottensteiner 2005; Samadzadegan, Bigdeli & Ramzi 2010a), which are specific for the used scanning technology and device. Other classification approaches use per-point features, segment features (Ni, Lin & Zhang 2017), or machine learning algorithms (Lodha, Fitzpatrick & Helmbold 2007; Samadzadegan, Bigdeli & Ramzi 2010a; Hu & Yuan 2016; Ao *et al.* 2017), which requires training data sets that have to be prepared manually. The presented approach, however, does not require any additional information per point or manually classified training data sets.

One main purpose of capturing and processing 3D point clouds from aerial scans is to construct digital terrain models (DTM) (Raber *et al.* 2002; Shao & Chen 2008; Chen *et al.* 2016; Che & J.Olsen 2017), surface models (DSM), 3D building models (Zhou & Neumann 2008; Meixner, Leberl & Brédif 2011), and identify city furniture, e.g., cars (Zhang *et al.* 2014a; Zhang *et al.* 2014b). The creation of DTMs requires to separate ground points from non-ground points (Lodha, Fitzpatrick & Helmbold 2007). Sithole and Vosselman (2004) performed an experimental comparison of several ground filter algorithms with different 3D point clouds from airborne laser scanning. Meng et al. (2010) give a detailed overview and evaluation of ground filtering algorithms to derive digital elevation models (DEMs). Chen et al. (2017) present the state-of-the-art of DTM generation approaches. The presented classification is based on the approach of Rabbani et al. (2006) to derive segments in a 3D point cloud. Ground segments are identified by analyzing the topological structure and relation between neighboring segments.

Building reconstruction algorithms require roof points as input data to derive building models. These algorithms typically require only points that belong to planar roof areas (Zhou & Neumann 2008; Awrangjeb & Fraser 2014; Wang *et al.* 2016). Therefore, an exact and complete detection of all points belonging to façade structures and small roof elements, e.g., smoke stacks, roof dormers, and other roof constructions, is not necessary (Meixner,

Leberl & Brédif 2011; Yang *et al.* 2017a). The purpose of the presented approach is to detect all building elements that can be recognized in the 3D point cloud.

Well-proven vegetation classification approaches use first and last pulse information (Charaniya, Manduchi & Lodha 2004; Clode & Rottensteiner 2005; Jiang, Zhang & Ming 2008), reflection intensity (Charaniya, Manduchi & Lodha 2004), or color attributes (Charaniya, Manduchi & Lodha 2004) of each point. These attributes are not always available, e.g., for 3D point clouds resulting from dense image matching approaches. Secord and Zakhor (2007) address the problem of identifying trees in LiDAR data. Rutzinger et al. (2008) perform an object-based analysis for vegetation detection. The presented approach is applicable for 3D point clouds without specific per-point attributes because it operates only on the point cloud topology.

Golovinskiy et al. (2009) uses data from airborne and mobile scans to identify city furniture (e.g., newspaper boxes, traffic lights, cars) in urban environments. Yao (2013) and Weinmann (2017) detect individual trees along road corridors using data from mobile laser scanning systems. Other approaches detect power lines (Clode & Rottensteiner 2005) using per-point return information and road networks (da Silva, Centeno & Henriques 2011) using digital images.

Machine learning and support vector machines (SVM) require a representative and manual classified 3D point cloud as training data (Lodha, Fitzpatrick & Helmbold 2007; Jiang, Zhang & Ming 2008; Samadzadegan, Bigdeli & Ramzi 2010a; Yang *et al.* 2017b). Shapovalov et al. (2010; 2011) use non-associative markov networks to classify airborne and terrestrial laser scans. Lodha et al. (2007) uses per-point features (height variation, normal variation, and return intensity) to group points with a similar characteristic. Alexander et al. (2011) use a decision tree approach that can dynamically handle a large number of attributes. The manual preparation of training data is time consuming and has to be accomplished for 3D point clouds that differ in their characteristics, e.g., point density and point distribution. In contrast to the presented approach, no specific domain knowledge is used, e.g., arrangement of ground, vegetation and building segments.

Carlberg et al. (2009) introduce a multi-category classification system for airborne LiDAR data with similar objectives compared to the presented system. Multiple classifiers perform a region growing algorithm and a segment-wise classification. Planar and scattered segments are identified using manual classified training data. This approach results in a large number of unclassified points, especially for segments resulting from building façades and small scattered areas. The presented approach reduces the amount of unclassified points by performing an iterative multi-pass analysis.

3.3 Object-Class Segmentation

The object class classification workflow can be divided into the following processing steps and is illustrated in Figure 3.2:

- *Preprocessing* includes detection and removal of outliers and duplicates to ensure a defined data quality for the further object-class segmentation. Calculation of basic per-point attributes such as color information (Richter & Döllner 2011b) from aerial images and surface normals by analyzing the local point proximity (Mitra & Nguyen 2003).

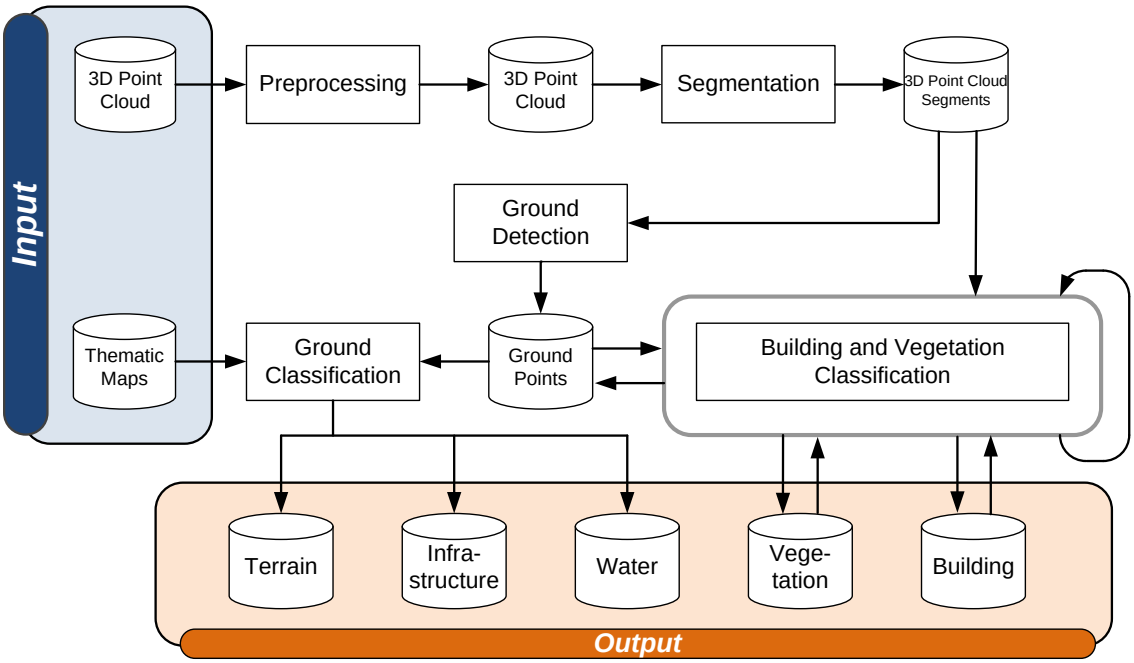


Figure 3.2: *System overview showing the data workflow and components of the object-class segmentation pipeline.*

- *Segmentation* groups points with a similar characteristic in a local proximity (Rabbani, van den Heuvel & Vosselman 2006) that most likely belong to the same object class.
- *Classification* to identify segments belonging to ground, building, and vegetation (Carlberg *et al.* 2009).

3.3.1 Preprocessing

The preparation of the 3D point clouds has the aim to ensure a defined data quality and to attribute the data with per-point attributes that are relevant for a further classification. In general, 3D point clouds from LiDAR scans include outliers (Arabsheibani, Abedini & Kanani Sadat 2015), duplicates and noise resulting from measurement errors which need to be removed (Sotoodeh 2006). Therefore, the number of neighboring points within a local point proximity (e.g., 2 meters) is determined. If no other point can be found, the point is defined as outlier and not considered in the further processing. Duplicates are identified by comparing the point position with points in a small proximity (e.g., 0.01 meter).

The following per-point attributes are computed in the preprocessing stage and used to classify the 3D point cloud and also to enhance the visualization:

- *Color data.* RGB and CIR color information is in general not required to classify 3D point clouds. A mapping of 2D image data, i.e., aerial images, cannot be performed exact enough to 3D point clouds from LiDAR scans. Typical problems occur below vegetation because the same color information is mapped the tree crown and ground points under trees (see Figure 2.2 (a)). However, a mapping of RGB and CIR data to 3D point clouds from dense image matching is possible and can be used to improve the detection of vegetation (see Chapter 2.1 and Figure 2.2 (b)).

- *Per-point normal* to approximate the surface of the local point proximity. It is computed using the covariance matrix of the nearest neighbors (e.g., 10 nearest neighbor points) and the corresponding eigenvectors and eigenvalues (Hoppe *et al.* 1992).

3.3.2 Segmentation

Segmentation is a process that partitions the 3D point cloud into disjoint subsets with similar characteristics (Grilli, Menna & Remondino 2017). Common segmentation approaches perform a region growing to group spatially connected points within a defined proximity (e.g., 0.5 meter) (Dong *et al.* 2017). The general objective is the robust detection of features (Daniels *et al.* 2007), e.g., for simplification and reconstruction purposes. The aim of the presented segmentation is to group only points belonging to the same object class (e.g., ground, building, vegetation) to enable a segment-based classification. However, points belonging to different object classes can be located next to each other (e.g., trees close to building roofs or ground surfaces close to building façades). For that reason, an additional segmentation criteria, such as a slope value, is required that considers surface structure properties (Yang, Zhang & Li 2017). Rabbani et al. (2006) introduced a segmentation approach that considers surface smoothness in addition to the local connectivity of points. A *surface smoothness* and *residual value* parameter are used by a region growing algorithm. The normal variation between points (surface smoothness) is used to stop the segmentation at edges and offsets in the 3D point cloud. This parameter is difficult to determine for the entire 3D point cloud because the segmentation should work for smooth regions (e.g., planar roofs) but also be tolerant to curved areas (e.g., bended building structures). For that reason, a second parameter is used that indicates areas of high curvature in the 3D point cloud (residual value). This enables to be flexible regarding surface smoothness and locally adjusting normal variation based on the 3D point cloud structure. Suitable segmentation thresholds for the presented datasets are 0.5 for surface smoothness and 0.05 as residual value. An increased residual value results in larger segments that could contain multiple object class points. A decreased residual value increases the size of small segments. The residual value strongly depends on the point density.

3.3.3 Segment Classification

Ground Detection

The ground detection algorithm operates on segments that result from the segmentation pass. Ground points typically form large-area segments due to smooth surface characteristic and connectivity of ground areas. In contrast, building and tree-covered areas generate smaller segments (Meng, Currit & Zhao 2010). For that reason, large-area segments (e.g., $> 50\text{k}\ m^2$) are assumed as ground segments. These segments include the majority of ground points. Special cases occur for small ground regions surrounded by buildings (e.g., backyards) or located in dense vegetated areas that are not connected with large-area ground segments. To identify those segments, all non-classified segments are analyzed concerning the relative position to already classified ground points. For each point in a segment the nearest already detected ground points are determined. If the height difference to these ground points is below a defined threshold (e.g., 0.5 meter) the point is tagged as

possible ground point. The whole segment is identified as ground segment if the segment contains a significant number of possible ground points (e.g., 80 percent).

Building and Vegetation Detection

Building and vegetation points are identified with an iterative multi-pass approach that is the most challenging part of the object-class segmentation pipeline. Traditional approaches use per-point or per-segment feature values for the classification. Typical features are normal distribution, regularity, horizontality and flatness (Zhou & Neumann 2008) of segments or points. They are derived from the structure and topology of the local point proximity. In general, planar segments are classified as roof elements and scattered segments are classified as vegetation (Sithole & Vosselman 2005). This approach works well for large segments but becomes unreliable for small segments, i.e., with less than 50 points. These segments are difficult to analyze with current state-of-the-art approaches and therefore often labeled as unclassified points (Carlberg *et al.* 2009). Regions with an accumulation of unclassified points typically occur due to:

- Scattered roof segments (e.g., stacks, antennas, roof constructions)
- Mixed roof and tree-covered areas (e.g., trees covering roofs, trees in the courtyard)
- Building façades with a small number of points (e.g., 1 $point/m^2$)

To overcome this limitation, an iterative multi-pass classification approach is used that benefits in each iteration from classified points in previous passes. Each pass analyzes aspects of the 3D point cloud topology that are specific for individual object classes. At first, only large segments that can be clearly or most likely assigned to an object class are identified. Second, a more precise vegetation analysis pass is performed using an additional segmentation and taking into account already classified vegetation and building points. In the third pass, remaining small segments are classified by considering a larger proximity.

Large Segment Analysis. All large segments (i.e., with more than 50 points) are analyzed in relation to already detected ground points. For each point in a segment, all subjacent ground points are determined. If a segment contains many points above the ground it is identified as vegetation. This assumption can be made because LiDAR rays partially run through vegetation up to the ground, in contrast to building structures where no subjacent ground points can be found. Consequently, segments containing many points without subjacent ground points can be most likely classified as building segments. Exceptions of this assumption are dense vegetation areas where LiDAR rays do not reach the ground surface and tree covered areas that are captured with image-based remote sensing technologies (e.g., dense image matching). For that reason, an additional validation pass is performed to analyze large segments without subjacent ground points. For each point, a flatness attribute is computed based on the normal variation to the nearest neighbor points. If the majority of points belong to a scattered region, the overall segment is classified as possible vegetation, otherwise as building. The results of the large segment analysis are building, vegetation, possible vegetation, and unclassified segments. Figure 3.3 (a) illustrates the input segments (left) and detected building and vegetation segments (right). Possible vegetation and unclassified points are colored white.

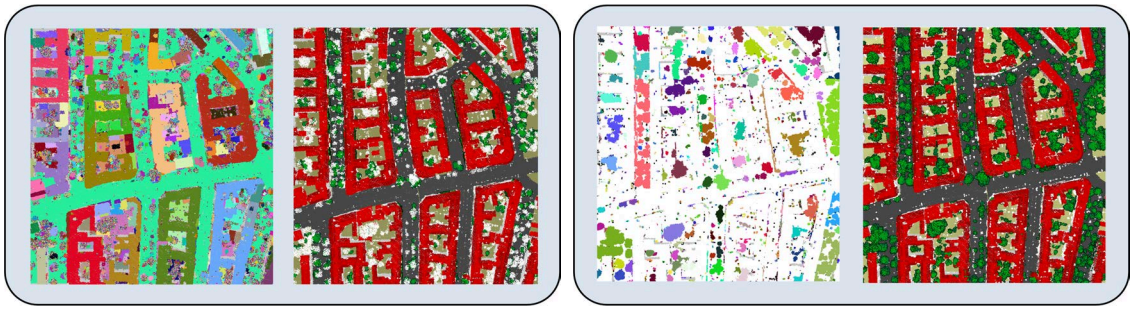
Vegetation Analysis. In contrast to areas with urban structures, tree-covered areas result in a large number of small segments (Figure 3.3 (a) left). These segments are difficult to classify due to similarities to other small segments (e.g., from buildings, stacks, antennas, roof constructions, and city furniture). Consequently, a segment-based analysis of vegetation points using segments of the previous pass becomes unreliable. Therefore, an additional segmentation pass is performed taking into account typical 3D point cloud topology for vegetation areas and already detected vegetation segments. The segmentation is performed on already detected vegetation, possible vegetation, and unclassified segments without smoothness parameters. This results in merged segment clusters and small segments are added to already detected vegetation segments (Figure 3.3 (b) left). In the next step, all segments that exceed a defined size (e.g., 50 points) are analyzed because these segments enable a reliable classification. This analysis is performed with respect to the amount of vegetation and possible vegetation points that were detected in the large segment analysis. If the majority of points belong to these object classes, all points in the segment are assigned to the object class vegetation. Figure 3.3 (b) shows segmentation and vegetation analysis results. The remaining unclassified segments are very small and not connected to already detected vegetation areas.

Small Segment Analysis. In this pass, all remaining unclassified segments are analyzed taking into account already detected building, vegetation, and ground points. These segments typically tend to occur at small building roof elements, façades, low vegetation, and city furniture (Figure 3.3 (b) - right). Small roof elements can be detected if all neighboring points are already detected building points. Façade segments are more difficult to detect, due to an incomplete surface and a small number of façade points. The detection is performed by specifying a point neighborhood volume, similar to a tube, with a small radius (e.g., 0.5 meter). A point is classified as façade point, if building points can be found above, and building or ground points below a point. If the majority of points in a segment fulfill this property, all segment points are identified as façade points belonging to the object class building. All remaining segments belong to low vegetation or city furniture. Assumptions are made by considering already classified ground points. For instance, cars can be found above or next to infrastructure points.

Classification of LiDAR and Dense Image Matching 3D Point Clouds. The used capturing technology affects the surface characteristic of 3D point clouds as described in Section 2.1. Differences occur due to the 2.5D and 3D nature of dense image matching and LiDAR data. Large forestry areas are difficult to classify in 2.5D point clouds because of missing ground points under vegetation and the smoother surface characteristic in contrast to 3D point clouds from LiDAR scans. Tree surfaces and roof surfaces could show an unusual flatness which makes it difficult to separate vegetation from building segments. To overcome this limitation, CIR data can be used to classify points and segments with a structure that does not clearly indicate the characteristic of vegetation or non-vegetation.

3.3.4 Specific Classification of Object Classes

The presented object-class segmentation splits a 3D point cloud into disjoint object classes ground, building, and vegetation. Each object class can be divided into subclasses using



(a) *Large segment analysis.* **(b)** *Vegetation analysis.*

Figure 3.3: *3D point cloud segments (left) and resulting object classes (right) for large segment (a) and vegetation (b) analysis.*

thematic data or object-class specific analysis approaches. Common subclasses for the following object classes are:

- **Ground**: Terrain, water, land use (e.g., cropland), and infrastructure (e.g., road networks and railway) (Lillesand, Kiefer & Chipman 2015).
- **Building**: Roof area, roof construction, façade, and building entity (e.g., belonging to house number) (Haala & Kada 2010).
- **Vegetation**: Low vegetation, high vegetation, vegetation species, and vegetation entity (e.g., single tree) (Secord & Zakhor 2007).

The ground detection pass outputs all points that represent the ground surface. In general, these points have a 2.5D characteristic, i.e., there are no overlapping structures. A detailed differentiation of ground points can rarely be derived from the 3D point cloud topology due to the smooth connection of ground surface areas. For that reason a more specific classification of ground points can be performed by utilizing additional geospatial data, e.g., thematic maps, open street map data, shapefiles, or any other data that provides georeferenced information. Typically, this thematic data can be automatically generated (e.g., with feature extraction from aerial images) but may require manual effort (Meng, Currit & Yang 2010).

The geographic registration and mapping between thematic data (e.g., maps) and the 3D point cloud could include a certain degree of uncertainty and varying accuracies. To overcome this limitation, multiple data sources can be used to get precise ground classification results.

The classification of ground points in the presented approach has been implemented with a flexible approach and is performed in three steps. First, all available data sources are determined and prepared. For instance, Web Map Services (WMS) are used to obtain maps with thematic information. Priorities for each input source and object class are used to handle redundant, contradictory, and overlapping information. For example, road network information are available in thematic maps and open street map data, but differ in coverage and accuracy. Second, a *ground classification texture* is generated by combining object-class information and priorities from the input data. In the last step, the ground classification texture is used to determine the object-class information for each ground point. Figure 3.4 illustrates the ground classification process for terrain, road, railroad, and water surfaces.

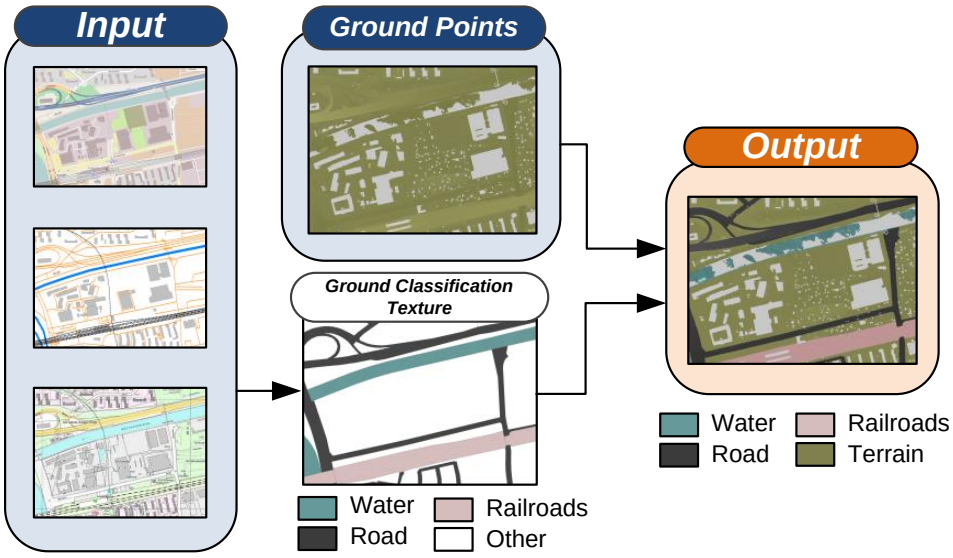


Figure 3.4: *Ground classification workflow using multiple geospatial data sources as input to subdivide ground points into infrastructure, water and terrain points.*

A more specific classification of building and vegetation points cannot be performed reliable with thematic maps in contrast to ground points. Reasons are the 3D characteristic of vegetation and building points (e.g., horizontally overlapping), the missing data accuracy (e.g., rapid change of vegetation structures), and incompleteness of additional geospatial data. For that reason, structure and segment information can be used to identify roof, façade, and roof structure within all building points as described in Section 3.3.3. The detection of points that belong to individual buildings is a well-studied field in research because it is required for the automatic building reconstruction (Zhou & Neumann 2008; Zhang, Yan & Chen 2006). In a first step, a segmentation of already detected building points is used to create roof patches. These roof patches can be merged if they share the same roof ridge. In a second step, all boundary points are determined and used to create a detailed building outline. In general, this outline has many details and is jagged due to the irregular structure of the 3D point cloud. For that reason, a third step is performed to simplify the boundary. The identification of individual buildings with the same roof type and height that adjacent to each other, e.g., in a building complex, can be improved be taking into account address data (Jarzabek-Rychard 2012).

3.4 Out-of-Core and GPU-based Processing

The algorithms and processes of the object-class segmentation have different memory and computation time requirements. This becomes particularly relevant when the data amount increases, i.e., billions of points need to be processed. To perform the classification for massive 3D point clouds on standard consumer hardware, spatial data structures, out-of-core processing strategies, and a parallel GPU-based processing of time consuming tasks are required and have been implemented.

To overcome main memory limitations, an adaptive out-of-core quadtree, introduced in Chapter 2.2.1, is used to handle the entire 3D point cloud and enable fast data integration,

subdivision, and updates (Figure 2.4 (left)). The quadtree enables an adaptive selection of data tiles that fit into available main memory. To ensure a correct classification for points close to tile boundaries, each tile includes overlapping areas of neighborhood tiles. Depending on the distribution and density of the 3D point cloud, suitable tiles that fit into available main memory are selected and processed in a sequential order.

The bottleneck of the object-class segmentation are multiple local neighborhood requests for all points belonging to the 3D point cloud. Different requests are required in the preprocessing (e.g., filtering and normal calculation), segmentation, and classification stage. A single local neighborhood analysis, e.g., at the beginning of the classification, is not possible due to varying parameters:

- Request volume (e.g., sphere, tube)
- Request direction (e.g., below or above requested points)
- Volume size (e.g., 10 meter, 0.5 meter)
- Requested object class (e.g., ground, vegetation, building)

An octree is used as spatial data structure to improve the performance of local neighborhood requests (Figure 2.4 (middle)). The octree instead of a k-d tree is used because request with different volumes and directions are necessary and the construction is faster. A GPU-based implementation is used to increase the performance. The process of requesting the local neighborhood for points in a parallel way is divided into five steps. At first, all points belonging to object classes that should be processed are transferred to GPU memory and arranged in an octree. Second, request parameters for volume, size, and object class are defined. Third, all points whose local neighborhood should be analyzed are transferred to GPU memory. Fourth, these points are assigned to parallel executed threads that traverse the octree structure and determine all point that fulfill the specified request parameter. Last, results are stored in a data structure on the GPU and transferred back to CPU memory. Special cases occur, if the number of points in the requested area exceeds the available GPU memory capacity (e.g., 2 GB). To overcome this limitation, a subdivision into suitable chunks is necessary. The presented approach performs best for requests with a large number of query points (e.g., 100k points) to use the full processing capacities of the GPU. Measurements show that the GPU implementation performs on average 25 times faster than a comparable CPU implementation.

3.5 Results and Evaluation

This section presents object-class segmentation results for 3D point clouds with different characteristics illustrated in Figure 3.5 – 3.8. To prove the usability of the presented approach areas with different surface characteristics have been classified such as downtown, suburban, and tree-covered regions. The datasets have been captured with LiDAR technology or generated with dense image matching approaches. The larges dataset contains approximately 80 billion points and the density of the data ranges from 5 up to 400 points/m^2. A detailed evaluation of the accuracy and performance is presented is performed for three tiles (480.000 m^2), representing different environment characteristics (i.e., a tree-covered, suburban, and downtown area). 3D point clouds used for the evaluation are listed in Table 3.1. They show the following capabilities:

Table 3.1: *Datasets used to show the capability of the object-class segmentation to classify 3D point clouds with different characteristics (DT - downtown, SU - suburban, TC - tree-covered).*

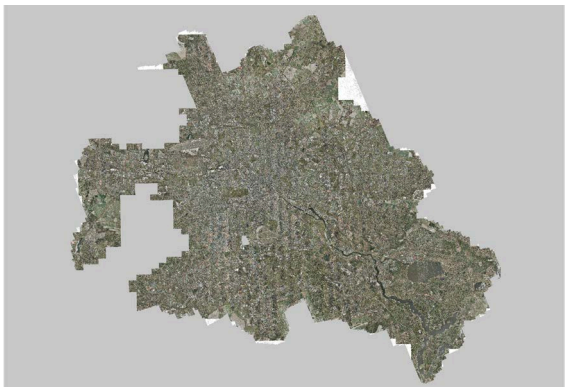
Name	Type	Density	#Points	Charact.	Fig.
Berlin2008	LiDAR	5-10 pts/m^2	4.7 bln	DT; SU; TC	3.5
- Tile1	LiDAR	11 pts/m^2	5.4 mln	TC	3.9 (a)
- Tile2	LiDAR	6 pts/m^2	2.8 mln	SU	3.9 (b)
- Tile3	LiDAR	5 pts/m^2	2.4 mln	DT	3.9 (c)
Frankfurt2005	LiDAR	7-8 pts/m^2	1.9 bln	DT; SU	3.6 (a)
Frankfurt2009	LiDAR	28 pts/m^2	7.1 bln	DT; SU	3.6 (b)
Poland	LiDAR	20 pts/m^2	24.1 mln	DT; SU	3.7
Bournemouth2010	Image Matching	220 pts/m^2	3.5 bln	SU	3.8

- Massive 3D point clouds: Figure 3.5 shows classification results for a 3D point cloud of an entire city. It is proven that the classification can be applied to massive data sets without any restrictions regarding the number of points.
- Different densities: Figure 3.6 illustrates results for an area captured at different points in time with varying resolutions. The classification was successfully applied to LiDAR 3D point clouds with a resolution between 5 and 100 points/m^2. A resolution of less than 5 points/m^2 reduces the classification quality.
- Different capturing seasons: Figure 3.7 shows a dense LiDAR point cloud captured in the summer with a dense canopy and less points below trees. The results show that the presented approach can separate building and vegetation structures reliable even for 3D point clouds with no or less ground points below vegetation.
- Dense image matching: Figure 3.8 shows a point cloud from dense image matching. It is proven that the classification can be applied to data sets with a 2.5D characteristic. In contrast to LiDAR data NDVI information is recommended to improve the segmentation stage.

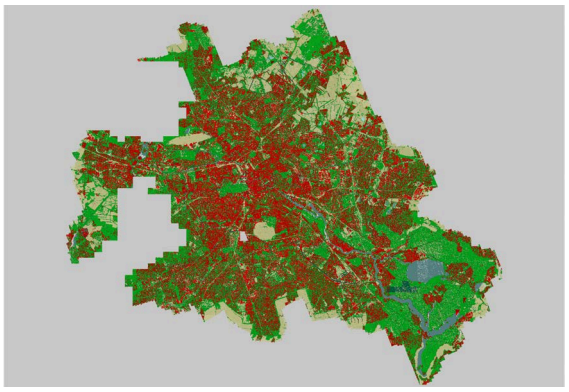
3.5.1 Accuracy Evaluation

The accuracy evaluation is performed for 3D point cloud with different surface characteristics. To this end, three subsets of the dataset *Berlin2008* were classified manually to generate ground truth data. Tile 1 is a tree-covered area with a few buildings, Tile 2 is a suburban area with solitary buildings and Tile 3 represents a downtown are with large connected buildings. The manual classification was performed with a tool that provides functionality for a human operator to classify points into ground, vegetation, and building. A classification of ground points into terrain, infrastructure and water points was not performed. The classification and evaluation was done for a representative part (25 %) of each tiles (Figure 3.9) to limit the required time for manual classification.

All manually classified tiles are compared with object-class segmentation results and indicate a precision of at least 93.4 % for vegetation, 95.2 % for building and 97.9 % for



(a) *3D point cloud with colors from aerial images.*

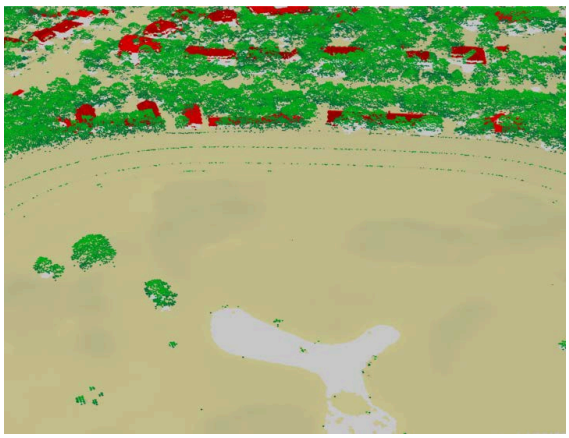


(b) *3D point cloud colored based on object-class information.*

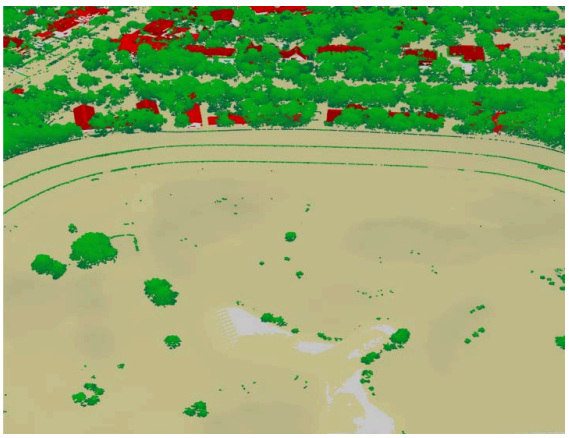
(c) *Detailed view of upper right.*

(d) *Detailed view of lower left.*

Figure 3.5: *Illustration of object-class segmentation results for a massive 3D point cloud of Berlin from an aerial LiDAR scans.*



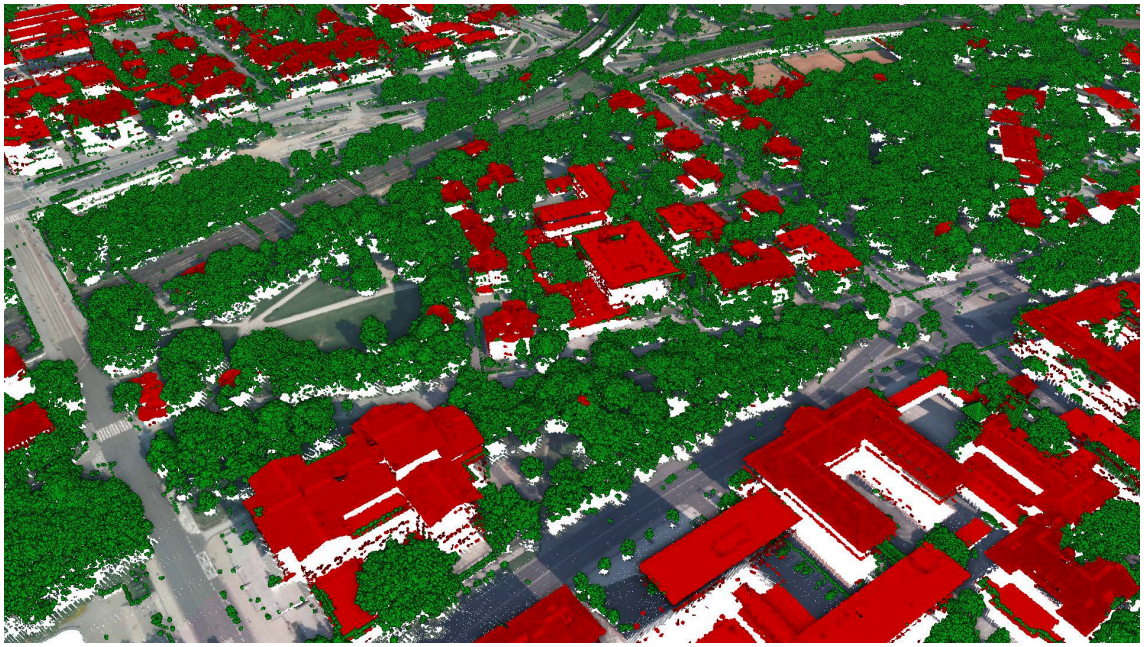
(a) *Frankfurt2005 with 7-8 points/m^2.*



(b) *Frankfurt2009 with 28 points/m^2.*

Figure 3.6: *Illustration of object-class segmentation results for 3D point clouds with different resolutions.*

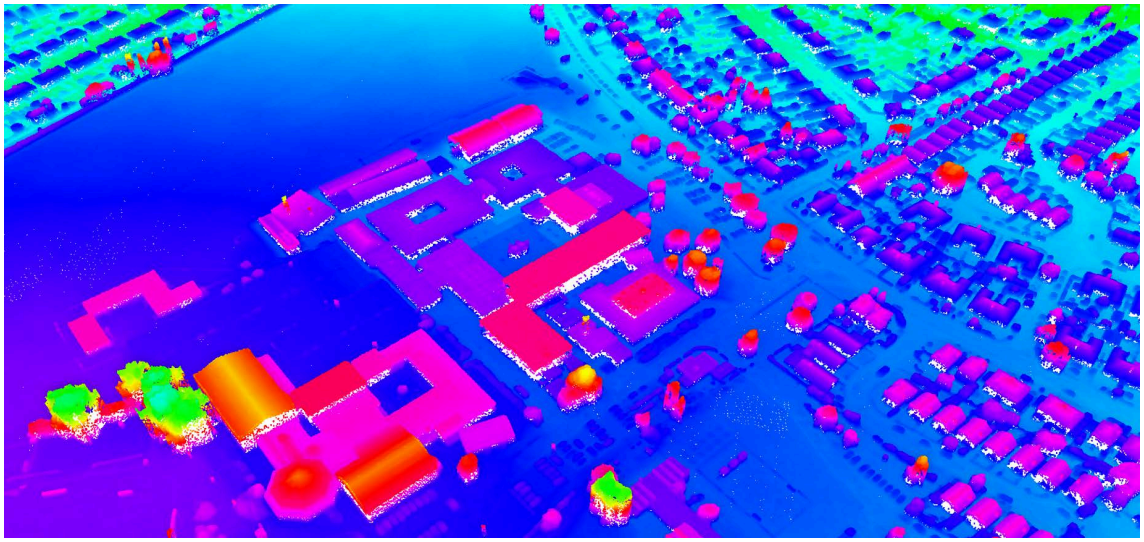
(a) *3D point cloud with colors from aerial images.*



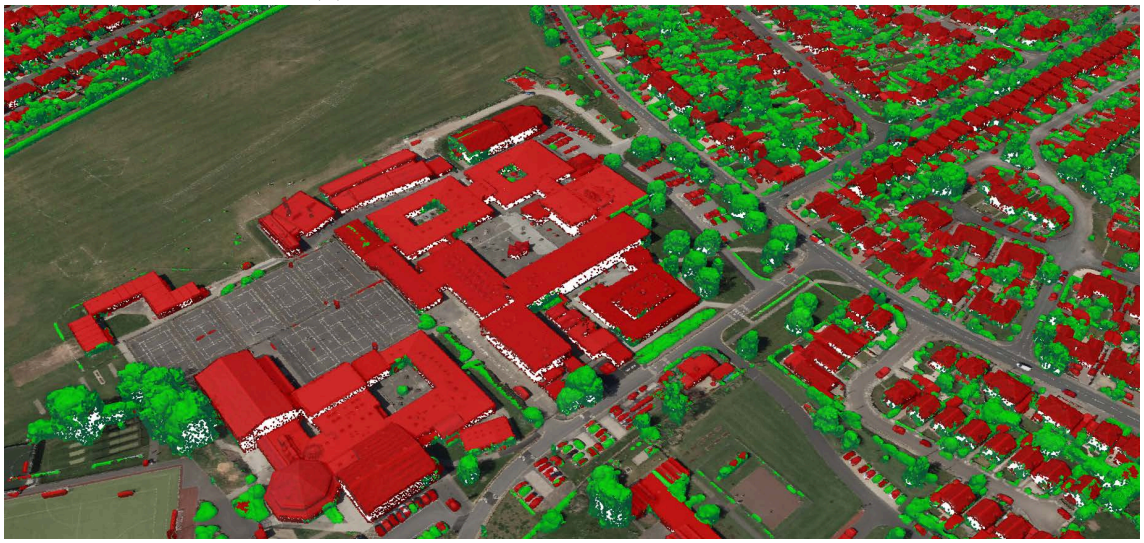
(b) *3D point cloud colored based on object-class information.*

Figure 3.7: *Illustration of object-class segmentation results for a 3D point cloud from an aerial LiDAR scans in the summer with a dense canopy.*

(a) *3D point cloud with colors from aerial images.*



(b) *3D point cloud with color gradient based height.*



(c) *3D point cloud colored based on object-class information.*

Figure 3.8: *Illustration of object-class segmentation results for a 3D point cloud from dense image matching.*

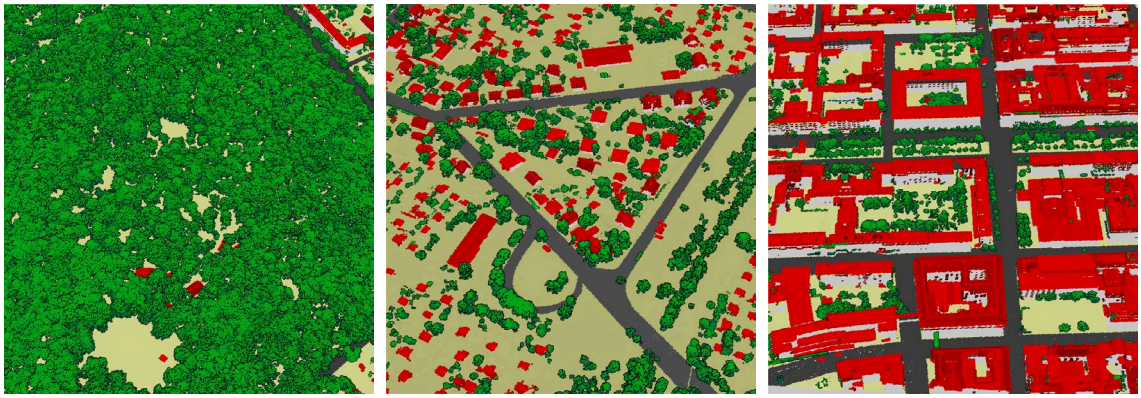
Table 3.2: *Object class and segment statistics for a tree-covered (Tile 1), suburban (Tile 2), and downtown (Tile 3) area with different characteristics (Figure 3.9) and points belonging to the Berlin2008 dataset (Figure 3.5).*

	Tile 1		**Tile 2**		**Tile 3**		**All Tiles**	
# Points	5 354 k	(100%)	2 819 k	(100%)	2 414 k	(100%)	4 601 mln	(100%)
- Terrain	2 464 k	(46.0%)	1 831 k	(65.0%)	571 k	(23.2%)	2 399 mln	(52.1%)
- Infrastructure	182 k	(3.4%)	197 k	(7.0%)	652 k	(26.5%)	428 mln	(9.3%)
- Water	25 k	(0.5%)	12 k	(0.4%)	8 k	(0.3%)	55 mln	(1.2%)
- Vegetation	2 587 k	(48.3%)	456 k	(16.2%)	160 k	(6.5%)	1 228 mln	(26.7%)
- Building	96 k	(1.8%)	323 k	(11.5%)	1 023 k	(41.5%)	491 mln	(10.7%)
# Segments	1 154 k		270 k		106 k		975 mln	
- Large	18 k		5 k		3 k		11 mln	
- Small	1 136 k		265 k		103 k		964 mln	

Table 3.3: *Object-class segmentation evaluation based on manually classified 3D point clouds used as ground truth (GT) for tree-covered, suburban, and downtown area. Precision, recall, and total error (TErr) are derived from true positives (TP), false positives (FP), false negatives (FN), and true negatives (TN) for vegetation, building, and ground points.*

	#GT	**#Result**	**TP**	**FP**	**FN**	**TN**	**Pre.**	**Recall**	**TErr.**
Tile 1									
Vegetation	556 697	557 301	555 621	1680	1076	587 634	99.70%	99.81%	0.24%
Building	31 657	30 721	30 511	210	1146	1114 144	99.32%	96.38%	0.12%
Ground	557 657	557 989	556 954	1035	703	587 319	99.81%	99.87%	0.15%
Tile 2									
Vegetation	88 202	89 298	83 414	5 884	4 788	432 138	93.41%	94.57%	2.03%
Building	98 515	99 230	94 517	4 713	3 998	422 996	95.25%	95.94%	1.66%
Ground	339 507	337 696	335 209	2 487	4298	18 4230	99.26%	98.73%	1.29%
Tile 3									
Vegetation	98 412	93 204	90 874	2 330	7 538	448 969	97.50%	92.34%	1.80%
Building	159 075	160 449	157 093	3 356	1 982	387 280	97.91%	98.75%	0.97%
Ground	292 224	297 077	291 045	6 032	1 179	251 455	97.97%	99.60%	1.31%

ground point detection. The results are listed in Table 3.2 and include the point distribution for each object class and the number of segments that result from the segmentation pass. Point density varies due to multiple LiDAR returns, especially caused by vegetation. Precision, recall, and error rates for each object class are listed in Table 3.3. The precision for detecting vegetation and building points indicates that the presented approach performs best for tree-covered and downtown areas where vegetation and building structures are mainly separated. However, also suburban areas with building and vegetation structures with a similar size and overlapping volumes can be separated with a precision of 93% for vegetation and 95% for buildings. The ground detection performs best for tree-covered and suburban areas because the number of ground points, in relation to the total number of input points, is much higher in contrast to downtown areas. Ground segments are more connected and cover larger areas which is advantageous for the ground detection pass. The results show that the presented object-class segmentation provides a robust classification with a maximum total error of 2.03 % for vegetation, 1.66 % for building, and 1.31 % for ground point detection.



(a) *Tile 1: tree-covered area.* **(b)** *Tile 2: suburban area.* **(c)** *Tile 3: downtown area.*

Figure 3.9: *Illustration of three tiles with different characteristics: tree-covered (a), suburban (b), and downtown area (c).*

3.5.2 Performance Evaluation

This section presents the performance evaluation of the object-class segmentation. First, performance measurements for three tiles with different characteristics that can be processed in main memory are presented and discussed. Second, the performance of the overall workflow to classify a 3D point cloud with almost 5 billion points is evaluated. All tests were performed on an Intel Xeon CPU with 2.66 GHz and 12 GB main memory. For GPU computations a NVIDIA GeForce GTX 480 with 1536 MB device memory and 480 CUDA cores is used.

Measurements in Table 3.4 show that the preprocessing and segmentation time are only slightly affected by the point cloud characteristic. Also the performance of the ground detection pass is only affected by the overall number of points. The increased processing time for the vegetation analysis for Tile 1 is caused by the large number of vegetation segments. Different processing times for the small segment analysis result from the total amount of small segments, e.g., façades, that need to be processed. In summary, the object-class segmentation of urban areas performs best due to the lower number of segments.

The overall object-class segmentation workflow for the 3D point cloud *Berlin2008* takes less than 23 hours. The processing without GPU accelerated algorithms would require about 250 hours. The average throughput of the object-class segmentation system is 3.3 million points per minute. However, there is still potential to increase the processing performance, e.g., by performing the segmentation pass on the GPU.

3.6 Discussion and Future Work

This chapter presents an approach for object-class segmentation of massive 3D point clouds from airborne scans that facilitates the use of 3D point clouds for a broad range of applications. The concept and implementation is based on a processing pipeline that classifies the data into the main categories building, vegetation, and terrain as well as water and infrastructure. The detection of building and vegetation points is performed with an iterative multi-pass algorithm taking into account 3D point cloud topology. The amount of unclassified points is reduced due to the iterative process. Key feature of the presented approach is that 3D point clouds are robustly processed without additional

Table 3.4: *Object-class segmentation performance for three tiles with different characteristics (Figure 3.9) and the overall 3D point cloud of Berlin (Figure 3.5) in seconds (all points / per million points).*

Task	Tile 1	Tile 2	Tile 3	All Tiles
Point Cloud Import	1.39 s (0.26 s)	0.54 s (0.19 s)	0.60 s / 0.25 s)	22.38 m (0.29 s)
Preprocessing	5.41 s (1.15 s)	2.31 s (1.04 s)	2.33 s (1.16 s)	87.12 m (1.14 s)
Segmentation	35.69 s (7.61 s)	13.79 s (6.20 s)	11.52 s (5.73 s)	534.83 m (6.97 s)
Ground Detection	1.90 s (0.40 s)	0.90 s (0.41 s)	0.51 s (0.25 s)	30.78 m (0.40 s)
Ground Classification	1.59 s (0.34 s)	0.76 s (0.34 s)	0.74 s (0.37 s)	26.33 m (0.34 s)
Large Seg. Analysis	10.71 s (2.28 s)	2.86 s (1.28 s)	3.16 s (1.57 s)	125.63 m (1.64 s)
Vegetation Analysis	5.21 s (1.11 s)	0.60 s (0.27 s)	0.35 s (0.17 s)	46.84 m (0.61 s)
Small Seg. Analysis	34.85 s (7.43 s)	11.84 s (5.32 s)	11.25 s (5.60 s)	475.42 m (6.20 s)
Point Cloud Export	0.71 s (0.13 s)	0.37 s (0.13 s)	0.31 s (0.13 s)	12.32 m (0.16 s)
Overall Classification	97.46 s (20.71 s)	33.97 s (15.18 s)	30.77 s (15.23 s)	1361.65 m (17.75 s)

per-point attributes or training data. Thematic maps can be used to split detected ground points into disjoint object classes.

The accuracy evaluation indicates that the approach works reliable for areas with different characteristics such as tree-covered, suburban, and downtown areas. In addition, it can be applied to 3D point clouds from LiDAR or dense image matching with different resolutions. The presented approach was combined with out-of-core concepts and GPU-based processing schemas to enable the object-class segmentation even for massive 3D point clouds.

The following drawbacks have been identified where the object-class segmentation does not work reliable depending on some environment scenarios. For example, large vegetation covered balconies are detected as vegetation, heavy smoke (e.g., from power plants) is classified as vegetation, and roofs with large window areas are not classified correct. Also low point densities ($< 5\ points/m^2$) could cause incorrect results for small structures. In some cases, round roofs (e.g., like domes) are not classified correct because they have the same characteristic like trees. The ground detection for dense image matching 3D point clouds from large forests could be difficult because of the limited number of available ground points. However, most limitations are mainly caused by the quality of the data rather than the presented approach. A more adaptive segmentation taking into account NDVI or color data, especially for dense image matching 3D point clouds, could improve the overall object-class segmentation quality.

To summarize, object-class segmentation facilitates analysis and provides a fundamental instrument to improve performance and functionality of applications, systems, and workflows. One application field that benefits from 3D point clouds with object-class information is disaster management. Here, the fast analysis of large 3D point clouds is essential to estimate object-class specific damages to buildings, infrastructure, or forests. Another application is the selective update for geospatial data, such as surface models, maps, and 3D city models. In addition, object-class segmented 3D point clouds are essential to enhance the exploration and visualization of massive 3D point clouds.

Future work will focus on classification approaches that use a fusion of 3D point clouds captured with mobile and aerial remote sensing systems. These dense datasets present a more complete surface representation of an environment and will lead to a more detailed

object-class segmentation (e.g., for city furniture, cars, power lines, façade elements). The usage of multiple 3D point clouds from different points in time presents new possibilities to perform a more reliable classification and identification of static and non-static entities in urban environments. This requires additional analysis passes and results in a more expensive analysis and processing. However, it enables to generate more detailed and realistic 3D models (e.g., 3D city models) that can be used in a variety of application domains. The processing performance can be improved by implementing the remaining time-consuming CPU-based processing steps, such as segmentation and segment analysis, on the GPU.

Chapter 4

Change Detection

This chapter introduces concepts and techniques to detect changes in 3D point clouds generated at different points in time. The ability to efficiently compare 3D point clouds from entire cities and landscapes represents a challenging part. This chapter presents a GPU-based approach based on an out-of-core spatial data structure designed to store massive 3D point clouds acquired at different points in time. All points are attributed with computed change information to enable later selective storage, processing, analysis, and presentation. The approach, for example, enables to draw conclusions about temporal changes in 3D geodatasets based on point cloud change detection at reasonable processing and rendering times. This chapter is partially based on the author's scientific publications in Richter and Döllner (2011) and Richter, Kyprianidis and Döllner (2013).

4.1 Potentials and Applications of Change Detection

Due to increasing geometric precision, decreasing acquisition costs, and higher frequency of capturing, applications and systems are more and more faced with massive sets of 3D point clouds. The need for efficient processing techniques for massive 3D point clouds is intensified because many applications demand for frequent scans and simultaneous use of scans taken at different points in time. However, there is no need to store, process, and manage the entire 3D point clouds of all scans. Differences between redundant captured surfaces occur generally only for locally bounded parts of the data. The process of identifying and marking regions or points that have most likely changed in the actual physical world (Butkiewicz *et al.* 2008) is known as *change detection* or *3D difference analysis*. Applications of change detection include:

- Performing selective updates for existing 3D point clouds (Kang & Lu 2011; Kalasapudi, Turkan & Tang 2014)
- Monitoring structures and constructions over time (Girardeau-Montaut *et al.* 2005; Schneider 2006; Tuttas *et al.* 2017)
- Evaluating the quality of captured data (Anil *et al.* 2011; Kim *et al.* 2015)
- Identifying static and non-static entities by temporal 3D point clouds (Richter & Döllner 2014)
- Updating 3D models and contents such as 3D city and building models (Matikainen 2004; Gröger & Plümer 2009)

- Visualizing multiple degrees of change or deformations in the built environment (Stojanovic *et al.* 2018)

In case of massive 3D point clouds with billions of points, straightforward approaches of change detection turn out to be computationally not feasible. Existing solutions typically thin out or rasterize the 3D point cloud to reduce the data size (Van Gosliga, Lindenbergh & Pfeifer 2006). This, however, leads to a loss of accuracy and makes it difficult to match the original unfiltered data with the results of the change detection.

This chapter presents approaches to process 3D point clouds of arbitrary size and to detect changes in sets of massive 3D point clouds that do not force users to thin out or to rasterize the data. To this end, an out-of-core spatial data structure is used to store datasets acquired at different times, which ensures fast access to subsets of the stored 3D point clouds. In particular, the technique supports the subdivision of the data into spatially arranged parts, which enables efficient distribution of workloads to multiple CPU cores or the GPU. Based on the out-of-core spatial data structure, three different parallel computation schemas are presented and evaluated regarding to performance aspects. The first is a multi-core CPU-based implementation, while the other two perform the computations on the GPU by using GPGPU and CUDA technology.

4.2 Related Work

Change detection in the scope of virtual 3D city models is typically performed based on polygonal geometry derived from 3D point clouds. Approaches using point-to-mesh and mesh-to-mesh algorithms were introduced by Besl and McKay (1992). The need for an explicit surface model generally requires a time-consuming preprocessing to reconstruct polygonal models (Vosselman *et al.* 2004). In Butkiewicz et al. (2008) all points were projected to a triangulated surface model derived from a 3D point cloud to detect changes in the urban structure. The accuracy of this approach depends on the resolution and quality of the surface model and, in contrast to the presented approach here, is only applicable to 2.5-dimensional geodata. Gosliga et al. (2006) presented a deformation analysis for a tunnel, acquired with a terrestrial laser scanner. However, the calculation is only performed on 1% of the 3D point cloud data due to the interpolation based on a regular grid used to perform the calculation. By contrast, the presented approach directly uses the raw data without thinning out or rasterizing the 3D point clouds. Hence, it can be used for 3D point clouds acquired with different devices and technologies for different scales (see Section 2.3), e.g., objects, infrastructure, and landscapes.

Point-to-point comparison approaches have the advantage that they can directly operate on 3D point clouds. A theoretical and computational framework for global comparison of uniformly sampled 3D point clouds was presented by Mémoli and Sapiro (2004). A further framework for 3D point clouds with divergent point data distribution, for example resulting from terrestrial laser scans, was presented by Girardeau-Montanut et al. (2005). Both approaches work well for small 3D point clouds (e.g., with a few million points). However, they cannot be applied to 3D point clouds with billions of points because the processing time will increase dramatically. Girardeau-Montanut et al. (2005) address the problem of missing surface information due to occlusion during the data acquisition process and solved it with visibility maps to mark regions with missing surface information for the change detection process. Barber et al. (2008) derived 3D point cloud data from aerial

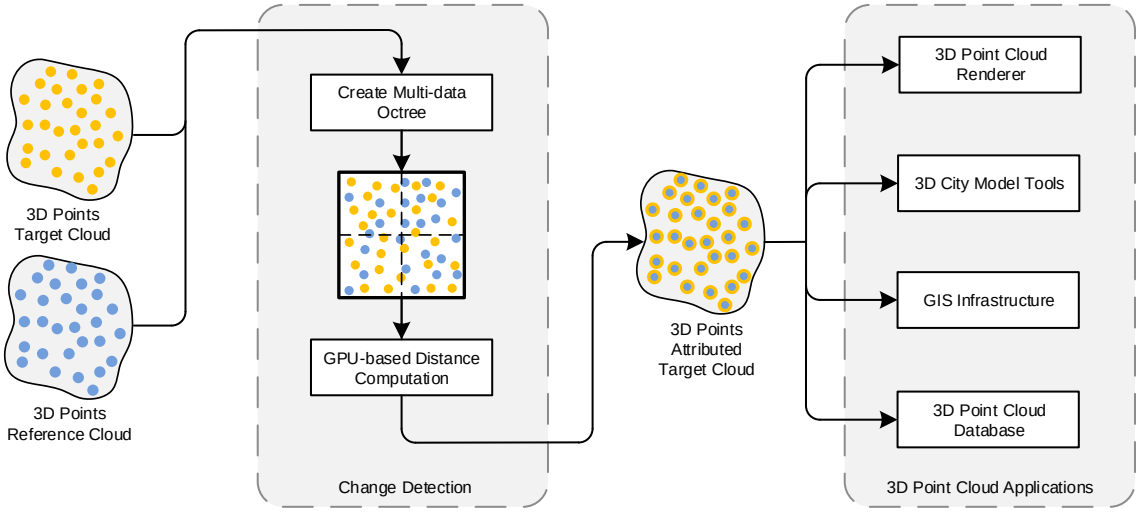


Figure 4.1: *Illustration of the system architecture showing components for change detection and rendering.*

imagery and airborne LiDAR systems organized in an octree data structure to identify significant changes in the urban area. Leite et al. (2009) presented a system to determine k-nearest neighbors (KNN) for 3D point clouds on the GPU. The data is organized as grid data structure on the GPU that can be evaluated in real-time. Their system is able to process data in real-time, but limited to data sizes that fit into GPU memory. By contrast, the presented system only needs to find the nearest neighbor, but has to deal with massive datasets. Therefore, the designed data structure enables out-of-core swapping mechanisms between GPU and CPU as well as CPU and secondary storage without the requirement to evaluate the data in real-time. Heinzle et al. (2008) presented a flexible hardware processing unit for 3D point clouds with focus on fundamental and computationally expensive operations. In the presented approach there is no need to do complex operations on 3D point clouds. Other GPU-based k-nearest neighbors approaches are presented by Qiu et al. (2009) and Pan et al. (2011).

4.3 Change Detection

A schematic overview of the system and data processing pipeline is illustrated in Figure 4.1. The processing starts with the creation of a spatial data structure and the calculation of 3D differences between a 3D point cloud called *target cloud* and another 3D point cloud called *reference cloud* that is used as reference object. The distance between the target point and the closest point of the reference cloud is used as metric for the *degree of change*. For each point in the target cloud the Euclidean distance to the closest point in the reference cloud is calculated and stored as an attribute of the related point in the target cloud.

A straightforward approach to detect changes in 3D point clouds would be to determine the minimum distance for all points of the target cloud to points of the reference cloud by simply computing the distance between all possible pairs. However, for massive 3D point clouds it is computationally not feasible due to quadratic complexity of the algorithm. The presented approach is able to process datasets of arbitrary size by utilizing a specialized out-of-core data structure that supports an efficient organization and access of the data.

4.3.1 Concept Multi-Data Octree

The spatial data structure is a fundamental component of the overall change detection process. It can be applied for:

- Multi-temporal 3D point clouds
- Massive 3D point clouds
- 3D point clouds from airborne, mobile, and terrestrial data acquisition
- Rapid preparation and preprocessing computations
- Fast access to subsets of 3D point clouds

The *multi-data octree*, introduced in this chapter, is designed to fulfill these requirements. It extends the concepts of an octree data structure and manages 3D point clouds with different characteristics (see Section 2.2.2). Out-of-core concepts for the multi-data octree facilitate the fast processing for data that exceeds the available main memory. In contrast to quadtrees, octrees can handle arbitrary 3D point clouds, e.g., from airborne, mobile, and terrestrial data acquisition, which differ in their horizontal and vertical spatial distribution and density. A k-d tree is not applicable, since the subdivision of the space depends on the arrangement of the point data. It would be difficult to subdivide the space to construct a balanced k-d tree containing more than one 3D point cloud. Moreover, the preprocessing times to prepare the data structure would increase due to the need to sort the data along the longest axis for each tree level to perform a correct spatial subdivision. The decision to store multiple datasets in a single tree structure is motivated by the following reasons. The memory required to arrange the 3D point clouds is reduced since structure information needs to be stored only for one data structure. Also management, swapping and selection of points for multiple datasets can be performed faster because the distance calculation algorithm, in general, requests data for the same spatial part of the tree structure. The structure of the multi-data octree is illustrated in Figure 4.2.

Each node of the multi-data octree is either an inner node, with up to eight child nodes, or a leaf node containing points of the target and reference cloud. Inner nodes subdivide the space in up to eight subspaces of uniform size. The root node represents a bounding box enclosing all 3D points of the target and reference cloud. The subdivision of nodes is controlled by the number of points in the spatial volume of the node. If this amount is above a fixed threshold, a subdivision is performed and the node becomes an inner node. Otherwise no further subdivision is performed and points of the target and reference cloud are stored in the leaf node. Inner nodes contain information about their child nodes and the number of points of the reference and target cloud in the represented subtree. This is necessary to estimate memory requirements for each part of the multi-data octree when loading data into main memory.

4.3.2 Construction Multi-Data Octree

The preparation of the multi-data octree is performed in a preprocessing step. If the number of 3D points of the target and reference cloud exceeds the capacity of the main memory, an in-memory calculation of the overall multi-data octree is not possible. In this case, the data is subdivided, using secondary storage, according to the hierarchy of the

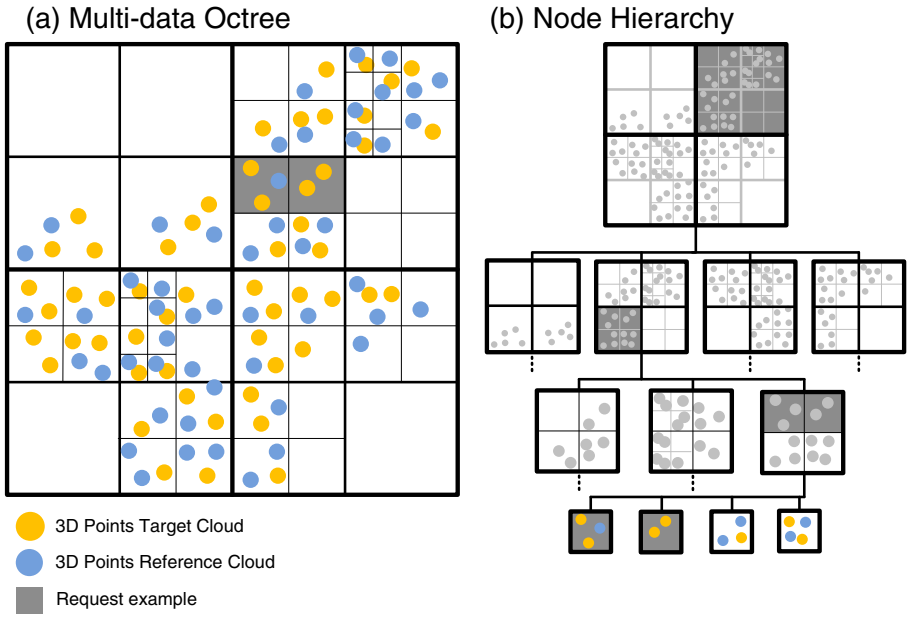


Figure 4.2: *(a) Illustration of the multi-data octree filled with target and reference points. (b) Node hierarchy resulting from the 3D point distribution.*

octree cells until the subclouds fit into main memory. For these subclouds the multi-data octree is prepared in main memory and serialized to secondary storage. Then, all prepared parts of the multi-data octree are merged on secondary storage to build the multi-data octree containing all points of the target and reference cloud.

The access to the data is performed with data requests based on a bounding volume. These volumes will be referred to as *in-memory chunks* containing the multi-data octree structure with all inner and leaf nodes that are inside the requested bounding volume or that intersect it. The time to load the in-memory chunks into main memory depends on the multi-data octree depth and the maximal capacity of leaf nodes. The total number of points in an in-memory chunk also depends on the maximum capacity of leaf nodes, since all points of intersected leaf nodes are selected to avoid a bounding volume test for each point. This is performed to improve loading times for in-memory chunks.

4.3.3 Distance Computation

The distance computation must access the data as fast as possible to be efficient. Different types of memory, e.g., secondary storage, main memory, GPU device memory, and GPU registers differ in latency times to access data and available capacity (Cheng, Grossman & McKercher 2015). For example, billions of 3D points can be stored on secondary storage, but access times are typically in the range of milliseconds. In contrast to that, GPU device registers can store only a view hundred points, but provide access within a few CPU clock cycles (Lee *et al.* 2012). In order to maximize the potential of available computing resources, it is necessary to perform a subdivision and partitioning of the data to perform the computation in memory with low capacity but adequate access times.

The partitioning of the data is performed using the multi-data octree and is illustrated in Figure 4.3. The multi-data octree is divided into volumes that contain a subset of data that fits into available main memory. These volumes referred to as *in-memory chunks* and

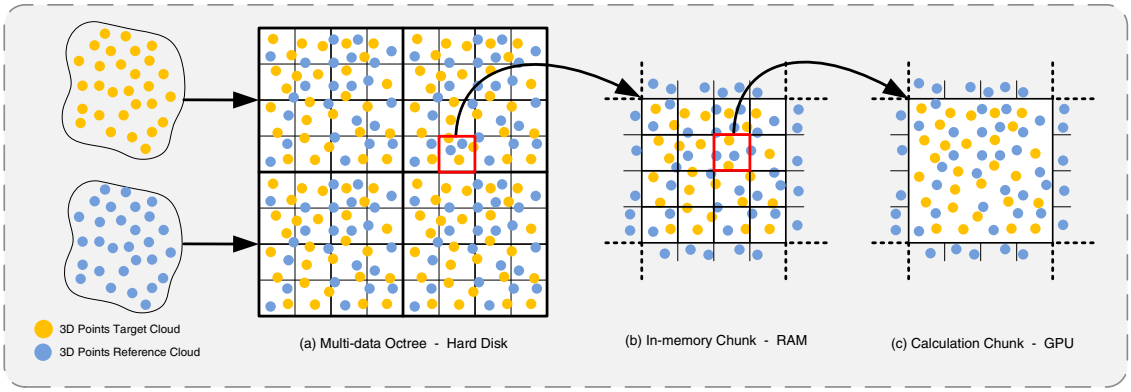


Figure 4.3: *(a) Illustration of the memory hierarchy of the multi-data octree on the hard disk. (b) In-memory chunk that is resident in main memory. (c) Calculation chunk that is used for the computation on the GPU.*

their spatial size is derived from the size of the multi-data octree nodes (Figure 4.3 (a)). All in-memory chunks together cover the overall volume of the multi-data octree. Special cases occur for target points close to boundaries of in-memory chunks, because the closest reference point could be located in an adjacent in-memory chunk. To ensure a correct computation, the volume of the in-memory chunk is extended to include all points from the reference cloud within a defined margin around the volume. This is illustrated in Figure 4.3 (b) for an in-memory chunk transferred from the multi-data octree to main memory.

In-memory chunks are further subdivided into small volumes called *calculation chunks*. These chunks are used as cache to avoid frequent requests to the multi-data octree, which are required to determine the reference points for a given target point. Similar to in-memory chunks, calculation chunks are also extended to guarantee a correct computation (Figure 4.3 (c)). The size of the boundary extension depends on the application requirements and the point density. Calculation chunks contain all necessary data for self-contained processing and enable, in particular, parallel processing. The size of the calculation chunks is adjusted to match the computation algorithm and device.

The following sections describe and compare several implementations to perform change detection in a parallel way using multiple CPU cores and modern GPU technology. A multi-core, a GPGPU, and a CUDA implementation are presented and analyzed regarding performance and hardware issues.

CPU Computation

The single-core CPU implementation processes all in-memory chunks in sequential order. For each in-memory chunk all calculation chunks are determined and the computation is performed sequentially. For a given target point the minimum distance to points of the reference cloud is computed by iterating over all reference points in the calculation chunk. This computation can be performed in parallel using multiple threads to take advantage of all available CPU cores.

GPGPU Computation

Graphics processing units (GPUs) were originally developed for fast processing of graphics primitives in a massively parallel way using multiple processors and dedicated memory directly on the device. Software development for GPUs is typically performed using domain specific programming languages such as the OpenGL shading language (GLSL) (Wolff 2013), HLSL (Varcholik 2014), or Cg (Fernando & Kilgard 2003). These languages have been designed for computer graphics purposes. However, a clever design of algorithms and their adaption to the available graphics primitives, such as vertex buffers and textures, allowing to use GPUs also for other tasks, and has led to the development of a separate branch called general-purpose GPU (GPGPU) computing (Luebke *et al.* 2005).

The implemented GPGPU approach of the distance calculation is based on GLSL. The selected reference points of the calculation chunk are encoded in a texture and transferred as a texture buffer object (TBO) to the GPU. The target points are transferred as a vertex buffer object (VBO) and then rendered using a single rendering pass. The distance computation is performed in the vertex shader by computing distances to all reference points, which are accessed by using texture fetches. Finally, for each target point the minimum distance is stored in a buffer object and the buffer object transferred back to the CPU.

The GLSL implementation does not allow to directly control or configure individual threads or GPU memory. The frequent texture fetches turn out to be the bottleneck of this approach since each element of the VBO needs to access all elements of the TBO. In contrast to the multi-core CPU implementation, the GPGPU implementation requires only 1-2 percent of the computation time due to the highly parallel execution by utilizing hundreds of GPU threads.

CUDA Computation

NVIDIA's Compute Unified Device Architecture (CUDA) is a parallel computing architecture that, in contrast to shading languages, provides a more flexible programming API for general purpose computations on GPUs (Sanders & Kandrot 2010; Cheng, Grossman & McKercher 2015). Interesting key features of CUDA are the ability to read and write arbitrary device memory addresses and faster transfers of data between CPU and GPU. Moreover, CUDA enables access to high-performance on-chip memory, called shared memory, that is shared by a collection of threads. Shared memory enables the implementation of customized user-managed caching strategies, resulting in higher bandwidth and throughput (NVIDIA Corporation 2011).

Modern NVIDIA GPUs consists of several single-instruction multiple-data (SIMD) multiprocessors, each having a large number of cores. An NVIDIA GTX 480, for example, has 15 multiprocessors, each with 32 cores. NVIDIA's currently most powerful state-of-the-art GPU, i.e., the Titan X, has 96 multiprocessors and a total number of 3072 CUDA cores (NVIDIA Corporation 2016). Threads on the GPU are organized into blocks and blocks are organized into a grid. A multiprocessor executes one block at a time. A warp is the set of threads executed in parallel and currently consists of 32 threads. A hardware-based thread scheduler manages scheduling threads across the available multi-processors. The threads are light-weight, enabling the scheduler to perform a zero-cost immediate context switch to another thread if a thread is waiting for memory access (Sanders & Kandrot

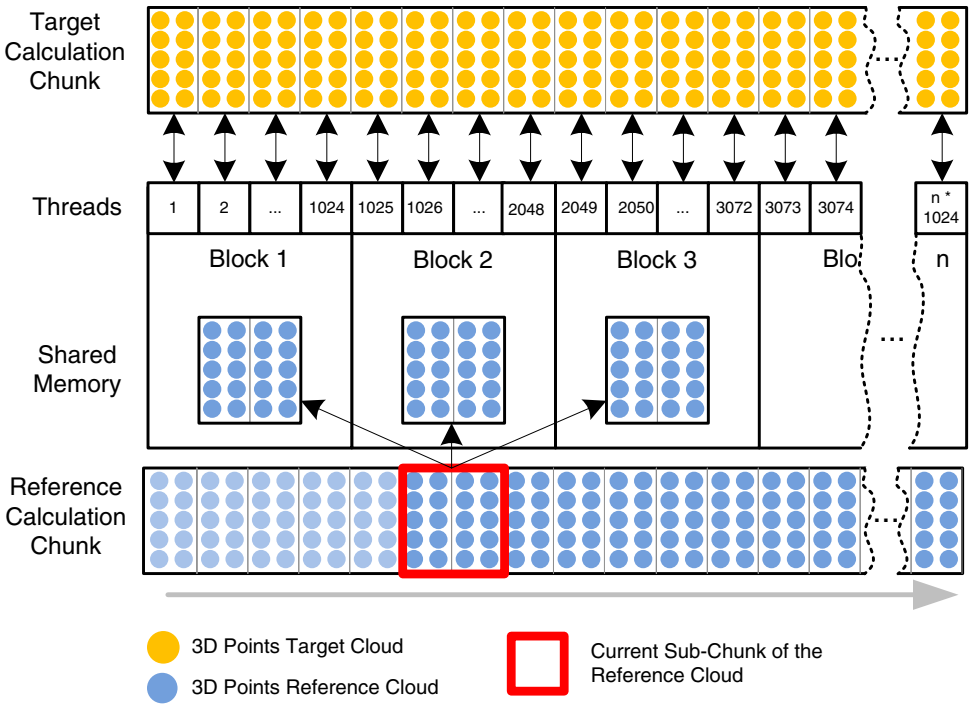


Figure 4.4: *Illustration of the change detection computation for a calculation chunk on the GPU.*

2010). Hence, the number of CUDA cores corresponds with the number of threads that can be executed in a parallel way.

Similar to the GPGPU approach, the CUDA implementation first transfers all target and reference points of the calculation chunk to the device memory of the GPU. For each multiprocessor one block with 1024 threads is scheduled. Target points are divided up between blocks and assigned uniformly to available threads. For all assigned target points, the threads compute the minimum distance to the reference points, by first reading the target point and then reading all reference points from device memory. This calculation schema is comparable to the GPGPU implementation and shows a similar processing performance. However, compared to the time required for the actual computation, device memory access is rather slow. For the performance this is critical, because each thread needs to read all reference points for all assigned target points.

A better approach is to first load portions of the reference points into the low-latency shared memory of the multiprocessor. In contrast to device memory, which has high latency, shared memory provides fast access, but is limited to below 48 kilobytes in size (Farber 2011). The reference points of the computation chunk are therefore, again, subdivided into smaller portions that fit into shared memory. The computation now runs in two phases that are repeated until all data has been processed. First, all threads collectively load a portion of the reference points into shared memory, resulting in the device memory being read only once instead of multiple times. Then, after a synchronization of the threads, each thread computes for all assigned target points the distance to all reference points in shared memory and updates the minimum distance that is kept in local registers. After all reference points have been processed, the computed minimum distance is written to device memory and then transferred to the CPU. This process is illustrated in Figure 4.4.

Processing of calculation chunks can be performed asynchronously to avoid blocking

Table 4.1: *Performance of processing calculation chunks with different sizes measured in seconds.*

#Target Points	#Reference Points	#Distance Calc. in mln	Single CPU	Multi-core CPU	GPGPU	CUDA
20000	20000	400	7.25	1.75	0.20	0.05
40000	40000	1600	29.40	6.86	0.32	0.08
60000	60000	3600	65.57	15.53	0.39	0.13
80000	80000	6400	116.50	27.78	0.54	0.20
100000	100000	10000	182.45	43.55	0.74	0.29
150000	150000	22500	410.77	98.76	1.32	0.59
200000	200000	40000	728.36	178.33	2.35	1.03

of CPU and GPU, since preparation, selection, and transfer of the next calculation chunk can be performed during the computation of the current calculation chunk.

4.3.4 Performance Evaluation

The performance evaluation is performed for different processing levels of the overall workflow from processing a single calculation chunk up to processing a dataset with a view billion points. First, a comparisons of the presented distance computation for a single calculation chunk using the multi-core CPU, GPGPU, and CUDA implementations is performed. Second, an evaluation of the processing of an in-memory chunk using the multi-data octree with different size limits to determine the calculation chunks is presented. Finally, the performance of the overall workflow starting with the preparation of the out-of-core multi-data octree up to the final attributed 3D point cloud using an example dataset of the urban region of a city is evaluated. All tests and measurements were performed on an Intel Xeon CPU with 2.66 GHz and 6 GB main memory. The GPU calculations were performed on a NVIDIA GeForce GTX 480 with 1536 MB device memory.

Performance of Distance Computation

Table 4.1 shows the processing performance for a calculation chunk using the presented CPU and GPU approaches. The evaluation is performed for calculation chunks that differ in their number of target and reference points. Hence, the total number of distance computations increases with a growing amount of data. The single CPU implementation shows as expected the worst performance. Using multiple CPU cores is also still significantly slower than the GPU approaches. The CUDA implementation performs best independently of the size of calculation chunks and is used for further performance evaluation.

The second evaluation scenario is related to processing of an in-memory chunk using the CUDA implementation and shown in Table 4.2. The used in-memory chunk contains 5 million target and reference points and is organized in the multi-data octree with a leaf capacity of 256 point for each dataset. A larger capacity increases the number of distance computations for calculation chunks, because more reference points are selected from intersecting leaf nodes for the computation. A lower capacity would result in more effort to evaluate and load the tree structure. The multi-data octree is used to select the calculation chunks for the distance calculation and enables to control the size and total number of target and reference points in the calculation chunks. Smaller calculation chunks

Table 4.2: *Performance of the CUDA implementation for different sized calculation chunks in seconds.*

Limit Target pts	#Calc. Patches	Mean Target pts.	Mean Ref. pts.	Calc time in ms	Select. time in ms	Total time in s
10000	1821	2745	29132	21352	2148	23.50
20000	1056	4734	36120	16461	1589	18.05
30000	547	9140	43595	12636	1392	14.03
40000	327	15290	49810	11772	794	12.57
50000	302	16556	51047	11754	765	12.52
60000	272	18382	54877	12140	700	12.84
70000	242	20664	59092	12687	690	13.38
80000	206	24271	64994	13439	686	14.13
100000	142	35211	82154	15396	622	16.02
150000	65	76923	136592	19808	540	20.35

make the selection and preparation of calculation chunks computationally more expensive, but would reduce the total number of distance computations for all calculation chunks. The optimal calculation chunk size depends on the processing schema and the available hardware capabilities. For the used system the CUDA implementation performs best using calculation chunks with a size between 40.000 and 60.000 target points.

Performance of Case Study

The performance is evaluated with a real-world dataset as case study. Two 3D point clouds of the urban area of the city of Frankfurt am Main are used to evaluate the performance of the overall change detection workflow. The target cloud *Frankfurt2011* is an airborne scan of the year 2009 with 7.1 billion points. The reference cloud *Frankfurt2005* was captured in the year 2005 and contains 1.9 billion points. Both scans cover almost the same urban area but differ in their sampling density.

The overall time to attribute the target point cloud can be divided into subprocesses to evaluate the different time demands as illustrated in Table 4.3. The process to create the multi-data octree takes the most time, due to the frequent access of secondary storage to organize the data and prepare the multi-data octree. The time to load in-memory chunks and to determine the calculation chunks is rather negligible, because of the fast access to the data provided by the multi-data octree. The distance computation for all calculation chunks takes about 10.5 hours on the GPU and is significantly faster than a CPU implementation that would require several months for the computation. The overall change detection takes about 35 hours and is a reasonable preprocessing time for two datasets with a total amount of 9 billion points and a size of 135 gigabytes. The subprocess to create the multi-data octree has optimization potential using multiple storage systems and parallel processing. This, however, is beyond the scope of this work. Moreover, the performance of the distance computation can be increased by distributing the computation over multiple GPUs.

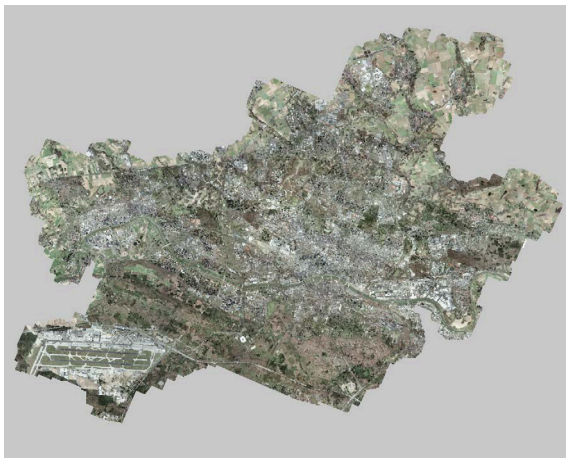
Table 4.3: *Performance of the overall change detection workflow in minutes.*

Task	Time	Percent
Create multi-data octree	1260	61.1%
Load In-memory chunks	114	5.5%
Determine calculation chunks	63	3.1%
Distance computation on GPU	624	30.3%
Overall change detection	**2061**	**100%**

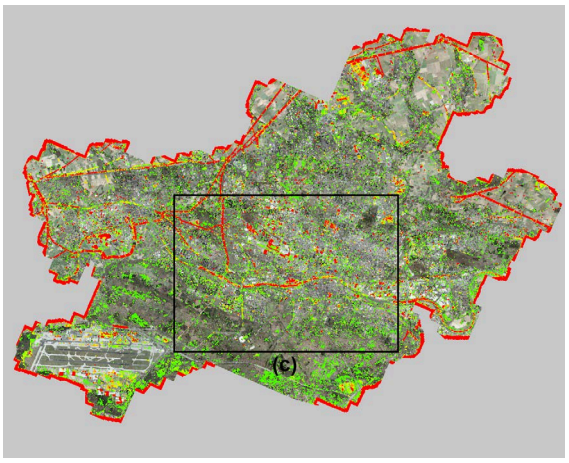
4.4 Results

This section presents change detection results for multi-temporal 3D point clouds of different cities. Results of the distance computations are mapped to color attributes and rendered with a point-based rendering technique. The color mapping is configurable to enable adapting the visualization depending on application and analysis task. Figure 4.5 shows the processing and rendering results of the entire dataset of Frankfurt am Main from different points of view (3D point cloud properties are listed in Table 2.2). The entire dataset with color information, mapped from aerial images, without distance information is shown in Figure 4.5 (a) (Richter & Döllner 2011b). Results are shown in Figure 4.5 (b)-(d). All points indicating a defined degree of change are highlighted with a color gradient from green to red depending on the degree of change. Points with a distance value below 1 meter are classified as points that have not changed and are rendered with colors from aerial images to enable a better perception of the 3D point cloud structure. The detailed view in Figure 4.5 (d) shows an area with buildings and sites that are clearly detected as changes. Another example is illustrated in Figure 4.6 and shows urban changes for the city of Bournemouth in the UK.

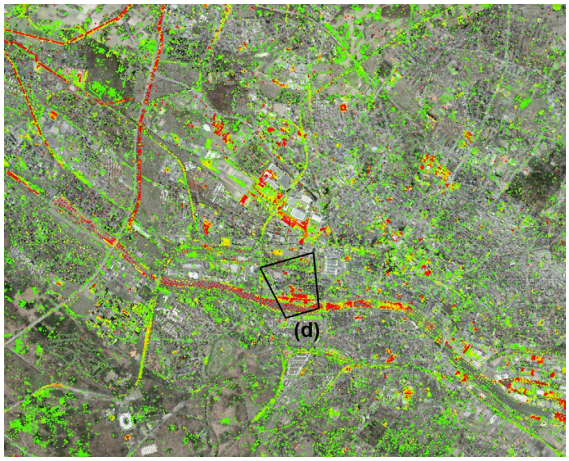
Swapping of target and reference cloud enables additional analysis and interpretation of the data, illustrated in Figure 4.7, where a construction site was captured. The left column shows a target cloud *Frankfurt2009*, attributed with a reference cloud *Frankfurt2005*. The right column shows change detection results using the dataset *Frankfurt2005* as target cloud and the dataset *Frankfurt2009* as reference cloud. In the upper row the 3D point clouds are attributed with color attributes from aerial images, whereas in the middle row a black-to-white color gradient is used to visualize points based on their height. This is helpful to recognize even small structures, e.g., cars and vegetation that are more difficult to recognize in the 3D point cloud colored by aerial images. In the bottom row change detection results are shown. Figure 4.7 (e) shows a construction site and a new building whereas Figure 4.7 (f) shows the former building structures that were removed. Figure 4.8 shows change detection results for the city of Johannesburg captured in 2006 and 2012 using *Johannesburg2006* and *Johannesburg2012*. The reconstructed stadium for the soccer world cup is shown in Figure 4.8 (c).



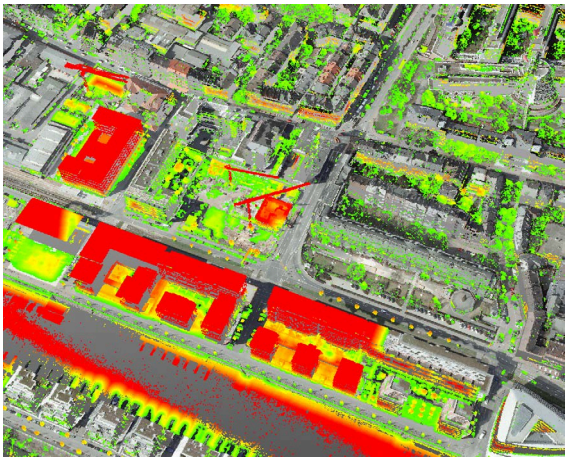
(a) *3D point cloud with colors taken from aerial images.*



(b) *3D point cloud showing detected changes.*

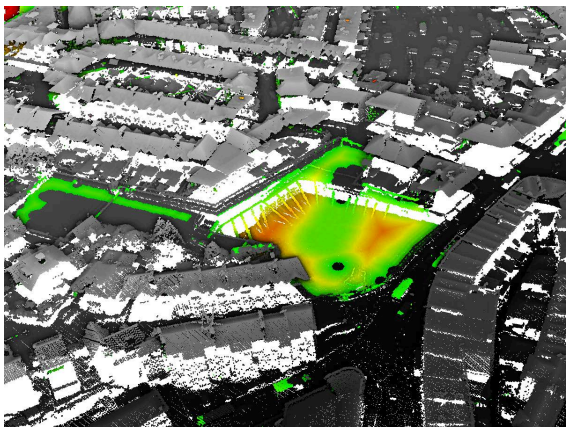


(c) *Detailed view of upper right.*

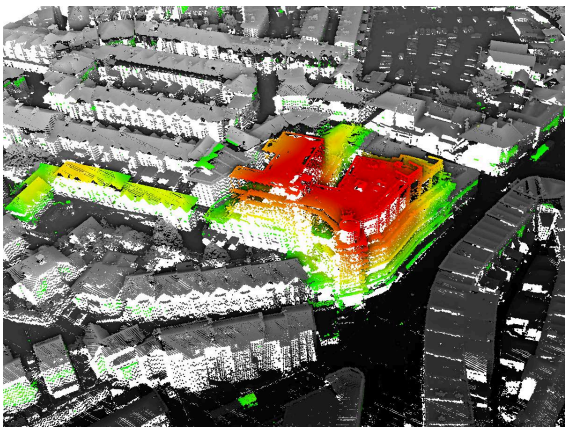


(d) *Detailed view of lower left.*

Figure 4.5: *Change detection results of a 3D point cloud with 7.1 billion points of the urban area of Frankfurt. (a) Visualized with color information. (b) Visualized with color and distance information. (c)-(d) Two closeup views of (b). A color gradient from green (1m) to red (>3m) indicates the degree of change.*

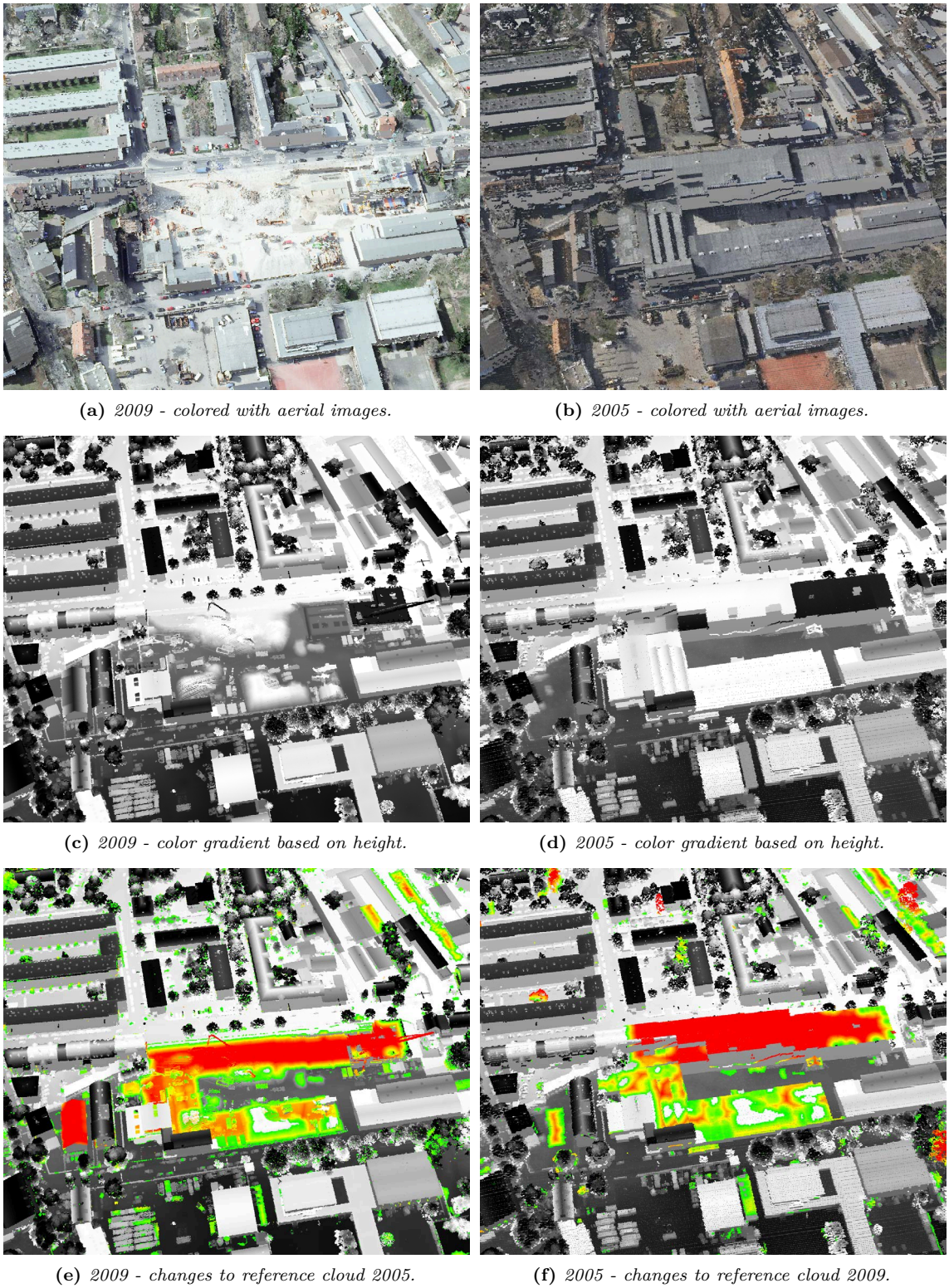


(a) *2006 - changes to 3D point cloud 2008.*



(b) *2008 - changes to 3D point cloud 2006.*

Figure 4.6: *Change detection results for the city of Bournemouth for 3D point clouds from 2006 and 2008. (a) 3D point cloud of a construction site in 2006. (b) 3D point cloud of the final building in 2008.*



(a) *2009 - colored with aerial images.*

(b) *2005 - colored with aerial images.*

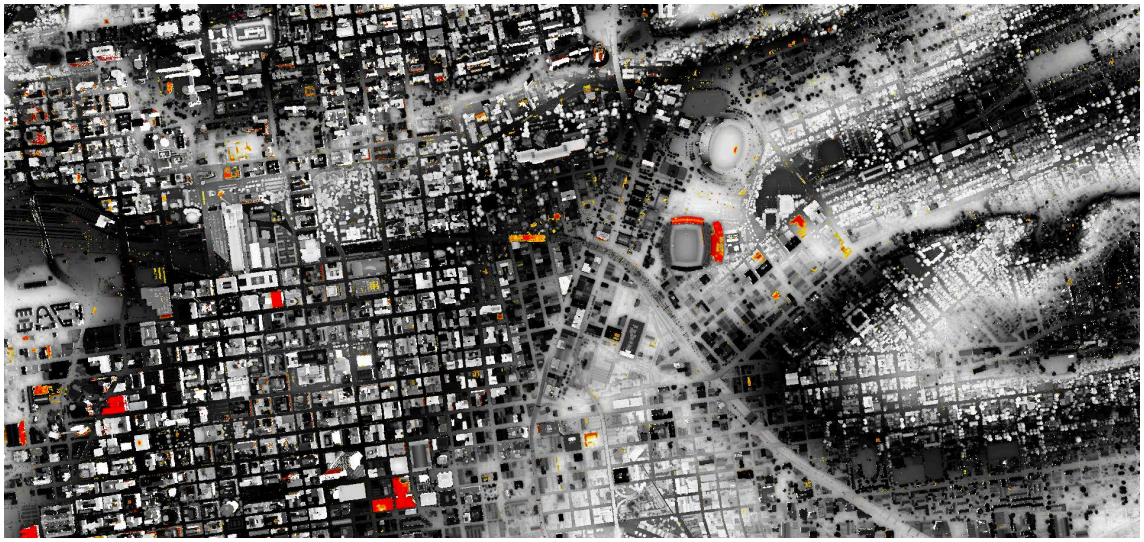
(c) *2009 - color gradient based on height.*

(d) *2005 - color gradient based on height.*

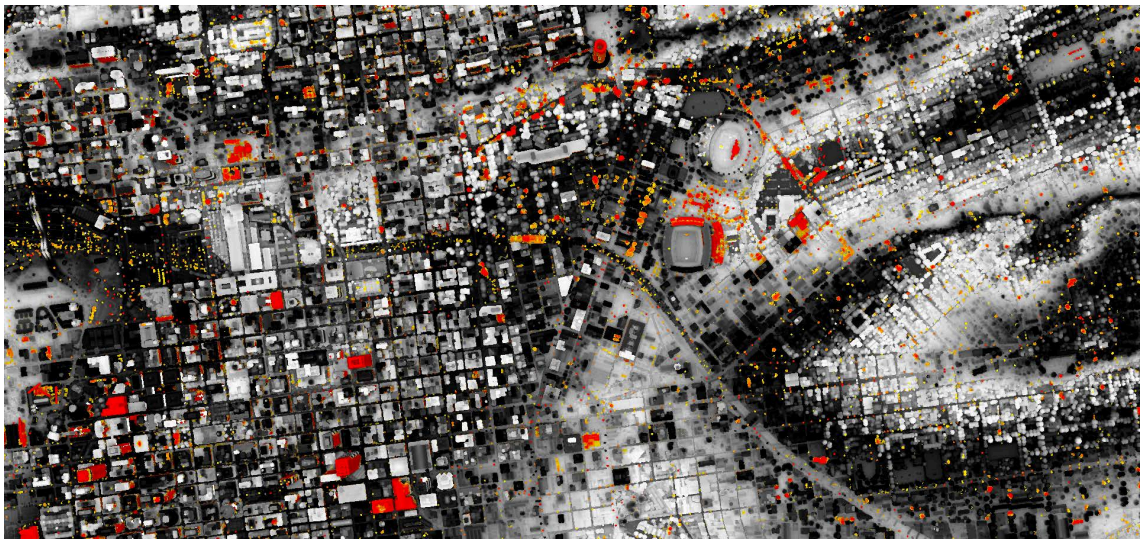
(e) *2009 - changes to reference cloud 2005.*

(f) *2005 - changes to reference cloud 2009.*

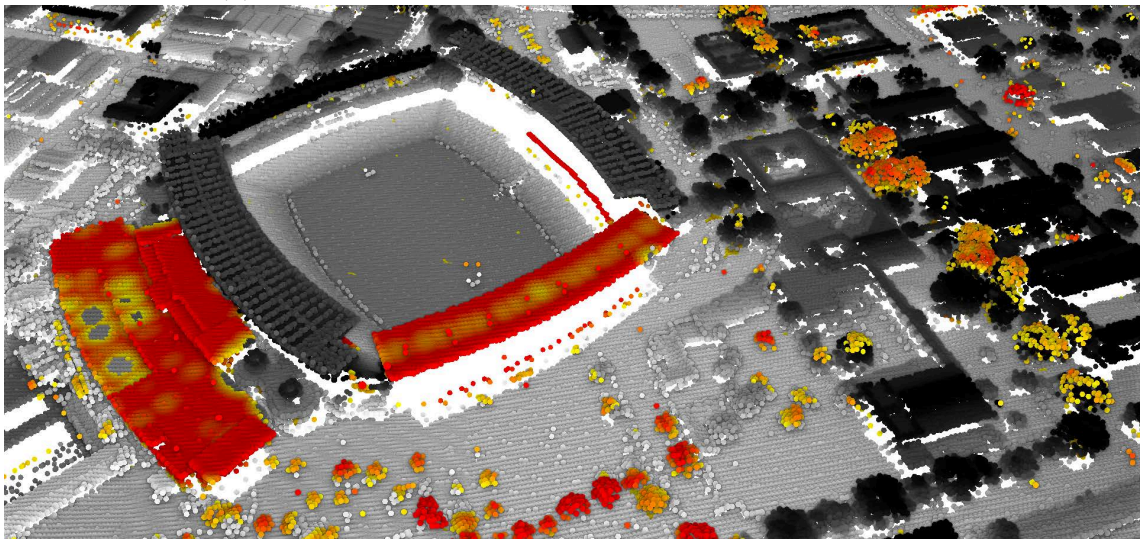
Figure 4.7: *3D point clouds and change detection results for scans Frankfurt2005 and Frankfurt2009. The first row shows the 3D point cloud with color information from aerial images, the second row uses a color gradient based on the 3D point height and the third row shows the degree of change with a color gradient from green to red based on distance values above 1 meter.*



(a) *Johannesburg2006 - changes to 3D point cloud Johannesburg2012.*



(b) *Johannesburg2012 - changes to 3D point cloud Johannesburg2006.*



(c) *Detailed view 2012 - changes to 3D point cloud 2006.*

Figure 4.8: *3D point clouds and change detection results for the city of Johannesburg.*

4.5 Categorization of Changes

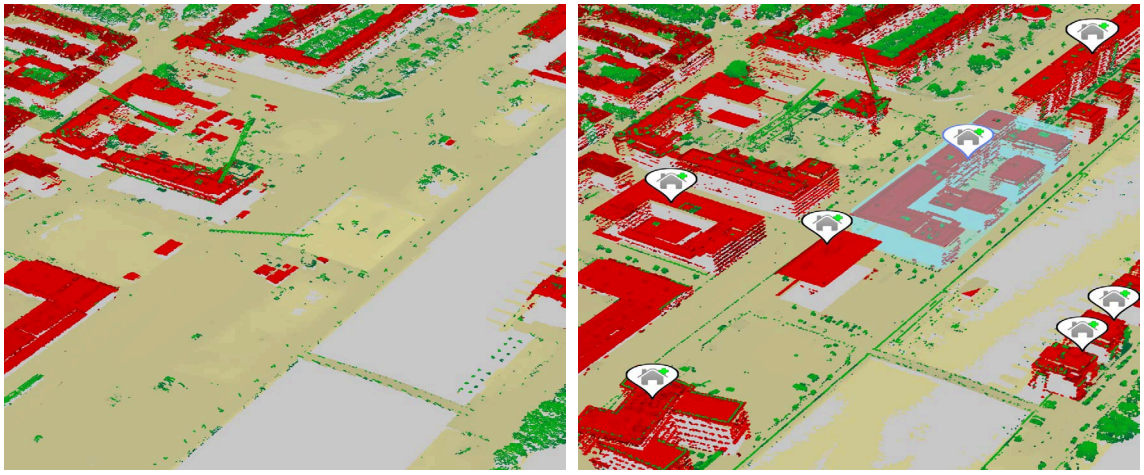
The change detection approach presented in Section 4.3 can be used to estimate the degree of change for each point of a captured surface. However, a categorization of changes is not performed because semantics information, such as object class information, is not considered. This section presents an approach for aerial 3D point clouds to determine the type of change and to estimate quantity parameters by taking into account object class information. Changes for surface categories such as ground, building, and vegetation can be categorized as follows and typically occur due to the following reasons:

- Ground-to-building - Construction of new buildings
- Ground-to-vegetation - Growing or planting of vegetation
- Ground-to-ground - Elevated or lowered ground due to natural events
- Building-to-vegetation - Demolition of constructions and growing of vegetation
- Building-to-ground - Demolition of constructions
- Building-to-building - Structural changes or continuation of constructions
- Vegetation-to-ground - Removed vegetation
- Vegetation-to-building - Construction of new buildings
- Vegetation-to-vegetation - Grown or reduced vegetation

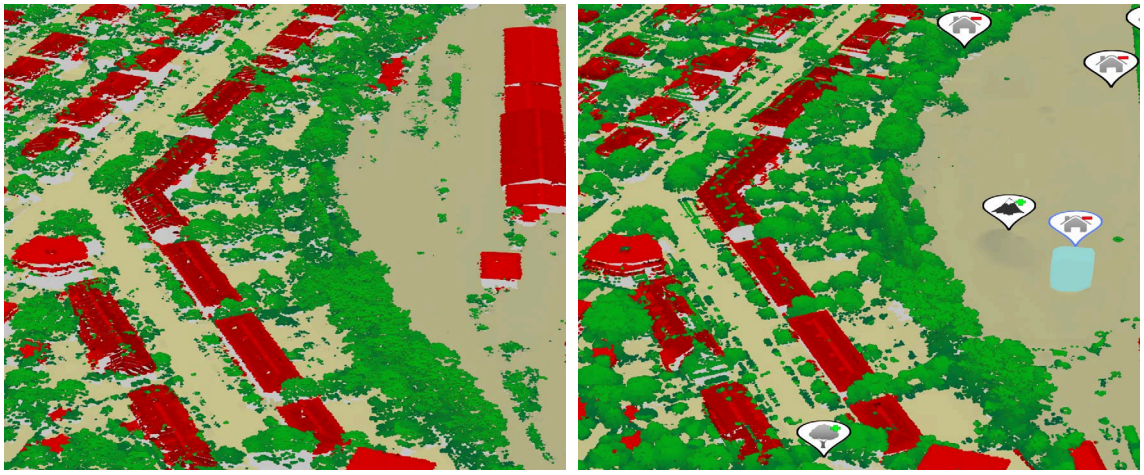
Figure 4.9 illustrates detected change categories for the city of Frankfurt. The workflow to categorize changes can be divided into the following steps:

1. Object-class segmentation for target and reference data.
2. Change detection by taking into account object-class information.
3. Object-class aware segmentation of changed regions.
4. Estimation and calculation of change parameters per segment.

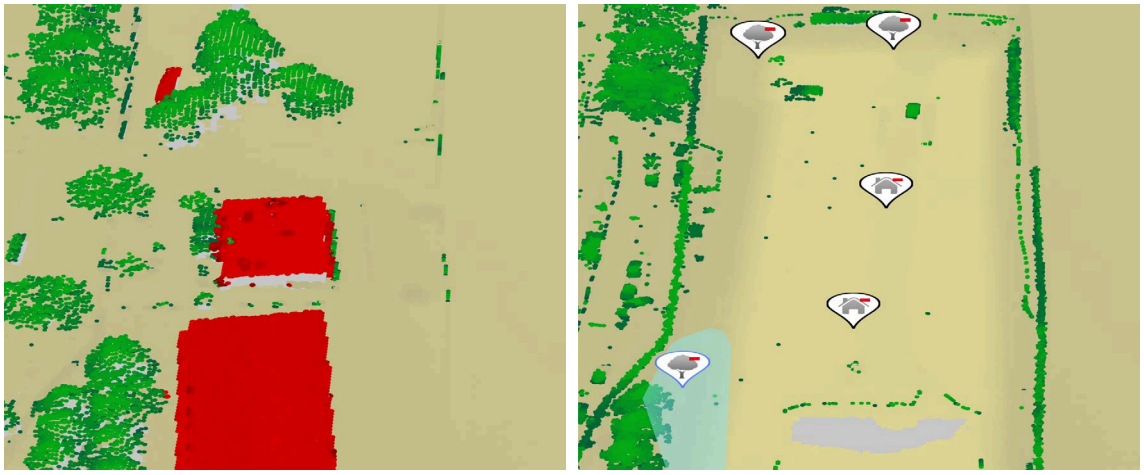
First, target and reference cloud need to be classified into ground, building, and vegetation points as described in Chapter 3. Second, the change detection approach, as described in Section 4.3, is used to determine the degree of change for each point of the target cloud. This approach is extended to determine in addition to the distance also object-class information of the reference cloud in the local proximity of each point from the target cloud. This information is used to assign each changed point to a category of change (e.g., ground-to-building). A limitation of this approach is that it does not work reliable for edge regions of changed structures that are located close to unchanged structures. The reason is that the closest reference point is inappropriate because it could belong to an unchanged structure and not to the removed part of the target cloud. To handle these scenarios, a more specific reference point query can be used. The reference query volume is bounded by an unbounded vertical circular cylinder with a defined radius (e.g., 1 meter) that depends on the point density of the data. Hence, changes are classified in an



(a) *Frankfurt2005.* **(b)** *Frankfurt2009 with categorized changes.*



(c) *Frankfurt2005.* **(d)** *Frankfurt2009 with categorized changes.*



(e) *Frankfurt2005.* **(f)** *Frankfurt2009 with categorized changes.*

Figure 4.9: *3D point clouds and categorization of change detection results of Frankfurt2005 and Frankfurt2009. Items indicate the change category and bounding boxes show the expansion of a selected item.*

appropriate way even for high structures (e.g., buildings). The third step is a segmentation of changed points that can be configured depending on the degree (e.g., small or large) and type (e.g., building-to-ground) of changes. In the last step, these segments are used to estimate additional parameters such as changed area, extend, and volume that are essential to evaluate, filter, and explore the 3D point clouds with change information.

4.5.1 Change Detection for Incremental Point Cloud Updates

In the near future, automatic, ubiquitous, and frequent data acquisition, e.g., with mobile mapping cars, will result in 3D point clouds with a high redundancy (Schachtschneider, Schlichting & Brenner 2017). It would not be possible to store highly redundant data for the same surface with straightforward approaches. The presented change detection approach computes per-point attributes to estimate the degree of change for each input point (i.e., it supports the decision whether a point is already represented by another points). Hence, it can be used to compare existing data (e.g., in a point cloud database) with new data to perform incremental updates.

The point-to-point comparison that computes the Euclidean distance between each input point and the nearest point of the reference data works well for 3D point clouds with similar point densities to detect large-scale changes (e.g., new constructions). This can be done with a global threshold, e.g., all points with distance values above 2 meter are declared as new or changed structures (e.g., for urban areas, landscapes, buildings, and constructions (Matikainen 2004; Barber, Holland & Mills 2008; Butkiewicz *et al.* 2008)).

However, the *Euclidean distance* as a metric for changes becomes unreliable for a non-uniform surface sampling and different point densities (e.g., caused by varying capturing devices and distances between capturing device and scanned surface). Moreover, the detection of small-scale changes (e.g., construction and building deformations (Schneider 2006; Monserrat & Crosetto 2008; Kang & Lu 2011)) is difficult for datasets with varying resolutions.

A solution is a *point-to-splats* approach that takes into account the surface structure of the reference point cloud. Splats are used as reference data that are derived from the local point density and proximity. They can be computed based on the point position, surface normal (required for orientation), and locale density (required for splat size) to approximate the surface of the scanned target. Hence, a distance to these splats can be used as change metric that is tolerant against non-uniform surface sampling and varying point densities. Figure 4.10 illustrates the point-to-point and point-to-splat approach. A target 3D point cloud (P_{T2}) is attributed with change attributes based on a reference 3D point cloud (P_{T1}). The distance threshold can be determined adaptively based on the available data, density, precision, and application. Restrictions of the point-to-splats approach are scattered parts of 3D point cloud (e.g., vegetation) because a surface approximation with splats cannot be performed reliable.

The presented point-to-splat change detection approach constitutes an important component for incremental point cloud updates. It is essential to reduce system workloads for further processing and analysis tasks to control storage costs. The overall number of points can be reduced by avoiding the need to store points for surfaces or surface parts that are already present in the data.

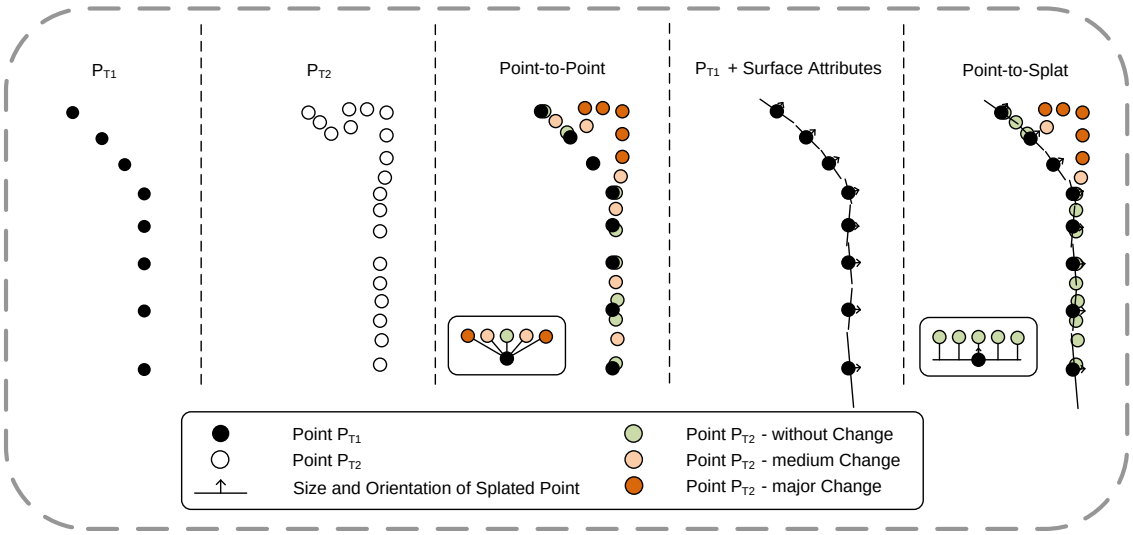


Figure 4.10: *Illustration of a point-to-point and point-to-splat change detection approach to update a database.* P_{T1} *is the 3D point cloud in the database and is used to compute change attributes for the input data* P_{T2}.

4.5.2 Potential of Multi-temporal 3D Point Clouds to Detect Static Structures and Non-static Entities

A redundant capturing of surfaces and structures can be used to update 3D point clouds incrementally, as explained in the Section 4.5.1. In addition, it has a lot of potential for the identification of static structures and non-static entities in 3D point clouds. Examples of static structures are buildings and ground surfaces that can be typically found in a large number of scans of the same spatial area. In contrast, non-static entities, such as vehicles and pedestrians, can only be found in individual scans at the same position. The separation of static and non-static entities (e.g., in road traffic scenarios) is well studied for robotic application and mobile mapping applications, e.g., by analyzing principal component analysis (PCA) features (Li *et al.* 2012a). The detection of non-static entities in aerial 3D point clouds can be performed by comparing overlapping flight strips (Höfle, Hollaus & Hagenauer 2012). In general, these approaches use data that is captured in a short period of time. It can be assumed that a redundant capturing (e.g., at more than 20 timestamps) for a longer time (e.g., a few weeks, months, or years) enables the possibility of separating static structures and non-static entities based on the frequency of points that have been captured for a surface.

A simple approach uses a voxel grid to store the frequency of points for a spatial area. This supports the determination of points that belong to static structures that were captured with a defined redundancy (e.g., at more than 5 different timestamps). In addition, points that can only be found once in a redundantly captured area can be identified as non-static entities. One challenge for this approach are occluded areas (e.g., vehicles that occlude façades). The detection of occluded areas is addressed by Girardeau-Montaut et al. (2005) and could be resolved with visibility maps. Another solution for occluded structures is a segment-based approach that takes into account segments of locally connected points with a similar characteristic. If parts of the segment are occluded but most points that belong to the segment can be identified as static, then all segment points can be declared

as static points. The overall quality of this approach would increase with the number of 3D point clouds from different timestamps. However, this is an open research topic and need to be investigated in future work.

4.6 Discussion and Future Work

This chapter presents and discusses concepts and implementations to detect changes in 3D point clouds taken at different points in time. Existing frameworks for point cloud comparison (Mémoli & Sapiro 2004; Girardeau-Montaut *et al.* 2005) work well for small 3D point clouds (e.g., with a few million points) but cannot be applied to massive 3D point clouds. The challenge of massive 3D point clouds is addressed and solved with a specialized out-of-core data structure that enables fast and efficient access to multiple 3D point clouds for a parallel processing. Several parallel processing schemas have been investigated, with the GPU-based approaches turning out to be clearly superior. In particular, the evaluation shows that the CUDA implementation performs best due to its ability to leverage high-performance on-chip memory in a flexible way. Hence, the GPU-based presented approaches can cope with massive 3D point clouds. Among potential applications are:

- **Visualization and inspection tools**, using change detection results as input for real-time visualization systems that enable exploring and analyzing temporal changes of highly accurate 3D point clouds of arbitrary size.
- **Categorization of changes** is possible by taking into account classification results. It enables to separate different types of temporal changes as requested by a growing number of applications that can take advantage of the full resolution and potential of the 3D point clouds.
- **Incremental updates** are important for applications that demand for up-to-date data. An adaptive change detection for different scales and data characteristics is required and can be implemented with a point-to-splat approach.
- **Separation of static and non-static entities** is a further example for semantics information that can be derived from multi-temporal 3D point clouds and used to provide more precise, i.e., filtered data to domain specific applications.
- **Up-to-date traffic networks** are important for autonomous driving and navigation tasks. A detection and monitoring of temporary network modifications is possible with GPS trajectories (Kuntzsch, Sester & Brenner 2016) and can be supplemented with change detection results.

Future work could focus on change detection approaches for 3D point cloud streams. These steams are generated by camera systems, i.e., stereo cameras with depth images as output, or LiDAR systems as a constant stream of data. A permanent change detection would be required to handle the massive, redundant amount of data. Technical and algorithmic challenges are a spatial data structure that can be created, extended, filled, and requested in real-time. Multiple streaming devices and a distributed system infrastructure for storage, processing, and access are required. Applications also include autonomous driving and navigation, using dynamic point cloud as models of the environment.

Chapter 5

3D Point Cloud Rendering

"Point-based rendering has been shown to offer the potential to outperform traditional triangle based rendering both in speed and visual quality when it comes to processing highly complex models."

— Botsch & Kobbelt 2003

This chapter introduces concepts and techniques for the visualization, presentation, and exploration of 3D point clouds. In particular, it presents implementations of real-time rendering algorithms and interaction techniques to make an interactive visualization and exploration of 3D point clouds feasible. Point-based rendering techniques use points as display primitives to avoid the generation of traditional polygon-based models that use triangles as display primitives. Out-of-core strategies and level-of-detail data structures allow us to handle even datasets that exceed available memory resources by several orders of magnitude. The presented point-based rendering techniques are adaptive and can be applied to 3D point clouds of any origin, e.g., from airborne, mobile, and terrestrial scans. The rendering techniques take into account per-point attributes such as color, object class, and change information and can be configured based on system, application, and user requirements.

5.1 Motivation

Applications of 3D point clouds include large-scale models of 3D environments and virtual 3D city models that require time-intensive and costly workflows for model creation and geometry updates. Point-based rendering systems proved an alternative to represent 3D models and environments by dense 3D point clouds, which significantly minimizes the effort for preparation and updates (Nebiker, Bleisch & Christen 2010).

Visualization and presentation techniques for massive 3D point clouds are essential for the understanding and communication of spatial information, as well as analysis and simulation results (Kreylos, Bawden & Kellogg 2008; Bettio *et al.* 2009; Kim & Medioni 2010). Point-based rendering techniques can cope with massive 3D point clouds and enable an interactive visualization and exploration of the data (Wimmer & Scheiblauer 2006; Richter & Döllner 2010b). These techniques can be applied to 3D point clouds, thereby enabling different rendering styles (e.g., photorealistic, non-photorealistic, or based on point attributes (Botsch *et al.* 2005)).

Thematic and semantics information as well as analysis and simulation results can be provided as per-point attributes to avoid the generation of polygon-based 3D models,

which become outdated when the 3D point cloud partially changes (Richter, Kyprianidis & Döllner 2013). This becomes particularly important when the 3D point cloud permanently changes and up-to-date models and analysis results are required. Visualization of 3D point clouds can be used to:

- Control, examine, and evaluate the quality of the captured data
- Understand and communicate spatial information
- Communicate analysis, simulation, and processing results
- Edit, filter, and correct the data
- Avoid costly generation of 3D models in case of massive 3D point clouds
- Represent 3D GeoVEs as dense 3D point clouds to avoid time-consuming generation of 3D models

The design and implementation of point-based rendering techniques for real-time visualization of 3D point clouds requires to address the following main challenges:

- Out-of-core strategies to handle any sized 3D point clouds
- Guarantee high frame rates during user interaction
- High rendering quality even on standard consumer hardware
- Adaptive consideration of user interaction, memory resources, and hardware performance
- Adaptive regarding semantics information to apply individual rendering techniques or render only selected subsets (e.g., based on object-class information)

5.2 Related Work

The first rendering techniques for 3D point clouds have been presented in Surfels (Pfister *et al.* 2000) and QSplat (Rusinkiewicz & Levoy 2000). Surfels is based on an octree data structure to store points. QSplat is based on a k-d tree to store a bounding sphere hierarchy that provides position, normal, and color information for the rendering process. Sainz, Pajarola & Lario (2004) and Gross & Pfister (2007) give a good overview about point-based rendering techniques and systems.

Several point-based rendering techniques aim for a photorealistic and, thus, solid visualization of 3D point clouds without holes in the surface (Sibbing *et al.* 2013; Yu & Turk 2013). These techniques commonly represent points as splats, i.e., oriented flat disks (Zwicker *et al.* 2001; Botsch & Kobbelt 2003; Botsch *et al.* 2005), spheres, or particles. To visualize closed surfaces, an adequate size and orientation based on the local point density need to be applied to each point (Kim *et al.* 2012). These attributes can be simply determined from polygon-based models that are used to generate 3D point clouds (Pfister *et al.* 2000). However, these techniques are difficult to apply for irregular and unstructured 3D point clouds, such as LiDAR scans, because of varying point densities, e.g., on horizontal and vertical structures, as well as fuzzy and planar areas. In addition, it

is difficult to combine these techniques with out-of-core rendering approaches that take into account density attributes. Reasons are changing point density due to LoD-techniques that adaptively select and render points.

The focus of this work are 3D point clouds from real-world scenes and surfaces with varying densities and irregular surface characteristics, captured with different scanning technologies and systems. As a consequence, attributes required to improve the rendering quality need to be calculated in a preprocessing step (Wu & Kobbelt 2004) or on a per-frame basis as proposed by Preiner, Jeschke & Wimmer (2012). Surface normal and density information can be computed by analyzing the local point proximity (Dey, Li & Sun 2005). Color information can be added by using multiple images (Abdelhafiz & Niemeier 2006) or projected videos during the rendering process (Zhao, Nister & Hsu 2005).

Dachsbacher et al. (2003) have introduced sequential point trees to efficiently render 3D point clouds by evaluating the data structure on the GPU. This approach is limited by the available GPU memory and therefore not suitable for 3D point clouds that do not fit into GPU memory.

Non-photorealistic rendering techniques for 3D point clouds have been proposed by Goesele *et al.* (2010) and Xu *et al.* (2004). Olson *et al.* (2011) show how the complete set of silhouette points of a surface can be calculated instant. However, that information comes with the cost of an additional preprocessing step.

XSplat (Pajarola, Sainz & Lario 2005) is a rendering system based on Dachsbacher et al. (2003) with out-of-core strategies to render massive 3D point clouds. It concentrates on display quality and has been designed to render sampled models with continuous surfaces. Further out-of-core rendering systems for 3D point clouds have been presented in Gobbetti & Marton (2004b), Wimmer & Scheiblauer (2006), and Goswami *et al.* (2013). These systems use LoD data structures that aggregate or generalize points for a spatial area to create a defined LoD. Wimmer & Scheiblauer (2006) have presented a rendering approach that does not assume any sampling density or availability of normals for rendering of LiDAR data. The rendering of unprocessed 3D point clouds is based on memory-optimized sequential point trees and has out-of-core strategies to handle massive 3D point clouds. LoD data structures that aggregate or generalize points for a spatial area are not applicable for the presented purpose because it could be required to separate points according to their semantics, e.g., object-class information, at any time during rendering to apply individual rendering techniques as well as render only selected subsets of the data.

Pauly *et al.* (2003) have introduced a modeling framework for point-based geometries. A data structure to enable surface modifications at different scales was introduced by Pauly, Kobbelt & Gross (2006). Rendering systems with out-of-core data structures can be also combined with features to edit 3D point clouds (Zwicker *et al.* 2002; Wand *et al.* 2007; Kreylos, Bawden & Kellogg 2008; Scheiblauer, Zimmermann & Wimmer 2009). The data structures are designed and optimized to edit, insert, and remove points during the rendering process. Editing of 3D point clouds during the rendering process is beyond the scope of this thesis. It is not implemented and supported to keep the spatial data structure compact.

The presented techniques do not use the sequential point tree approach introduced by Dachsbacher et al. (2003) and adapted by the Instant Points rendering system of Wimmer & Scheiblauer (2006). The presented approach uses some basic strategies from QSplat (Rusinkiewicz & Levoy 2000) to perform efficient culling and data chunk selection. The

GPU memory is only used to store points that need to be rendered.

Bettio *et al.* (2009) have introduced a client-server framework to render large point-based 3D models, which can perform different measurements. The used data structure is based on Layered Point Clouds from Gobbetti & Marton (2004b). Boubekeur, Duguet & Schlick (2005) introduce the use of normal-mapped polygons as approximation to reduce the complexity of 3D point clouds.

Some other approaches propose hybrid rendering to increase frame rates by replacing point sampled areas with meshes (Wahl, Guthe & Klein 2005). Rendering of huge triangle-based meshes assumes a simplification of the mesh (Isenburg & Lindstrom 2005). In contrast to massive 3D point clouds connectivity information for meshes must be stored. Most solutions use a progressive mesh representation for storing and transmitting arbitrary triangle meshes (Hoppe 1996). Out-of-core techniques to render huge triangle meshes have been investigated in Gobbetti, Kasik & Yoon (2008), Callieri *et al.* (2008), and Wald, Dietrich & Slusallek (2004). Eggert & Sester (2013) present a visualization approach for mobile mapping data using partially translucent images. This compressed representation is used for a fast data browsing of massive 3D point clouds (Eggert & Schulze 2014).

Streaming solutions for meshes (Isenburg & Lindstrom 2005) and 3D point clouds (Pajarola 2005) enable a fast transfer of data into main memory but are limited for interactive visualizations.

5.3 Data Characteristics

This section describes the data characteristic and per-point attributes that can be used to enhance and adapt the rendering of 3D point clouds. In general, only coordinates (x, y, z) of each point are require. 3D point clouds from different sources need to be transformed into a uniform coordinate system and standardized to a predefined precision (e.g., floating point precision). Additional per-point attributes are optional and can be used to improve the rendering quality and visual appearance. These attributes can be categorized into thematic and structure attributes. Thematic attributes are related to some information that is application specific and relevant for users to understand and recognize structures and relationships. Structure attributes are generally not relevant for the user and used to enhance the visual appearance of the data. All attributes can be mapped during the rendering to color attributes and geometric properties of each point. This enables customizing the visualization depending on the data characteristics and application.

5.3.1 Thematic Attributes

- **Color data**. Color data is typically extracted from images that are ideally captured at the same point in time as the 3D point cloud (Abdelhafiz, Riedel & Niemeier 2005). In general, mobile and terrestrial scanning systems capture image data and are able to export a color value for each point. Color data for aerial 3D point cloud is typically added in a separate processing step. This process is similar to the texturing of 3D models (e.g., 3D city models) and can be performed for massive data sets (Richter & Döllner 2011b). Each point can have multiple color attributes, e.g., from RGB images, infrared images, or thematic maps. RGB color data is generally used for a colorization, e.g., when a photorealistic and natural appearance of the points is

required. Infrared and thematic data can be used to highlight surface categories such as water, vegetation, and non-vegetation.

- **Object-class information**. This attribute denotes to which surface category a point belongs and can be automatically generated for aerial 3D point clouds as described in Section 3. Common object classes are vegetation, building, terrain, and water, which can be derived by analyzing the 3D point cloud topology, i.e., local neighborhood of a point. A more detailed subdivision of terrain points (e.g., infrastructure, land use) or building points (e.g., commercial, residence) can be made by taking into account additional map data (e.g., infrastructure maps) (Richter, Behrens & Döllner 2013). Object-class information for mobile and terrestrial may include city furniture (e.g., cars, traffic lights) that can be identified with approaches presented in (Golovinskiy, Kim & Funkhouser 2009).

- **Distance information**. Distance information indicates the distance to a reference geometry and is typically encoded as a value per point, e.g., floating point number. Common reference geometries are 3D point clouds from a different date indicating changes (Chapter 4.3), digital surface models (e.g., DTMs, DSMs), or 3D models (e.g., digital 3D city models) that need to be compared with the captured data. Hence, it is important to use and visualize these attributes together with the entire 3D point cloud for quality control and evaluation of acquired data.

5.3.2 Structure Attributes

- **Surface normal**. Per-point normals approximate the surface of the local point proximity. They can be computed efficiently by analyzing the local neighborhood of a point (Mitra & Nguyen 2003) and are used to orientate the points primitive according to the represented surface.

- **Horizontality**. This attribute indicates how vertical the surface normal of a point is oriented, i.e., points representing horizontal surfaces (e.g., flat building roofs). It features a higher value for points on vertical surfaces (e.g., building façades) (Zhou & Neumann 2008). The horizontality can be used for a colorization to accentuate detailed object structures such as roof elements.

- **Global height**. This attribute describes the height of a point in relation to all other points that belong to the same object class. Colorizing points based on their global height emphasizes height differences for different objects belonging to the same object class (e.g., trees with different heights).

- **Local height**. The local height describes the height of a point in relation to all point belonging to the same object class in the point proximity. Using local heights for a colorization facilitates to highlight edges and differences in the structure of an object (e.g., roof ridges and smokestacks).

- **Segment information**. Segment information, i.e., IDs stored as per-point attribute, are used to identify and group points belonging to the same object, surface, or category. Segment information is typically a result of a processing or analysis pass and can be used to highlight specific points, e.g., belonging to a single tree or a roof surface.

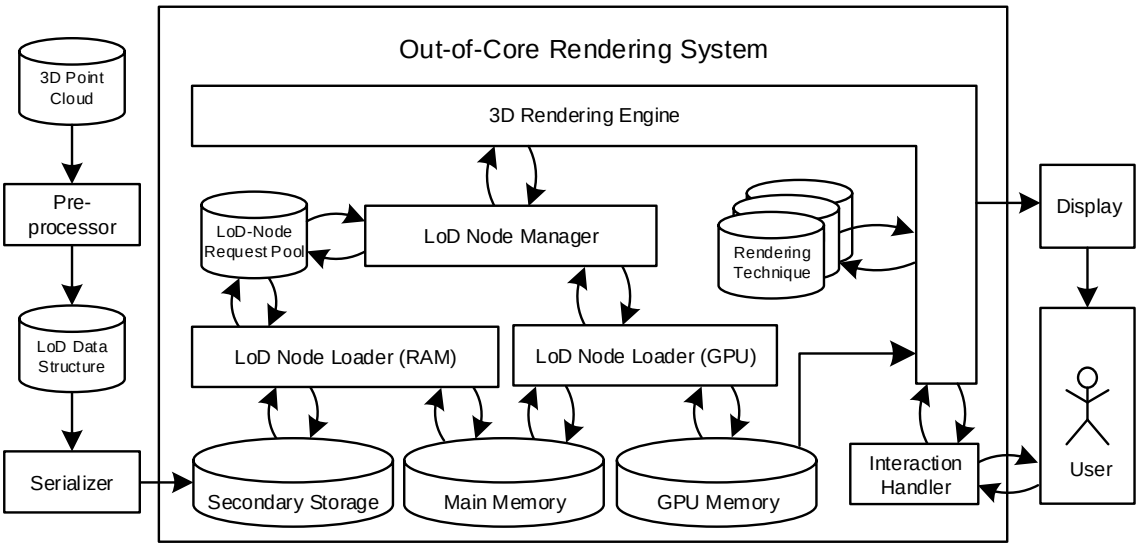


Figure 5.1: *Common architecture of an out-of-core rendering system for 3D point clouds.*

A challenge for rendering massive 3D point clouds at interactive frame rates are limited storage capacities of main memory and GPU device memory. The size of the data affects the performance of the rendering process. Each point of the 3D point cloud is typically given by a three-dimensional floating point vector with a size of 12 bytes and optional thematic or structure attributes, such as color, distance, and normal data.

5.4 Out-of-Core Real-Time Rendering

The interactive visualization of massive 3D point clouds demands for out-of-core rendering techniques that combine level-of-detail (LoD) concepts, spatial data structures, and external memory algorithms to handle and render 3D point clouds that exceed available memory resources and rendering capabilities. The spatial data structure is used to store the data and to determine subsets of a 3D point cloud at an arbitrary LoD at runtime. The preparation and construction is typically performed in a preprocessing step that serializes the data to secondary storage. The rendering algorithm uses this structure to select, update, and remove data chunks in an efficient way to perform an interactive or real-time image synthesis. Figure 5.1 illustrates the common architecture of an out-of-core rendering system for 3D point clouds. 3D point clouds are prepared and converted into a LoD data structure that is stored on secondary storage (Section 5.4.1). The 3D rendering engine controls a LoD node manager to schedule required data chunks, i.e., LoD nodes, for the rendering depending on the user interaction (Section 5.4.3). Point-based rendering techniques are applied to the data to provide an interactive visualization (Section 5.5).

5.4.1 Multi-Resolution Data Structure

Most spatial data structures use k-d tree, quadtree, or octree derivations to arrange 3D point clouds in a preprocessing step (Rusinkiewicz & Levoy 2000; Gobbetti & Marton 2004b; Wimmer & Scheiblauer 2006; Richter & Döllner 2010b; Goswami *et al.* 2013). The construction of quadtrees and octrees can be performed faster in contrast to k-d trees because there is no need to sort the points during the construction process. However, the

use of quadtrees and octrees for irregular and sparse distributed data, e.g., airborne laser scans, results in LoD nodes with a varying number of points. This complicates efficient caching and memory swapping mechanisms that would benefit from *equal-sized* data chunks. For that reason, the basic concept of a k-d tree is used and has been extended to fill the following requirements:

- R1 - Efficient storage of 3D point clouds and related per-point attributes.
- R2 - Ability to handle massive 3D point clouds that exceed available memory resources.
- R3 - Storage of LoD chunks to enable an interactive visualization and control of the rendering budget (e.g., 30 frames per second).
- R4 - Adaptive memory management with equal-sized LoD chunks to implement efficient caching and out-of-core strategies.
- R5 - Attribute specific subdivision of the 3D point cloud to enable a selective access and visualization, e.g., only building or vegetation points.

The root node of the data structure represents the overall expansion of the 3D point cloud. Child nodes subdivide the area of their parent node. Nodes with children are inner nodes and nodes without children are named leaf nodes. Each inner node is a generalization of its child nodes. Every node stores a defined number of points and corresponds to a LoD for a spatial area (R3). The root node is the highest abstraction of the data and a leaf nodes provides a maximum of details. Each point is stored only once in the tree to reduce the memory requirements (R1). Thus, all nodes together represent the entire data and are equal to the input 3D point cloud. The capacity of nodes affects the total number of nodes in the tree, which has an impact on main memory requirements, caching mechanisms, and the rendering quality.

This multi-resolution data structure can be extended for an attribute-specific organization of the data. This is important to enable an efficient selection of subsets based on spatial and thematic attributes. Points belonging to different categories (e.g., object classes) can be assigned to different layers of the tree. A layer is characterized as part of the tree structure that contains only points with defined attributes. Common attributes are object class, time, or other semantics information that can be used to divide the data efficiently in addition to the spatial extend (R5). An example is illustrated in Figure 5.2 that shows different layers of the multi-resolution data structure according to object-class information. The spatial subdivision of the data enables to implement a memory management that load parts of the structure from external memory. Thus, it is possible to interactively visualize 3D point clouds of arbitrary size. Practical, the size is limited by the available hardware storage capacity of the hard-disk or network storage (e.g., cloud or NAS system). The visualization of 3D point clouds is implemented by point-based rendering.

5.4.2 Construction

This section describes the construction of the multi-resolution k-d tree with out-of-core concepts to arrange massive 3D point clouds in multiple layers according to per-point attributes. The *layerd, multi-resolution out-of-core k-d tree* is named data structure in the following.

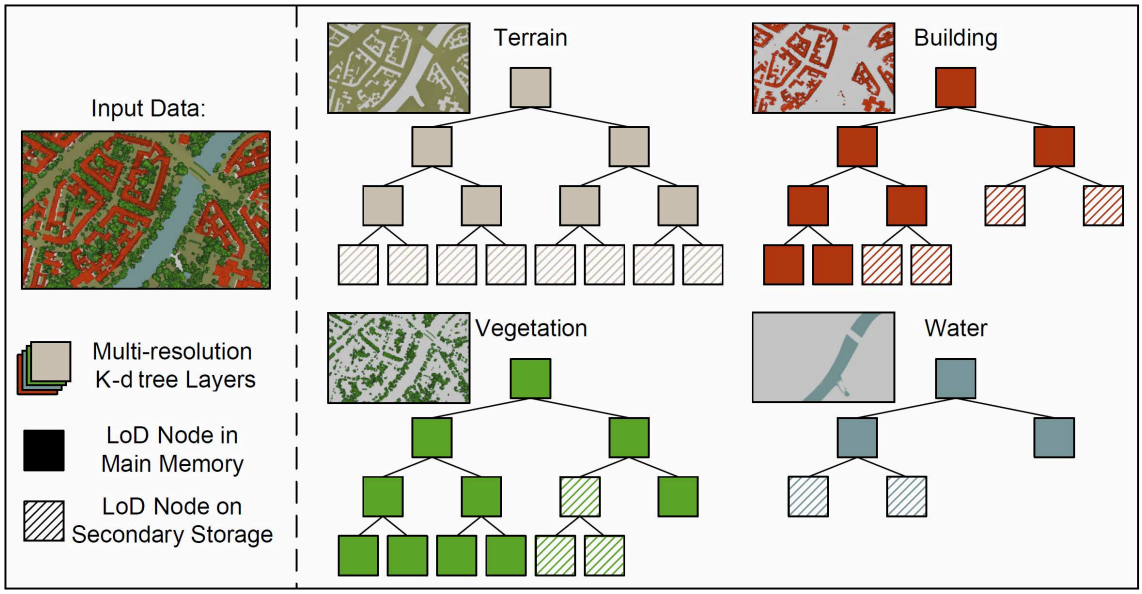


Figure 5.2: *Schematic overview showing the structure of a layered, multi-resolution data structure using the basic concept of a k-d tree. For each object class a separate multi-resolution k-d tree is maintained.*

Histogram-based Subdivision

The data structure is constructed in a preprocessing step and can be stored on secondary storage. First, the given 3D point cloud is subdivided based on layers that should be used, such as object class or point in time (Section 5.4.1, R4). Second, for each layer the related points are arranged in individual data structures with equal-sized data chunks (R4). The construction of a balanced tree with equal-sized data chunks per node requires to order the data for each tree level along the longest spatial extent. To avoid a time-consuming sorting of the entire data, at worst for each tree level, a histogram-based approaches can be used. The histogram is comparable to a voxel grid and is used to count the number of points for spatial areas. It can be used to determine a small subset of the data called *median chunk* that contains the median element that is required to split the data into a left and right subtree (Figure 5.3). In addition, the histogram is used to determine representative points for a defined area that are used to construct a LoD node. The first iteration over the input 3D point cloud fills the histogram. The histogram is used to determine the pivot element of the 3D point cloud that is equal to the splitting plane required to construct a balanced tree structure. A second iteration over the 3D point cloud is used to fill up the current node with representative points (i.e., to create a LoD) and to assign all points to the left or right part of the tree. All points belonging to the median chunk need to be sorted to determine the exact median element. The median element for the split is chosen so that the number of points to the left is a multiple of the node capacity. The histogram for the left and right belonging to the next tree level is prepared during the separation of points of the previous tree level. Thus, only one iteration of the data per tree level is necessary, which is important to reduce the time for preparing the whole data structure.

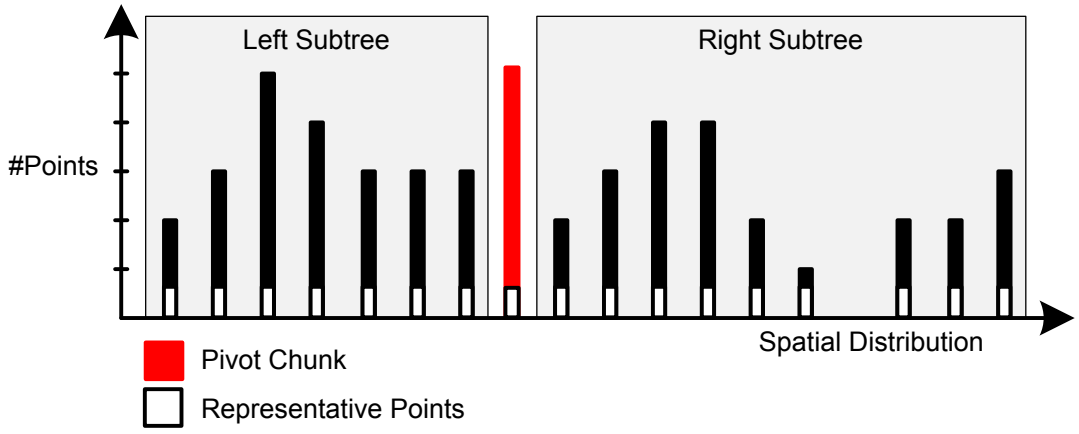


Figure 5.3: *Illustration of the histogram-based construction of the k-d tree to reduce preprocessing times for massive 3D point clouds.*

Out-of-Core Construction

An out-of-core construction is important to handle massive 3D point clouds, e.g., with billions of points (Section 5.4.1 (R2)). If the size of the input data exceeds available memory, the histogram-based subdivision is performed on secondary storage until a subset of the data fits into memory. The processing in memory can be performed in a parallel way using multiple CPU cores. Therefore, a defined number of subtrees (e.g., 16) are constructed by separate threads. The performance of the out-of-core construction depends on the following conditions:

- Size of the 3D point cloud
- Main memory budget or capacity for in-memory processing (Table 5.1)
- Node capacity (Table 5.2)
- Number of available CPU cores (Table 5.4)
- Performance of secondary storage (Table 5.3)

All performance measurements are evaluated independent from each other on a system with the following hardware specifications:

- CPU: Intel Xeon E5 1620, 4x3.6GHz, with hyper-threading and turbo frequencies up to 3.8Ghz.
- Main memory: 2x4GB with 1333MHz, timings are 9-9-9-24, and dual-channel mode.
- HDD: ST31000524AS, Seagate Barracuda 7200.12 with 1TB capacity and an average data transfer rate of 115 MB/sec.
- SSD: Samsung SSD 840 EVO with 500GB capacity, connected with SATA III and an average data transfer rate of 495 MB/sec.

- NAS: Dell PowerEdge R720 with an attached direct-attached storage (DAS). The DAS was a DS S16S-G2240 with 16 ST33000650SS hard drives (Seagate Constellation ES.2 SAS, 6GB/s, 7200RPM, 3TB) in a RAID-6 configuration. The testing system was connected with a Gigabit-LAN to the network and an average data transfer rate of 110 MB/sec.

Default values are a main memory budget of 6 GB, a node size of 4096, a SSD as storage device, and a parallelization with 8 CPU cores.

Main memory budget and performance specification of secondary storage device have the strongest impact on the overall performance. A higher memory budget significantly reduces the required read and write operations. A HDD and NAS as secondary storage have a much lower sequential read and write speed as well as latency that is higher by orders of magnitude compared to a SSD. The number of CPU threads and capacity of nodes does only slightly affect the performance because of the large number of read and write operations that are required during the overall construction process.

5.4.3 Rendering Algorithm

The rendering process can be divided into three stages that are performed per frame. The first stage is responsible for the data provision, caching, and transferring of points from secondary storage to main memory as well as from main memory to GPU memory (Figure 5.1: LoD Node Manager). The second stage applies a point-based rendering techniques (Section 5.5) to all points in GPU memory that should be rendered (Figure 5.1: 3D Rendering Engine). The last stage performs an image compositing if 3D point clouds are rendered with different rendering techniques, e.g., based on semantics information such as object classes (Section 5.5.2).

Memory and LoD Management

The rendering algorithm has to cope with a finite main memory (e.g., 8 GB) and GPU memory (e.g., 2 GB) budget that can be used. The size depends on the available hardware capability, application, and user input. In general, only a small subset of the data can be present in main and GPU memory at the same time. Common 3D point clouds from cities and landscapes have billions of points and could have a size of a few terabyte on secondary storage (e.g., *Berlin2013* contains 80 billion points and requires 2 TB of data). The *LoD Node Manager* has to load, schedule, and remove LoD nodes for the rendering. The following parameters affect the rendering:

Table 5.1: *Evaluation of processing performance in seconds for different main memory budgets in MB. A larger budget leads to less secondary storage access and significantly improves the performance.*

	8 GB		**16 GB**		**32 GB**	
Budget	Time	Speedup	Time	Speedup	Time	Speedup
384	1130	−49.3%	2574	−38.2%	6009	−31.2%
768	1022	−35.0%	2367	−27.1%	5564	−21.5%
1536	932	−23.1%	2195	−17.9%	5183	−13.2%
3072	841	−11.1%	1994	−7.1%	4807	−5.0%
6144	757	0.0%	1862	0.0%	4579	0.0%

Table 5.2: *Comparison of tree construction performance in seconds for nodes with different point capacities. The capacity has a strong impact to rendering performance and quality and does slightly affect the processing performance.*

	8 GB		**16 GB**		**32 GB**	
Node Capacity	Time	Speedup	Time	Speedup	Time	Speedup
1024	809	−6.9%	1971	−5.9%	4806	−5.0%
2048	779	−2.9%	1931	−3.7%	4704	−2.7%
4096	757	0.0%	1862	0.0%	4579	0.0%
8192	726	4.1%	1805	3.1%	4389	4.1%
16384	714	5.7%	1773	4.8%	4363	4.7%

Table 5.3: *Performance comparison in seconds for different storage devices. Seek time and throughput have a strong impact on the performance.*

	8 GB		**16 GB**		**32 GB**	
Storage Type	Time	Speedup	Time	Speedup	Time	Speedup
NAS	2147	−183.6%	5410	−190.5%	14392	−214.3%
HDD	1021	−34.9%	2759	−48.2%	7091	−54.9%
SSD	757	0.0%	1862	0.0%	4579	0.0%

Table 5.4: *Performance overview in seconds for a different number of threads. The performance improvement is not equal with the number of used threads because reading and writing the data cannot be performed in a parallel way.*

	8 GB		**16 GB**		**32 GB**	
# Threads	Time	Speedup	Time	Speedup	Time	Speedup
1	900	−18.9%	2153	−15.6%	5002	−9.2%
2	815	−7.7%	1971	−5.9%	4635	−1.2%
4	789	−4.2%	1898	−1.9%	4482	2.1%
8	757	0.0%	1862	0.0%	4579	0.0%

- Screen resolution (e.g., Full HD)
- User input and navigation (e.g., overview or detailed view)
- Point-based rendering technique (e.g., glPoints or Splats)
- Available main and GPU memory (e.g., 8 GB main memory and 3 GB GPU memory)

The first step to render a 3D point cloud is to load the root node of the data structure into main memory. If the tree structure has multiple layers, all root nodes belonging to layers that should be rendered are loaded. Each node is equal to a LoD node and has a defined size. A bounding sphere is used to estimate the volume of a LoD node in object space. Bounding boxes or oriented disks would estimate the volume of a LoD more precise. However, culling and a size evaluation is computational more expensive and additional memory to store the LoD representation is required. Bounding spheres as LoD approximation enable the evaluation of much more LoD nodes per frame in contrast to other primitives (Richter & Döllner 2010b). The bounding sphere is specified in object space and can be projected into screen space to perform culling and determine the size on the screen. The size of a LoD node on the screen can be estimated as a number of pixel and is called projected node size (PNS). The number of covered pixels per node and the number of points per LoD node is used to estimate the required LoD to render the 3D point cloud. If the PNS is above a threshold, child nodes are added to increase the point density. If the PNS of a node is below a threshold, it does not have to be rendered because the parent node is sufficient to render a frame with a dense 3D point cloud. The PNS for each node changes during user interaction due to a changing position of the virtual camera. All nodes that need to be rendered are mapped to GPU memory into a vertex buffer object (VBO). The VBO is divided into equal sized chunks that can store exactly one LoD node. Nodes that are not available in main memory are loaded from secondary storage into main memory. Loading nodes from secondary stored typically requires several frames due to slower access times on secondary storage. The loading time depends on secondary storage performance and the number of nodes that are already in the request queue. If the user interacts with the 3D point cloud some node reload requests can become unnecessary. Reasons can be that they are outside the frustum or too far away, and the parent LoD is sufficient. For that reason all reload requests are evaluated and updated per frame. Figure 5.4 illustrates a possible memory usage and related LoD node locations.

Rendering Layered 3D Point Clouds

In general, the PNS-threshold is used for all LoD nodes and depends on the available memory, computing capability of the GPU, and used point-based rendering technique. However, a more adaptive and configurable setting of the PNS-threshold is required if subsets of a 3D point cloud are rendered with different rendering techniques. An example is the subdivision of the point cloud into object classes. Each object class that should be rendered has its own memory budget (Figure 5.5) and is balanced permanently during the rendering process because the amount of memory required by an object class may vary due to the following reasons:

- Only a small number of points belonging to an object class is visible during the exploration

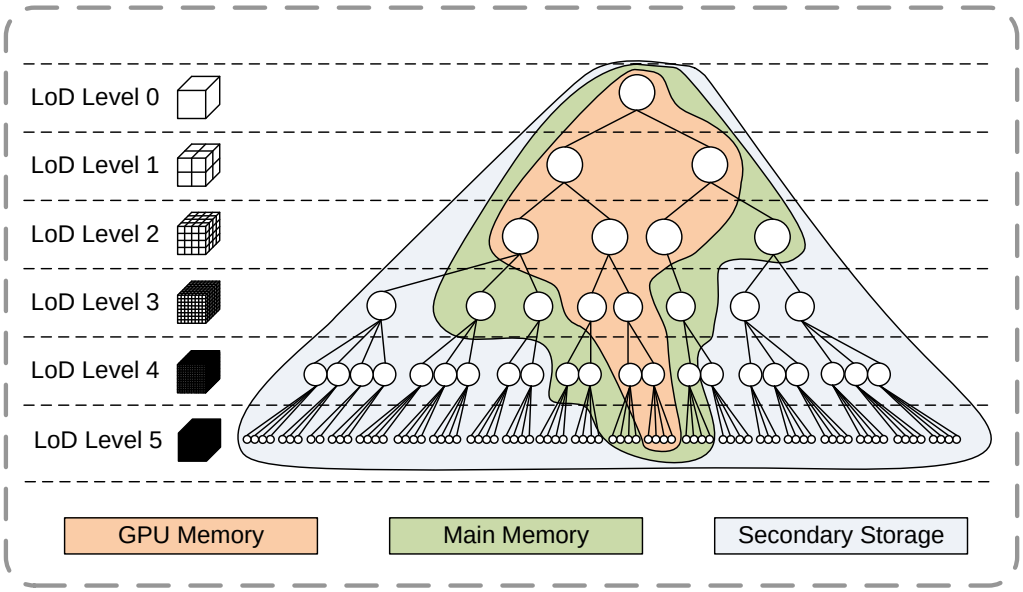


Figure 5.4: *LoD data structure used for out-of-core rendering. LoD nodes are available in GPU memory (orange), main memory (green), or secondary storage (blue).*

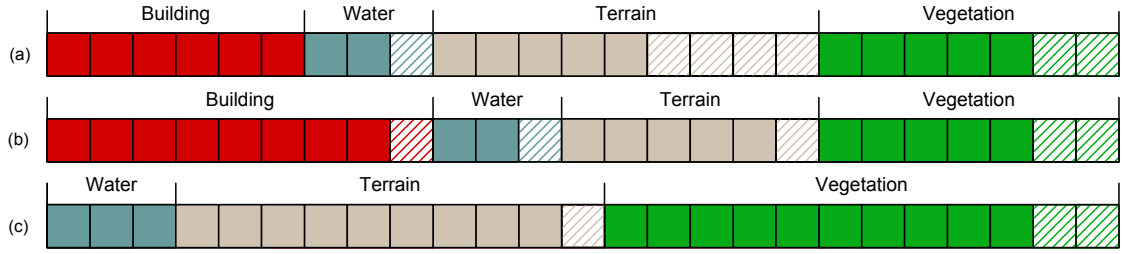


Figure 5.5: *Illustration of an exemplary GPU memory usage that is balanced during rendering according to memory requirements of LoD nodes that belong to different object classes. (a)-(b) illustrate how unused memory is assigned to other object classes. (b)-(c) illustrate the balancing process when the visualization of one object class (i.e., layer) is disabled.*

- Visualization of certain object classes is disabled
- Close up views require a high point density for an object class (e.g., for a building)

Rendering techniques for a subset can allocate additional memory until the memory capacity is exceeded. Therefore, the PNS-threshold is decreased to increase the point density, i.e., the rendered LoD. Object classes can be rendered with different LoDs because the required number of points for an appropriate rendering result depends on the structure. For example, buildings can be rendered with more points due to detailed roof structures in contrast to terrain or vegetation that can be rendered with less points. The lower point density can be compensated by using larger primitives, e.g., splats for terrain or spheres for vegetation.

Rendering Front for Caching

The rendering algorithm is designed to fulfill two aims: the first purpose is the point throughput to render as many points as possible to get a closed surface. This is necessary to present large parts of the 3D point cloud with a high resolution. The second one is to guarantee high frame rates during interaction to enable interactive exploration of the data.

The projected node size (PNS) threshold for displaying nodes is adapted dynamically depending on the user input and available hardware resources. Traditionally, a depth-first traversal of data structures is performed until the PNS threshold is reached (Wand *et al.* 2007). The traversal of the entire data structure is time-consuming and, if less or no interaction or navigation is performed, a high frame-to-frame coherence is given. To exploit the frame-to-frame coherence all LoD nodes selected for rendering are kept per frame. These nodes are stored in a list, called *rendering front*, which cuts the data structure in two parts (Figure 5.6). All nodes above the rendering front have a PNS larger than the PNS threshold and are rendered whereas all nodes in the rendering front have a lower or equal PNS. The rendering front is used for the next frame to check the validity of nodes. If the PNS of a node is larger than the threshold, it can be replaced with its child nodes (Figuge 5.6 (a)). On the other hand, if the size of a parent node in the rendering front is below the PNS threshold, all nodes with this parent can be replaced by the parent node (Figure 5.6 (b)).

The number of LoD node updates per frame is limited to achieve interactive or real-time frame rates as well as a smooth refinement of the rendered 3D point cloud. The number of nodes that can be added or removed per frame are estimated from the rendering duration of the last frames. The PNS threshold is increase if a user interacts with the scene and can be decrease if no interaction is performed to increase the point density and rendering quality.

Memory Release Strategies

A fixed memory budget is used to prevent running out of main memory. If this memory limit is reached, no reloads are performed and the algorithm releases memory resources by removing unused parts of the tree structure. The rendering front is used to determine which LoD nodes can be removed from main memory: all nodes in the rendering front were rendered in the last frame. They serve as good indicator to approximate, which parts of the data structure can become important in the next frames. The possibility that a node

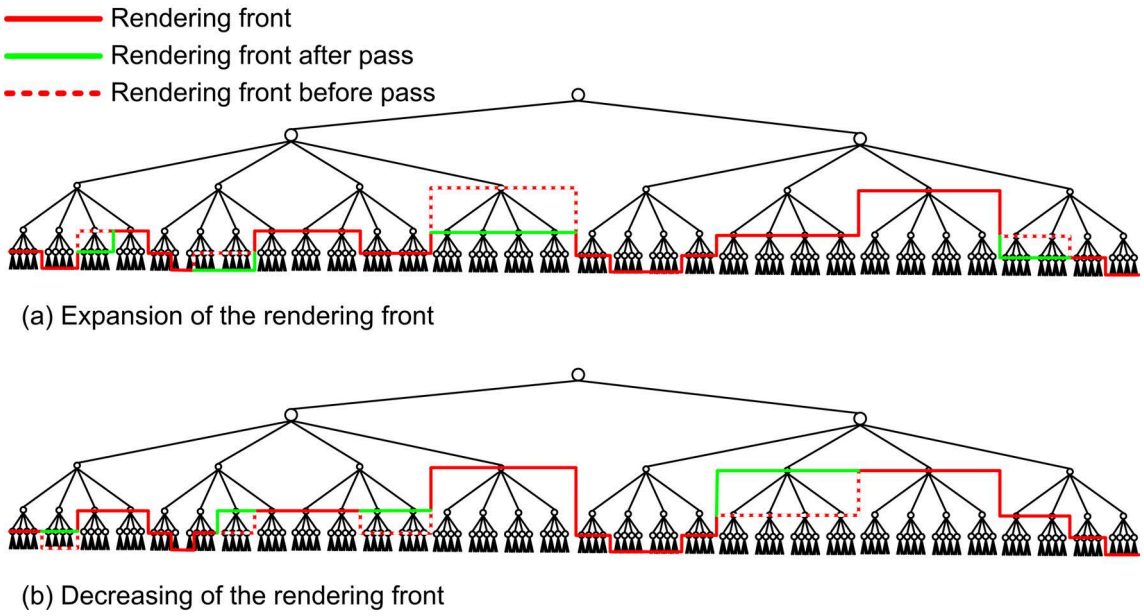


Figure 5.6: *Illustration of the tree structure with a related rendering front for one frame. In (a) the rendering front is expanded at 11 nodes because their PNS does not fulfill the PNS threshold; (b) illustrates the decreasing of the rendering front by replacing nodes by their parent.*

is needed in the next frame decreases with the distance of the node to the rendering front in the tree hierarchy. An additional indicator is the timestamp of the node that enables to identify outdated nodes that were not rendered in the last frames.

5.5 Rendering Techniques

The out-of-core rendering algorithm, presented in the previous section, is responsible to select LoD nodes and keep all required points in GPU memory. In this section, different point-based rendering techniques are presented that can be used in combination with the presented rendering system and out-of-core data structure in Section 5.4.

To efficiently render 3D point clouds the Graphics Processing Unit (GPU) supports point primitives, such as *GL_POINTS* in OpenGL. However, these primitives have a fixed size in pixels (Shreiner *et al.* 2013) (e.g., Figure 5.7 (a) uses a size of 3 pixel), i.e., their size in object space varies according to their perspective depth. Depending on the view position undersampling, i.e., holes between neighboring points (Figure 5.7 (a) - bottom), or oversampling, i.e., visual clutter due to overlapping points (Figure 5.7 (a) - top), occurs. Rendering all points in a "uniform way" does not take into account characteristics of different surfaces that belong to object classes, such as vegetation, building, terrain, street, or water. For example, building façades generally exhibit lower point density in contrast to roofs and terrains. A uniform rendering, therefore, results in gaps between neighboring façade points (Figure 5.7), complicating their perception as a continuous surface. If points are rendered by the point primitives of the underlying rendering system (e.g., OpenGL's *GL_POINTS*) they are not scaled according to the camera distance making it difficult to correctly estimate depth differences and leading to visual artifacts due to overlapping of points close to each other. In addition, a uniform rendering does not differentiate between surface characteristics such as planar (e.g., terrain), structured (e.g., roof structures), and

(a) (b)

Figure 5.7: *(a) Example of a massive 3D point cloud rendered in a uniform way by GL_POINTS primitives and textured by aerial photography. (b) Same scene rendered by class-specific point-based techniques: different object classes can be better distinguished, holes on façades are filled, and visual clutter in the background is reduced.*

fuzzy areas (e.g., vegetation), complicating the visual identification and categorization of objects and structures by the user.

Point specific attributes can be used to adapt the appearance of a point, i.e., its color, size, orientation and shape, at run-time (Kim *et al.* 2012; Goesele *et al.* 2010; Gao, Nocera & Neumann 2012). The color of a point can be chosen based on its color value, object class, topology attributes (i.e., surface normal, horizontality, global, or local height), or a combination of them. The orientation of a point can either correspond to its surface normal, the current view direction or a defined uniform vector. In addition, size and shape type of a point can be set dependent on its object class.

Figure 5.8 shows a schematic overview of the 3D rendering engine that implements different point-based rendering techniques. Points are categorized by object classes and rendered with different styles to fulfill different requirements:

- Close the surface represented by the 3D point cloud
- Improve recognition of structures and objects
- Generalization of structures
- Combination of rendering techniques based on object-class information

5.5.1 Point-based Rendering

This section presents a variety of point-based rendering techniques that have been implemented. Advantages and disadvantages are discussed and illustrated for different data sets.

Point Splats

To avoid undersampling and oversampling due to changing view positions, the Point Splats technique renders each point as an opaque disk defined in object space that can be

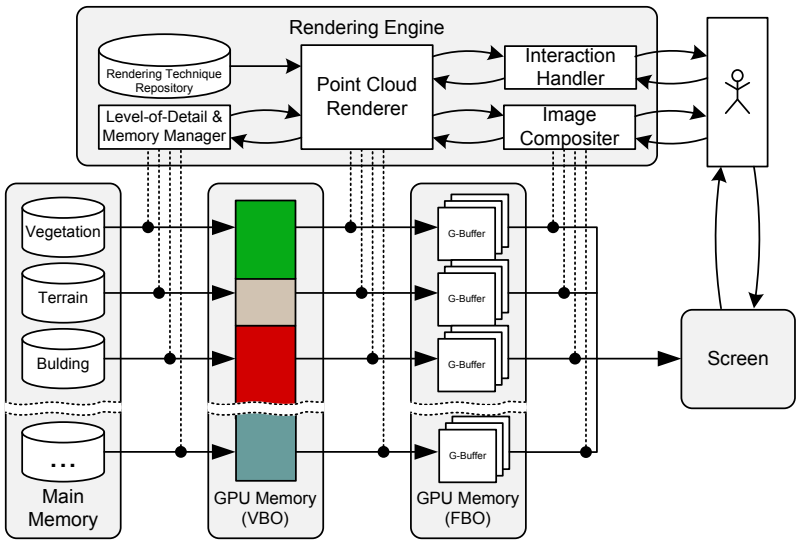


Figure 5.8: *Schematic overview of the class-specific point-based rendering system. Categorized by object classes, points are transferred to GPU memory and rendered into separate G-Buffers that are composed to synthesize the final image.*

oriented alongside the surface normal (Rusinkiewicz & Levoy 2000; Botsch *et al.* 2005). The on-screen size depends on the current view position and angle, ensuring a perspective correct visualization (Figure 5.9 (a–f, i)). However, the perception of depth differences between overlapping points that are colored homogeneously (e.g., points belonging to the same object class), is generally limited.

Point Spheres

This point-based rendering technique is useful to emphasize the three-dimensional character of a point. The proposed Point Spheres extend the original splat concept by rendering points as hemispheres instead of flat disks that are always facing the view position and, thus, look like spheres (Rusinkiewicz & Levoy 2000). These hemispheres are created by adding an offset to each depth value of the rendered fragment and by shading each fragment. The depth offset as well as the shading color can be determined by projecting the fragment onto the plane defined by the related splat and by calculating the projected distance of the fragment to the center of the splat. Point spheres are well suited for non-planar and fuzzy surfaces, such as vegetation (Figure 5.9 (g)).

Silhouette Rendering

Point-based silhouettes highlight and abstract silhouettes and distinctive surface structures (e.g., deep differences). This technique extends the splat rendering approach and was originally proposed by Xu et al. (Xu *et al.* 2004). Similar to the rendering of point spheres, color and depth of each fragment depend on its projected distance to the center of the splat. In addition, the splat is split into an inner and an outer part. Fragments in the outer part represent the silhouette and are rendered with an increased depth value and another color. As a result, depth discontinuities between overlapping points exceeding the depth offset are highlighted (Figure 5.9 (h, j, l)).

Table 5.5: *Characteristics of the datasets used to evaluate the performance of the presented point-based rendering approach.*

	Dataset 1	**Dataset 2**	**Dataset 3**
Name	*Berlin2008*	*Frankfurt2009*	*Berlin2013*
Point Density	7-8 pts/m^2	28 pts/m^2	100 pts/m^2
Number Points	4.7 bln	7.1 bln	80 bln
Data Size	112 GB	159 GB	1788 GB

Solid Rendering

This point-based rendering technique was developed to render buildings with solid and hole-free façades. As the point density on façades in airborne laser scans is very low in contrast to horizontal structures, the efficient identification of building segments is limited because other structures behind a building are visible through the façade (Gao, Nocera & Neumann 2012). To overcome this, a second rendering pass to fill the area below roof points with new primitives is implemented. The *geometry shader* is used to render a point-based splat, sphere, or silhouette equal to the rendering techniques presented above and an additional quad that imitates the façade below a point. The quad width is equal to the point size used in the first pass and the height ranges from the point position to the terrain level. All quads are aligned to the view position and have the same color or color gradient (e.g., height-based) to create a solid façade look (Figure 5.9 (k)).

5.5.2 Compositing of Point-based Rendering Techniques

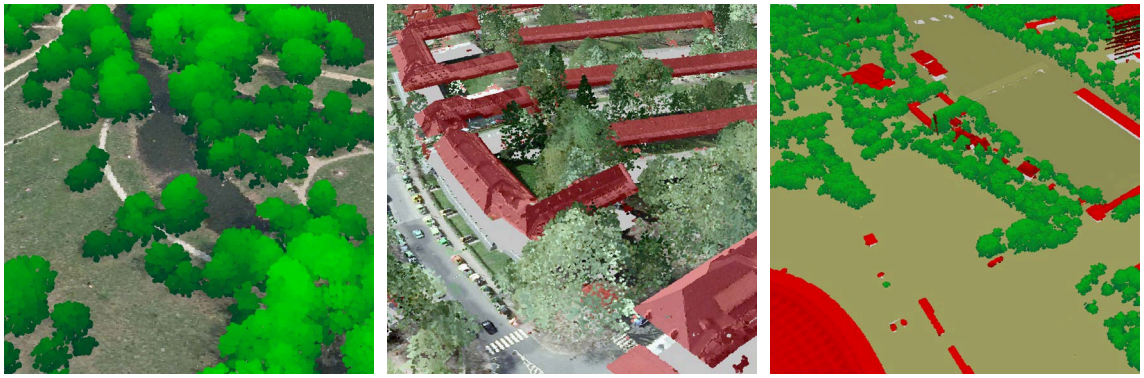
To combine different point-based rendering techniques, a multi-pass rendering utilizing G-Buffers for image-based compositing is used (Figure 5.8). G-buffers are specialized frame buffer objects (FBO) that store multiple 2D textures for color, depth, or normal values (Saito & Takahashi 1990). Each object class requires one rendering pass. Results for each class are stored in G-Buffers that are combined by the final rendering pass. Figure 5.10 shows an example of the combination of different point-based rendering techniques. The image-based compositing enables to implement rendering techniques for focus + context visualization (Trapp *et al.* 2008; Vaaraniemi, Freidank & Westermann 2012) such as interactive lenses (Figure 5.11 (b)). Moreover, object-class specific visibility masks, i.e., static lenses, can be computed and applied during the rendering to highlight occluded structures (Figure 5.11 (c)). All point-based rendering techniques can be independently selected, combined, and configured at run-time to adjust the appearance of each object class.

5.6 Performance Evaluation

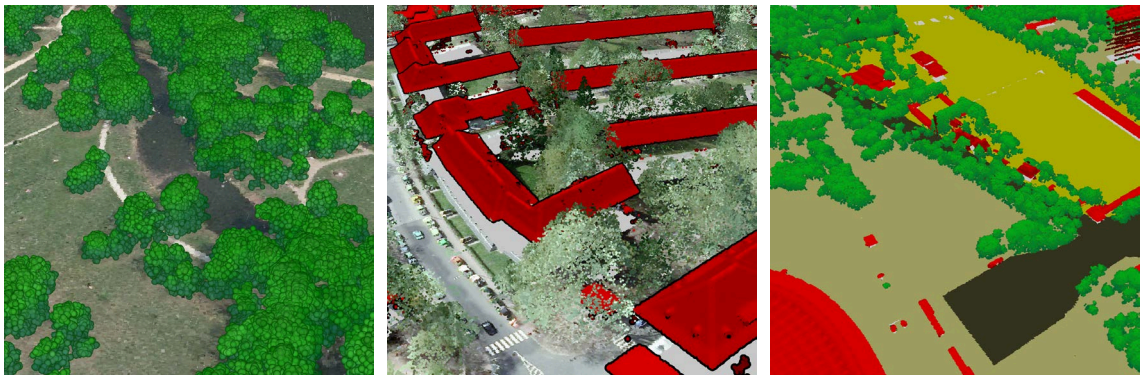
The presented system and point-based rendering techniques are evaluated with three massive 3D point clouds containing up to 80 billion points (Table 5.5). Measurements and tests were performed on an Intel Xeon CPU with 3.20 GHz, 12 GB main memory, and a NVIDIA GeForce GTX 770 with 2 GB device memory.

As shown in Figure 5.12, interactive frame rates can be achieved for each rendering technique as long as the overall number of rendered points does not exceed a certain

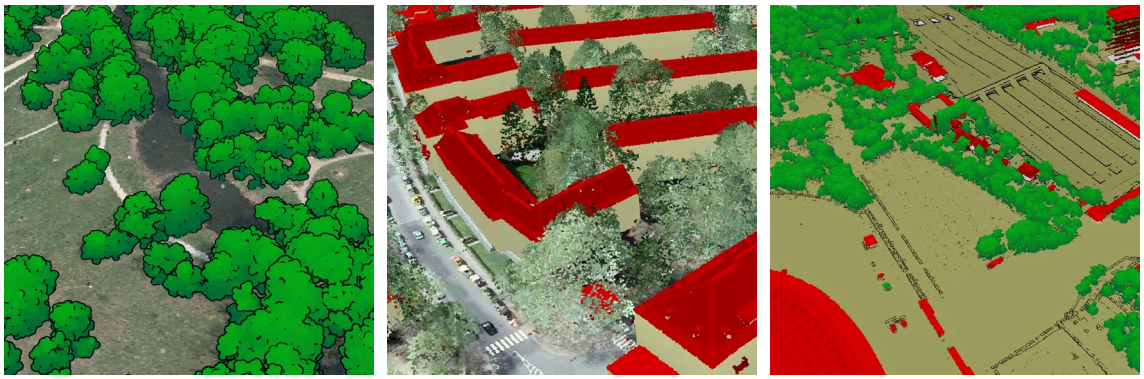
(a) *Point Splats; aerial image colors.* **(b)** *Point Splats; aerial image colors.* **(c)** *Point Splats; aerial image colors.*



(d) *Points Splats; global height.* **(e)** *Point Splats; aerial image colors and object-class information.* **(f)** *Point Splats; global height.*



(g) *Point Spheres; local height.* **(h)** *Silhouettes; horizontality.* **(i)** *Splats; object-class information.*



(j) *Silhouette Rendering; local height.* **(k)** *Solid Rendering; horizontality.* **(l)** *Silhouettes; global height.*

Figure 5.9: *Examples of massive 3D point clouds rendered with different rendering setups for vegetation (left), buildings (middle), and terrain (right).*

Figure 5.10: *Examples of a 3D point cloud rendered with different point-based rendering techniques taking into account object-class information per point. (left) All points are rendered as point splats with color information from aerial images. (right) Buildings are rendered with solid rendering, vegetation with point spheres, and ground with point splats. The color depends on the surface category.*

(a) **(b)** **(c)**

Figure 5.11: *Examples of focus + context visualization for classified 3D point clouds. (a) Regular visualization with buildings partially occluded by vegetation. (b) Interactive focus + context lens. (c) Static focus + context lenses positioned around building points.*

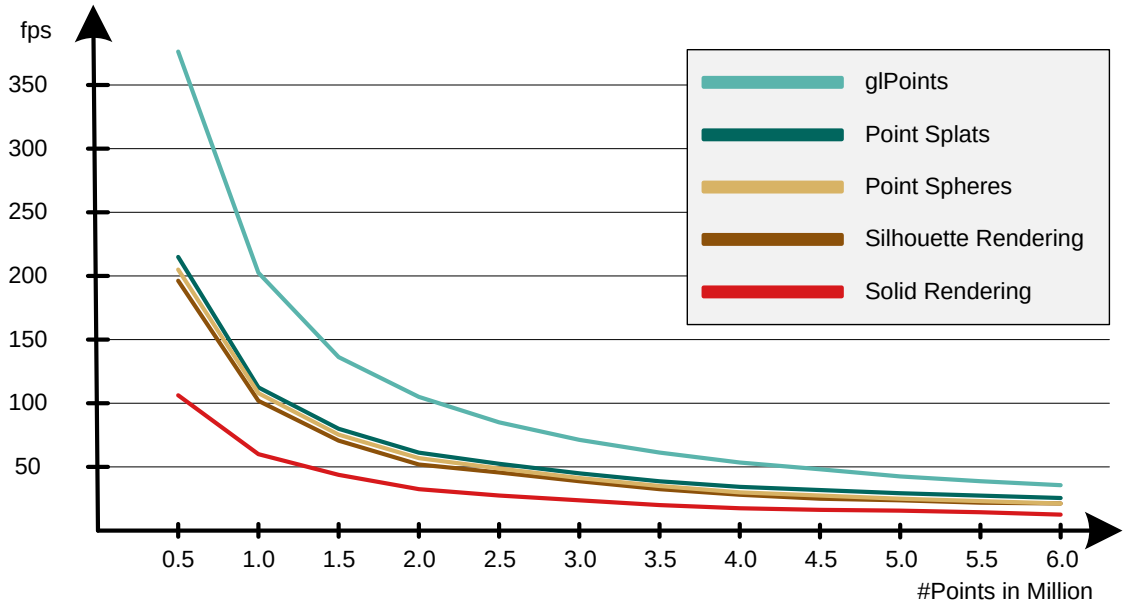


Figure 5.12: *Rendering performance in frames per second (fps) using different sized subsets of the datasets from Table 5.5.*

threshold (e.g., 6 million points for the solid rendering approach). The highest frame rate could be observed for *GL_POINTS*, which was expected since these primitives are supported natively by the GPU. Point Spheres as well as the solid and silhouette rendering approach extend the concept of Point Splats and increase the computational effort during rendering. Consequently, lower frame rates were achieved when using these techniques for rendering as opposed to Point Splats. Furthermore, the performance for Point Spheres is higher than for Point Silhouettes due to a more hardware demanding shading implementation (e.g., conditional branching).

Since the proposed out-of-core rendering approach limits the number of rendered points by dynamically selecting them, massive datasets with varying point densities can be rendered in real-time as well (Table 5.6).

Table 5.6: *Rendering performance in frames per second (fps) using the proposed out-of-core rendering approach. Each dataset is evaluated for a close and a far perspective.*

	Dataset 1		**Dataset 2**		**Dataset 3**	
	Far	Close	Far	Close	Far	Close
#Rendered Points in Million	2.32	0.50	3.42	0.85	4.85	1.04
GL_POINTS	86.39	378.07	60.02	246.12	40.24	194.35
Point Splats	51.84	214.32	32.27	138.67	23.01	108.63
Point Spheres	49.57	203.81	28.31	133.72	22.35	107.07
Silhouette Rendering	46.07	195.65	26.66	127.38	22.18	106.97
Solid Rendering	27.32	100.13	20.22	63.78	18.74	59.45
Combination 1 (Figure 5.9, row 3)	40.51	200.97	27.33	128.73	22.45	107.75
Combination 2 (Figure 5.9, row 4)	33.28	126.31	22.21	80.80	19.90	68.47

5.7 Summary and Discussion

This chapter presents a rendering system for point-based graphics that is designed to process and render 3D point clouds. Main advantages of the presented real-time rendering system for 3D point clouds are the following:

- **Out-of-core rendering strategies** to cope with large-scale 3D point clouds with billions of points in short processing times. The presented system adapts automatically to available hardware resources such as main memory and GPU memory. It achieves high frame rates during user interaction and also a high point density if no interaction is performed for any sized 3D point clouds. The technical limit is the available storage capacity of the used system or network infrastructure.
- **Point-based rendering techniques** to visualize captured objects, surfaces, and landscapes with a photorealistic, non-photorealistic, or solid look for different applications. These techniques support to dissolve occlusion and enable a task-specific interactive exploration.
- **Adaptive multi-resolution data structure** that enables in addition to a spatial data selection an object-class specific selection of LoDs. Hence, memory and processing resources can be used economically and adaptively.
- **Thematic attributes and additional per-point information** to select and combine specialized point-based rendering techniques to enhance the visual appearance and facilitate recognition of objects within 3D point clouds. In addition, it enables focus + context techniques, e.g., lenses for filtering and highlighting.

To summarize, rendering techniques for 3D point clouds offers many degrees of freedom for graphics and interaction design. Due to an increasing availability of scanning devices the number of datasets will most likely grow dramatically. A variety of traditional and new application domains will be confronted with large-scale, dense, and time-variant 3D point clouds. The visualization itself is a fundamental component for systems and IT-infrastructures to derive, provide, and communicate spatial information.

Future work could focus on point-based rendering techniques that enable a per-frame reconstruction of object surfaces (Preiner, Jeschke & Wimmer 2012), e.g., for terrain or roof points. The combination of 3D point clouds from aerial, mobile, and terrestrial scans for indoor and outdoor environments has a big potential. Further research directions are service-based and web-based solutions to be independent from data storage location and hardware capabilities of end user systems. A web-based solution that can be integrated into existing applications and systems can open up new markets and enables to provide massive 3D point clouds to a large number of users (Schütz 2015; Martinez-Rubi *et al.* 2015).

Chapter 6

Framework for Analysis and Visualization of 3D Point Clouds

This chapter introduces the software framework that implements the processing, analysis, and visualization techniques for 3D point clouds introduced in this thesis. The implementation relies on a service-based software architecture that provides the prerequisite for the integration into various applications and system infrastructures and for the distribution of this services to a variety of application domains. This chapter is partially based on the author's scientific publications in Richter and Döllner (2012) and Discher, Richter and Döllner (2018).

6.1 Architecture

The developed framework for 3D point clouds can be divided into two subframeworks called *PCLib* and *PCViewer*. The PCLib includes features and implementations to import, export, manage, process, and analyze 3D point clouds as well as other spatial data such as 3D models, aerial images, and vector data. The PCViewer provides real-time rendering and visualization techniques for inspection and exploration of massive 3D point clouds and related analysis results. In addition, the PCViewer implements features to render various 2D and 3D models (e.g., 3D city models, terrain models, aerial images, shapefiles) in the context of 3D point clouds. Both subframeworks implement out-of-core data structures to cope with massive 3D point clouds. Used programming languages, middleware technologies, and third party frameworks are:

- **C++** as object oriented programming language
- **Compute Unified Device Architecture (CUDA)** as programming API for general purpose computations on the GPU
- **OpenSceneGraph** as scene-graph based rendering middleware and container for the 3D rendering engine
- **OpenGL and GLSL** to implement rendering and shading techniques that are provided by the 3D rendering engine
- **Qt** as application framework to implement the graphical user interface
- **Point Cloud Library (PCL)** as open source library to use selected processing algorithms for 3D point clouds

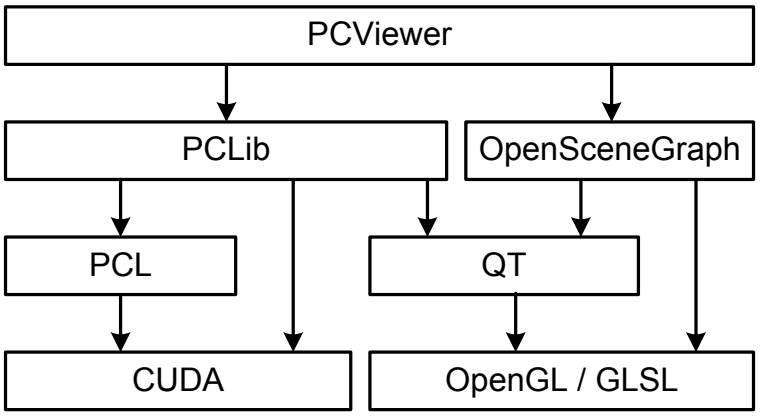


Figure 6.1: *Overview showing the usage relations of the point cloud framework, third party frameworks, and middleware technologies.*

The usage relation of PCLib, PCViewer, third party frameworks, and middleware technologies are illustrated in Figure 6.1.

The PCLib is responsible to import and export various data formats for 3D point clouds, 3D models, image data, and vector data. All data formats can be used as input for processing and analysis as well as for rendering and visualization tasks implemented by the PCViewer. Main features of the PCLib are illustrated in Figure 6.2. Core features for import, export, processing, and analysis have been implemented modular with a defined interface describing all required and optional input and output data. Hence, all implemented features and algorithms can be re-used and combined in a flexible way for an efficient implementation of domain specific applications. A graphical user interface allows to configure, combine, and design point cloud processing workflows as illustrated in Figure 6.3. A more complex pipeline is given in Figure 6.4 and illustrates a composition of input, processing, and output components to detect individual trees in 3D point clouds from aerial scans. The following enumeration is a selection of applications that have been implemented based on core functions of the PCLib:

- Classification of aerial 3D point clouds
- Building outline extraction
- Terrain model extraction and generalization
- Tree detection and tree cadastre extraction
- Tree cadastre evaluation
- Change map generation
- Update of 3D city models
- Point cloud converter

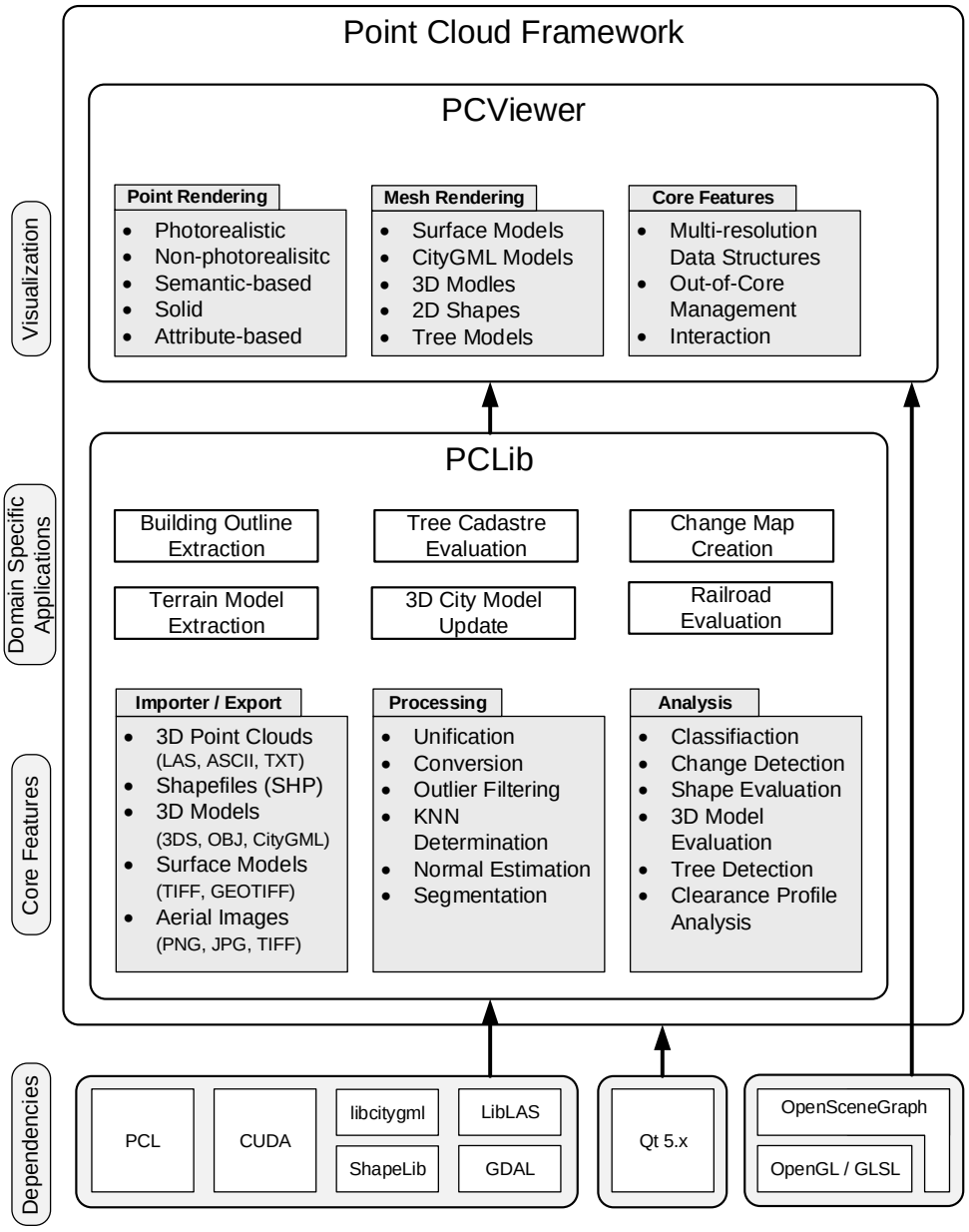


Figure 6.2: *Illustration of the main features for the import, export, processing, and analysis as well as applications that have been implemented in the PCLib.*

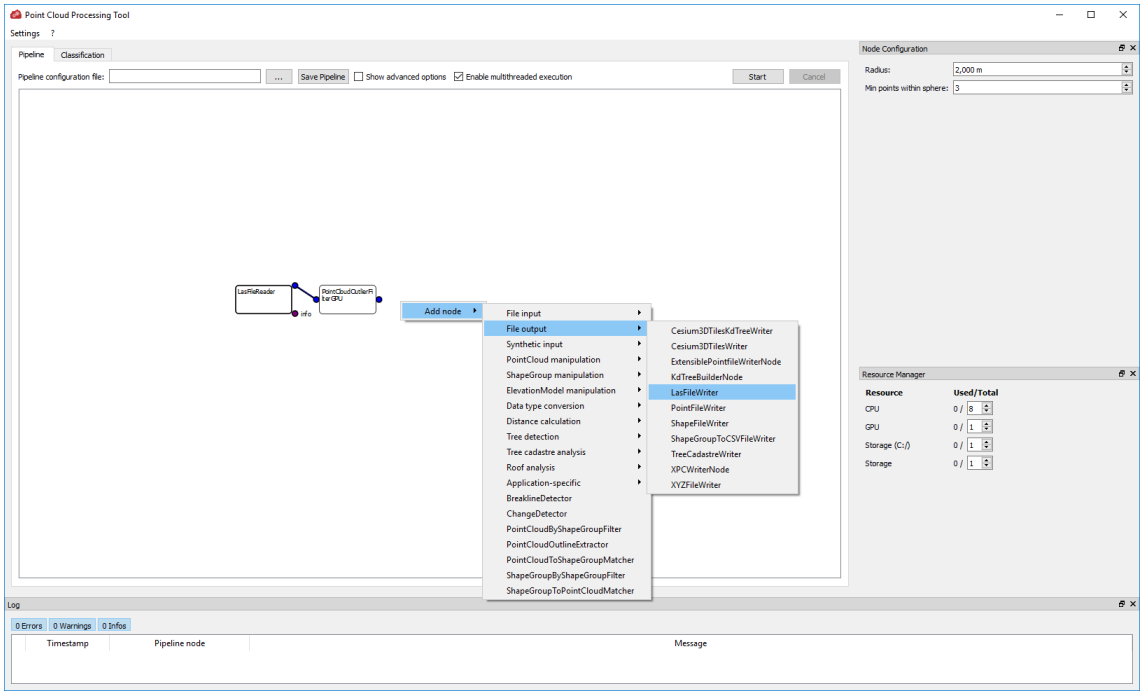


Figure 6.3: *Application for configuration of point cloud processing pipelines. The "Pipeline" canvas can be used to add different node types for import, processing, analysis, and export. A "Node Configuration" widget allows to configure nodes of the pipeline and setup parameters, e.g., radius of point proximity analysis as illustrated in the figure.*

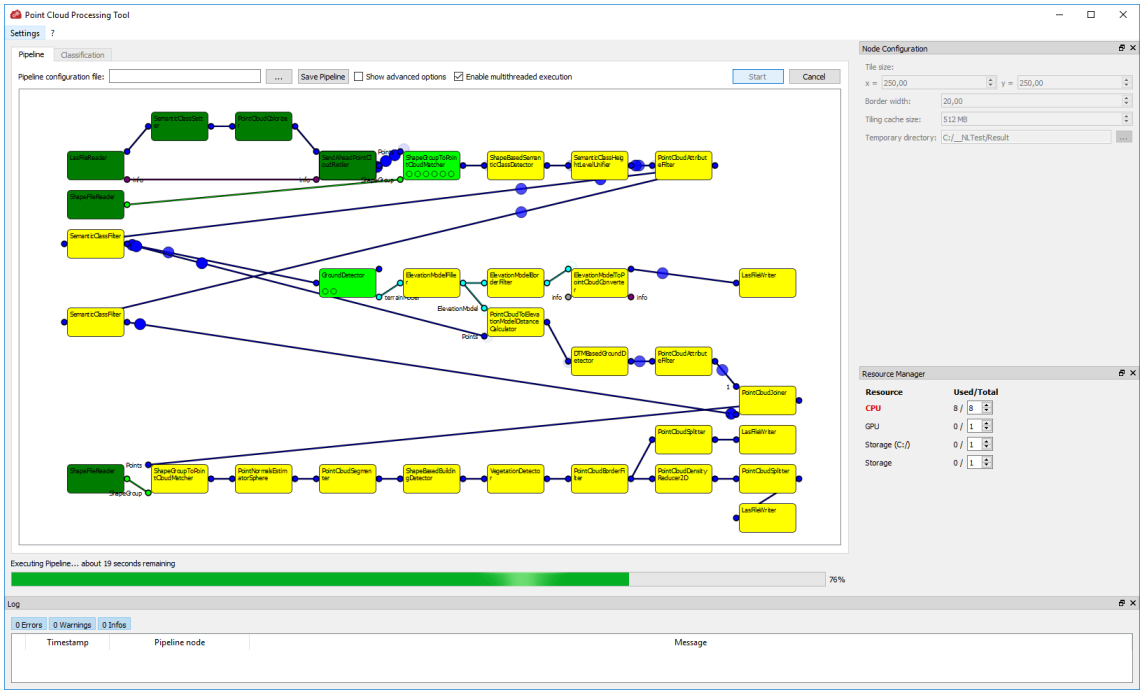


Figure 6.4: *Illustration of a running processing pipeline for classification of 3D point clouds. The "Resource Manager" widget shows available and used hardware resources, e.g., CPU cores. The state of each node is either finished (dark green), running (light green), or waiting (yellow). Data packages are illustrated as blue dots on the edges between nodes.*

The PCViewer implements a 3D rendering engine for 3D point clouds that is capable to render 3D point clouds in a photorealistic, non-photorealistic, or solid style as well as in a way that includes semantics information and per-point attributes as presented in Chapter 5.5. Figure 6.5 shows the application with the user interface that allows to configure the visualization. Several interaction techniques, such as panning, hovering, zoom as well as flight and game navigation techniques can be used to explore the data. The out-of-core functionality is implemented based on a multi-resolution data structure and out-of-core management modules as presented in Chapter 5.4.1.

The memory management is implemented adaptively to assign main memory and GPU memory resources automatically or manually. This is important for use cases with other 3D models that should be visualized in the context of 3D point clouds. Supported data types and formats are:

- 3D point clouds (asc, pf, ptx, xpc, ply, las, laz, fx, t3c, bin)
- Digital surface and terrain models (Tiff, GeoTiff, ASCGrid, txt)
- 3D city models (CityGML, 3ds)
- 3D models (3ds, obj, vrml)
- 2D shapes (shp, osm)
- Aerial images (Tiff, GeoTiff, png, jpg)
- Tree models as parametrized impostors (jpg, shp, gml)

Another important feature is the capability to measure in the data and extract per-point information on-demand. This is a challenging task for massive 3D point clouds because straight forward techniques such as raycasting the point cloud data itself do not work. They cannot be performed in real-time for massive data sets and do not consider different rendering techniques, i.e., with a varying point size. Picking techniques that operate in the screen space can solve this limitations. The implemented technique uses an additional buffer, called ID-buffer, that is in contrast to the color buffer not visible for the user. This ID-buffer stores per point a unique ID and allows to identify a point and related attributes regardless the used rendering technique. Point coordinates and per-point attributes are shown to the user in the status bar (Figure 6.6). This accurate point selection approach allows to implement interaction techniques, for example to measure the distance between points and areas, which are required by most applications, e.g., in the field of documentation and construction (Figure 6.6).

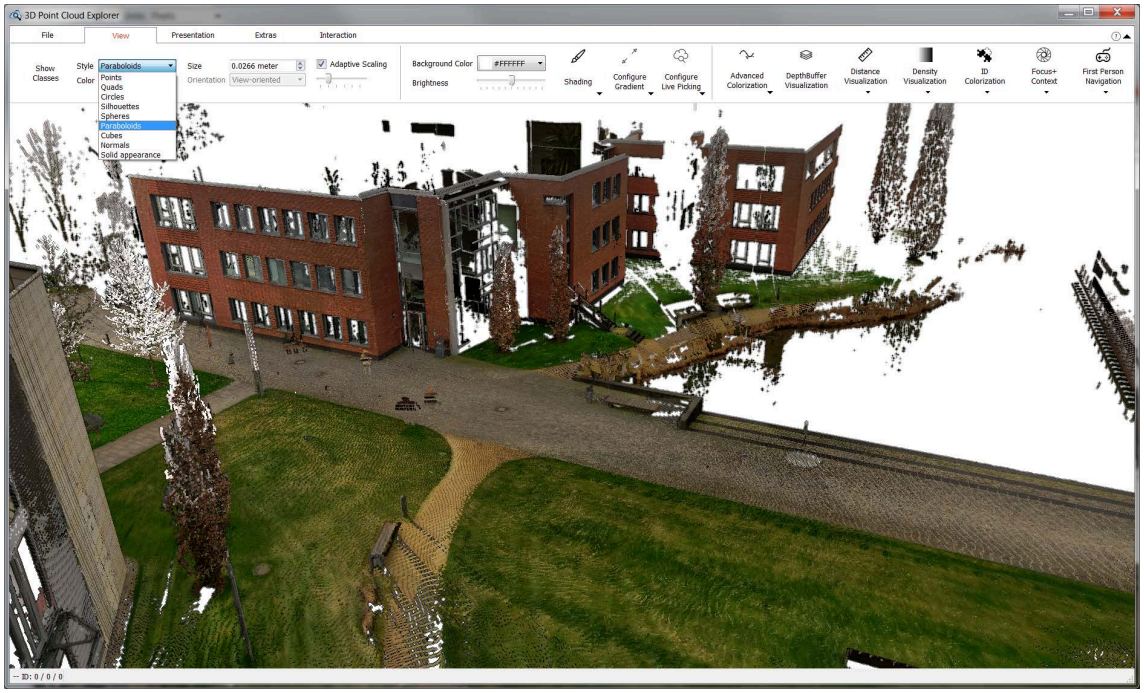


Figure 6.5: *PCViewer application that implements a variety of real-time rendering techniques for massive 3D point clouds. Users can select different rendering styles, color schemas, and semantic-based visualization modes (e.g., based on analysis results) to configure the appearance of the 3D point cloud.*

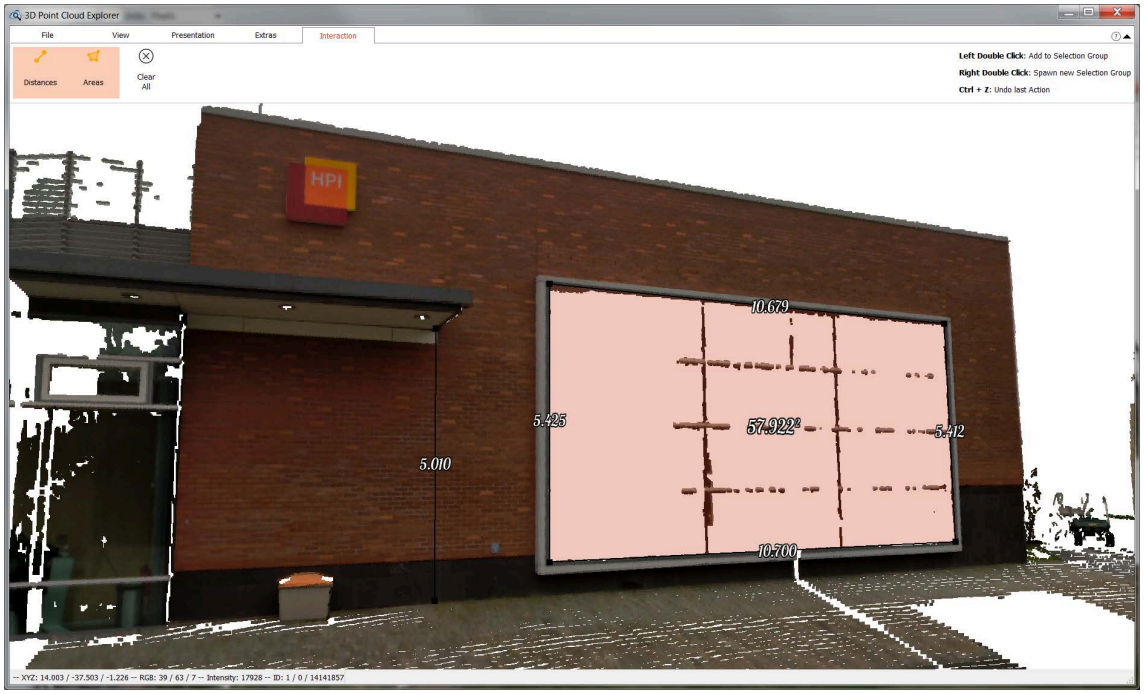


Figure 6.6: *PCViewer application showing features to measure distances (e.g., height of the roof) and areas (e.g., size of the window) in 3D point clouds.*

6.2 Service-oriented Architecture

The provision of individual modules is particularly important when system features should be integrated into existing applications, systems, and infrastructures that require only selected features for the import, processing, analysis, or visualization of 3D point clouds. Hence, the modular and loosely coupled architecture facilitates the construction of distributed, scalable, and expandable systems that can be easily configured according to the field of application.

Key requirements for the overall framework design is reusability and modularity of all components. Common workflows include the following tasks and subsystems:

1. an infrastructure to collect 3D point clouds, e.g., from multiple, heterogeneous sources
2. system components to integrate, update, and access 3D point clouds, e.g., in a database
3. applications and algorithms that operate directly on 3D point clouds
4. rendering techniques to enable a task and application-specific visualization of 3D point clouds
5. visualization services to provide 3D point clouds for heterogeneous clients

Figure 6.7 gives an example and illustrates a composition of components for the collection, preparation, and integration of 3D point clouds from different, heterogeneous data sources (e.g., aerial and terrestrial LiDAR scans or image collections) as well as autonomously operating scanning systems (e.g., cars). The continuous data acquisition results in redundant 3D point clouds for large parts of the captured area (Kang & Lu 2011). To avoid the need for redundant storage, *incremental updates* and *change detection* can be used to determine the differences between collected and already available 3D point clouds. This allows for *selective updating* parts of the data.

The service-oriented architecture takes into account and is partially based on GIS-oriented SOA-services, e.g.,:

- Web Processing Services (Schut 2007) for processing and analysis tasks
- Web Feature Services (Vretanos 2010), Web 3D Services (Schilling & Kolbe 2010) or Web Coverage Services (Baumann 2012) for data provisioning
- Web View Service (Hagedorn 2010) for visualization tasks

Figure 6.8 illustrates such an service-oriented architecture with call relations between components that are assigned to common layers of a service-oriented architecture (Döllner, Hagedorn & Klimke 2012; Prandi *et al.* 2014).

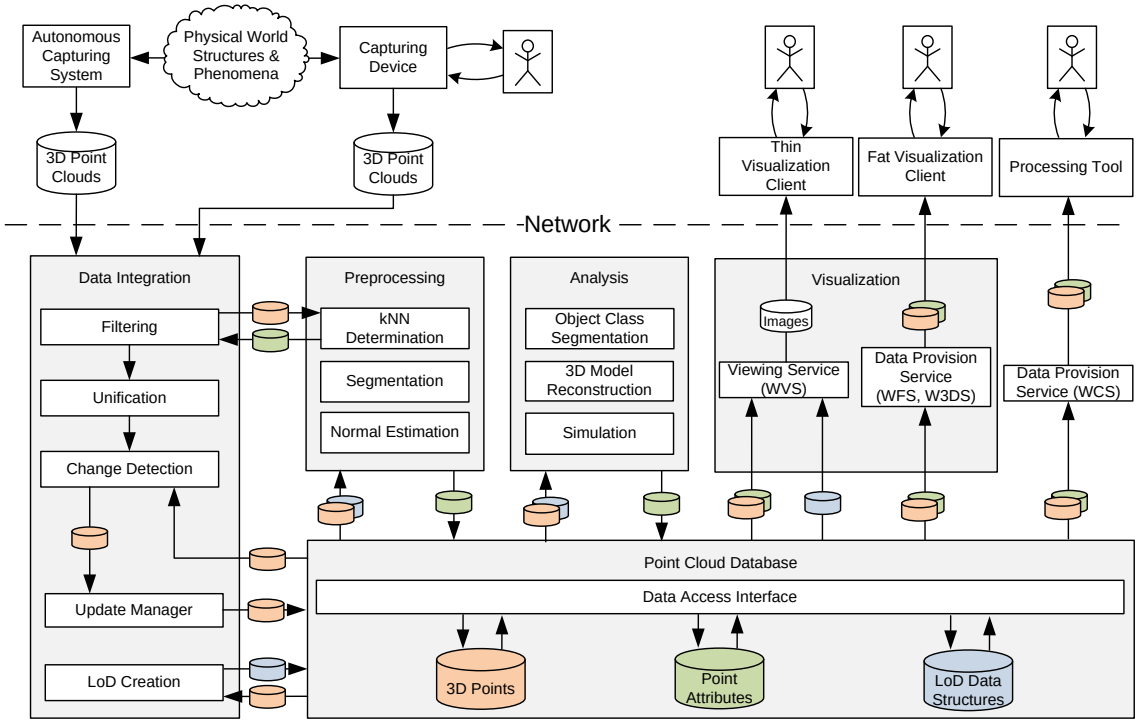


Figure 6.7: *System architecture illustrating data flow between components for data integration, processing, analysis, provision, and visualization. Data types are illustrated with different colors: 3D points - orange, point attributes - green, LoD data structures - blue.*

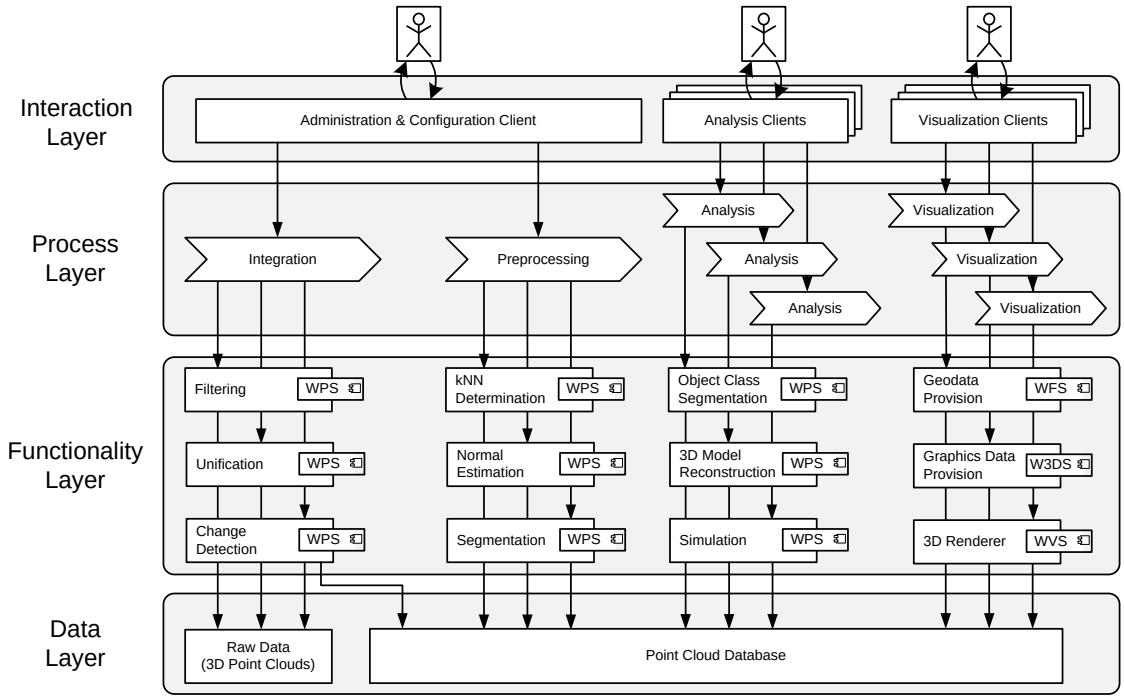


Figure 6.8: *Service-oriented architecture of the presented system. Functionalities are provided by Web Processing Services (WPS), Web Feature Services (WFS), Web 3D Services (W3DS), and Web View Services (WVS). Arrows indicate the usage relation between components.*

Chapter 7

Case Studies and Applications

This section presents case studies that have been implemented, performed, and evaluated based on concepts and techniques introduced in previous chapters. For each case study, a description of the application domain, core requirements and challenges, input data, and results is given and associated with the various processing, analysis, and visualization techniques that have been developed. The following case studies are presented:

- Updating 3D city models based on classification and change detection for multi-temporal 3D point clouds from aerial scans
- Monitoring of a railroad infrastructure based on 3D point clouds from mobile mapping
- Automatic tree detection and visualization based on 3D model extraction and hybrid rendering of 3D point clouds from aerial scans.

7.1 Updating 3D City Models

3D city models and, more general, geographical virtual environments (GeoVEs) are used in many disciplines and application domains such as urban planning, landscape architecture, monitoring, documentation, marketing, analysis, simulation, and disaster management (MacEachren & Kraak 2001; Döllner, Baumann & Buchholz 2006; Wolff & Asche 2008; Trinder & Salah 2011). These polygon-based 3D models can be constructed based on a portfolio of geodata (Lafarge & Mallet 2012). Commonly required data sources include 3D point clouds, orthophotos, oblique aerial images, surface models, and floor plans. However, the construction of high quality, semantic-rich, geometrically complex, well textured, and large-scale 3D city models with multiple LoDs (e.g., CityGML LoD2, LoD3) requires time-intensive, costly, and manual workflows that cannot be performed in a fully automatic way and become expensive in case of large areas and dense 3D point clouds. For that reason, it is generally not feasible (e.g., from an economic perspective) to construct 3D city models from the scratch once new geodata has been acquired. In practice, there is a huge difference between the high degree of automation for data acquisition, which can be performed in increasingly shorter time intervals, and the required manual effort to analyze and convert results in 3D city models. In the long term, approaches are required to enable continuation and update processes for 3D city models that reduce manual interventions, processing costs, and provisioning times. This section presents an approach to enable selective updates for 3D city models based 3D point clouds with classification and change detection results introduced in Chapter 3 and Chapter 4.

(a) *3D point cloud Berlin2013.* **(b)** *Old 3D city model of Berlin.*

Figure 7.1: *Illustration of the old 3D city model of Berlin that needs to be updated using the 3D point cloud of the new data acquisition.*

Table 7.1: *Overview and characteristics of the used data to update the 3D city model of Berlin.*

Name	Type	#Points	Density	Storage
Berlin2009	Aerial LiDAR	4.7 bln	5-10 pts/m^2	75 GB
Berlin2013	Image Matching	80 bln	100 pts/m^2	1.2 TB
Orthophoto *Berlin2013*	RGB and NDVI	10 cm res.	100 pts/m^2	1.2 TB
Building Footprints	Shapefile	1	527 k buildings	400 MB

7.1.1 Application Requirements and Input Data

The project aimed at updating the official 3D city model of Berlin, which was previously built with significant manual efforts. As a key requirement, buildings that have been newly constructed, significantly changed, or destructed should be detected based on the 3D point cloud taken from a new data acquisition round. In particular, the approach aimed at avoiding the complete reconstruction of all building models as a large number of details and corrections had been incorporated yet in the previous model. Based on these requirements, the approach included the following subtasks:

- R1 - Detection of buildings in the 3D city model, i.e., cadastre that cannot be found in the 3D point cloud (outdated buildings).
- R2 - Detection of buildings with structural changes and, therefore, which need to be updated in the 3D city model (changed buildings).
- R3 - Detection of buildings that can be found in the 3D point cloud but are not present in the 3D city model (new buildings).

As a further requirement, this analysis should be available and provide the results, in this case for the urban area of Berlin, within weeks.

7.1.2 Process

The change detection process required to update 3D city models can be divided into the following tasks, which are illustrated in Figure 7.2. First, information from aerial color and infrared images is added to the input 3D point cloud *Berlin2013* (Chapter 5.3).

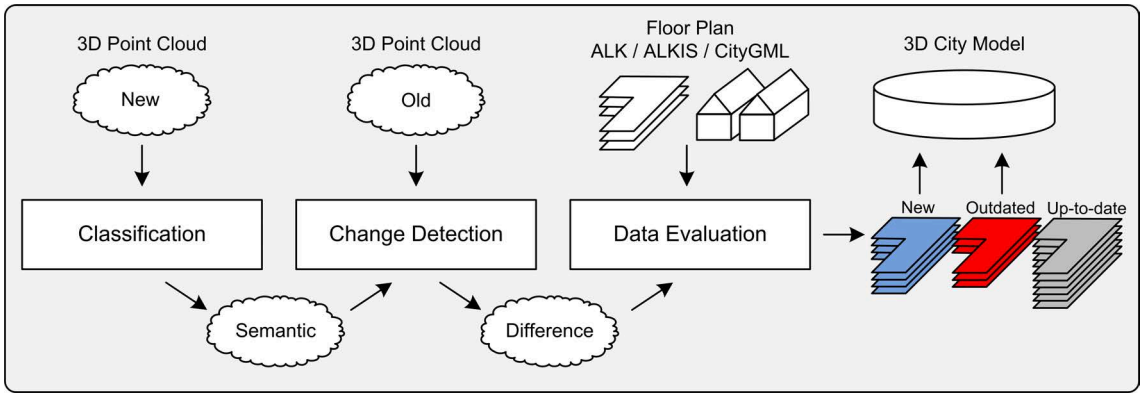


Figure 7.2: *Illustration of the workflow and processing pipeline used to update 3D city models.*

Second, both 3D point clouds are partitioned into tiles (e.g., 500 x 500 meters) to enable sequential processing, e.g., batch jobs on multiple processing systems. All tiles have overlapping borders of 50 meters to guarantee correct processing results for points close to tile borders. Third, the classification is performed for *Berlin2013* to detect all building points (Chapter 3). Forth, change detection is performed with *Berlin2013* as target cloud and *Berlin2009* as reference cloud (Chapter 4). The next steps are specific for the presented application and use classification and change detection information from previous passes. To fulfill requirement R1 and R2, each building footprint of the 3D city model is compared to the 3D point cloud *Berlin2013*. All points that are above a footprint are determined and analyzed to generate attributes for the footprint with the following information:

- Number of building, vegetation, and ground points
- Ratio of building, vegetation, and ground points
- Number of changed points (e.g., distance $>$ 1 meter)
- Ratio of changed points (e.g., 50 percent)
- Mean change of all points (e.g., 5 meter)
- Maximal and minimal change of all points (e.g., 20 meter and 3 meter)
- Changed area (e.g., 100 m^2) and change volume (e.g., 2000 m^3).

Buildings that need to be removed from the 3D city model (requirement R1) can be identified by determining all footprints for which the building ratio is below a defined threshold (e.g., 60 percent). The threshold depends on the characteristic of the urban area. For example, buildings in suburban areas where trees cover roof parts do not have a building ratio of 100 percent. The determination of footprints belonging to different coverage intervals (e.g.,$[70 - 80]$) is used to facilitate the updating process and supports building reconstruction tools.

Buildings with structural changes (requirement R2) are identified by evaluating the parameters *ratio changed points*, *mean change*, and *change volume*. In general, changed buildings have at least 20 percent changed points and a change volume of 200 m^3. All

values below the mentioned thresholds result from different resolutions, capturing methods, measuring inaccuracies, or minor roof constructions.

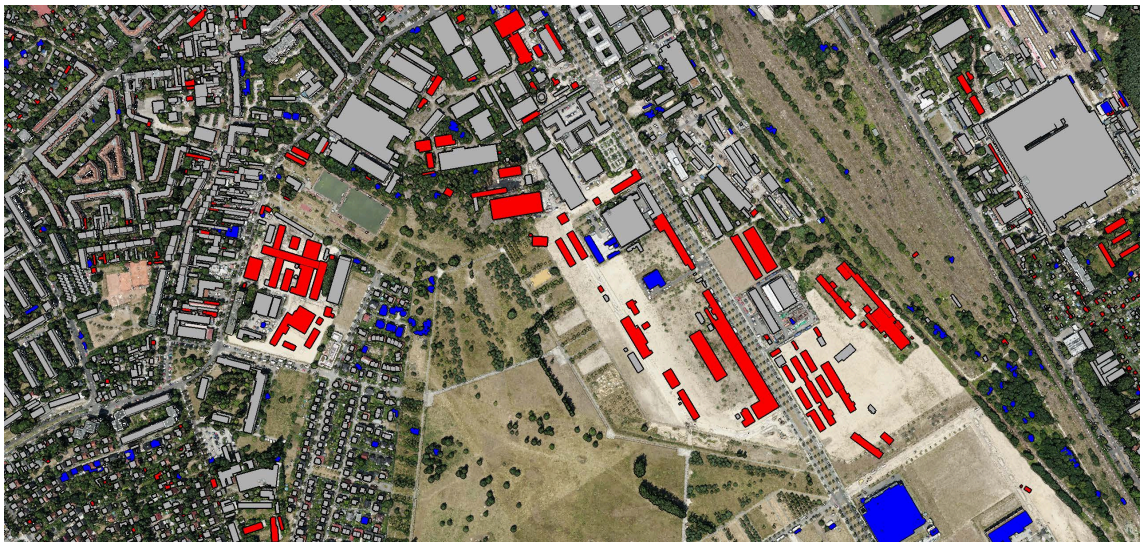
New building are not present in the cadastre or 3D city model (requirement R3), but can be detected by evaluating the classified 3D point cloud *Berlin2013*. This is done by removing all non-building points and all points above existing building footprints from the 3D point cloud. The remaining points are grouped based on the local connectivity to patches (Section 3.3.3). If a patch is larger than a defined threshold (e.g., 100 m^2) it indicates a possibly missing building, and the patch is used to compute a building outline. The outline extraction is implemented based on the approach of Zhang et al. (2006) and Zhou and Neumann (2008). For each patch, all boundary points are determined and used to construct the building outline that is in general very detailed and jagged due to the fussy structure of the input point cloud. The approach of Douglas and Peucker (1973) is used to simplify the building outlines by reducing the number of vertices. The algorithm provides smoothed as well as rectangular and orthogonal oriented surfaces (i.e., walls).

7.1.3 Results

All building outline polygons are exported in the *ESRI Shapefile* format that is a commonly used in GIS (ESRI 1998). Building footprint polygons for requirement R1 and R2 are derived from the original input shapes and attributed with additional information that indicates the degree of change (Figure 7.3). All new buildings that are derived from the classified 3D point cloud are provided as separate shape file (R3). The results were used as input for domain-specific building reconstruction and modeling tools to update the 3D city model. A comparison of the 3D city model before and after the update is illustrated in Figure 7.4.

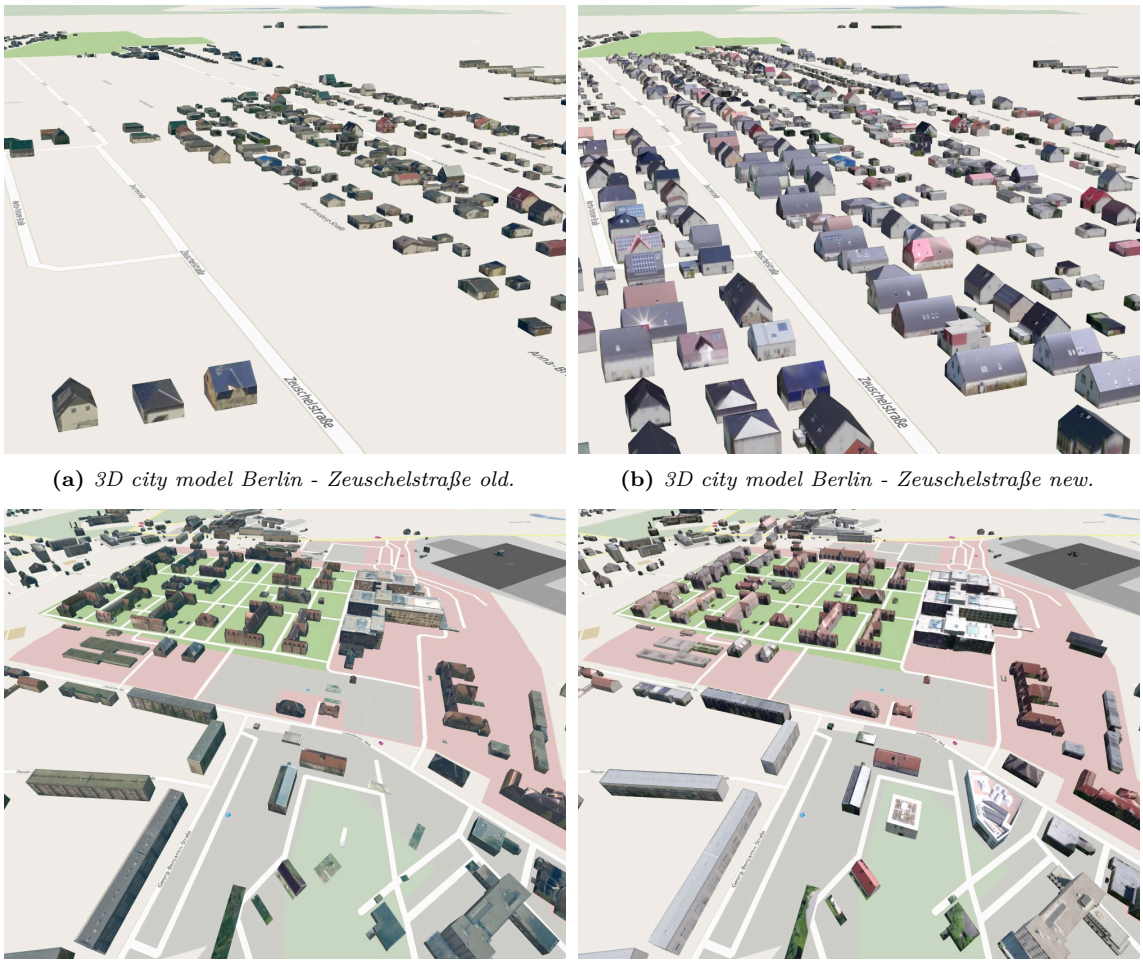
(a) *3D point cloud Berlin2013.*

(b) *Building footprints of existing 3D city model.*



(c) *Illustration of building footprints that are up-to-date (gray), new (blue), and outdated or changed (red).*

Figure 7.3: *Change detection results for the area Berlin Schöneweide.*



(a) *3D city model Berlin - Zeuschelstraße old.*

(b) *3D city model Berlin - Zeuschelstraße new.*

(c) *3D city model Berlin - campus buch old.*

(d) *3D city model Berlin - campus buch new.*

Figure 7.4: *Illustration of the 3D city model of Berlin before (left) and after (right) the update.*

7.2 Monitoring Railroad Lines

Mobile mapping systems are well established to capture the environment of infrastructure networks such as streets and railroad lines. The data acquisition can be performed in an automatic way by mounting capturing systems (e.g., laserscanners or cameras) to a vehicle used as its platform (e.g., car or train). Common applications are detection, documentation, and monitoring of obstacles, (e.g., on railroad lines), narrows (e.g., for heavy transporters), or potholes (e.g., on streets). Further applications are the object detection and categorization (Chauhan *et al.* 2014) to document the state of the environment (Golovinskiy, Kim & Funkhouser 2009). This section presents a use case and application scenario related to the field of monitoring a railroad infrastructure. The aim is to detect and categorize obstacles within a defined clearance profile in an automatic way based on 3D point clouds captured with a mobile mapping system mounted on a train.

7.2.1 Application Requirements and Input Data

The environment of railroad lines must be monitored in regular intervals for documentation, safety, and planning reasons. The data acquisition is performed with mobile mapping solutions, e.g., specific measurement trains, and is known also as *Railborn Laserscanning* (Kohut *et al.* 2012). The 3D point clouds for the presented application was captured by a the measurement train *LIMEZ III* of the DB Netz AG (Blug *et al.* 2007) shown in Figure 7.5 (a). A typical 3D point cloud of a track section containing a small train station is illustrated in Figure 7.5 (c,d). The general objective is the detection and categorization of objects and structures inside a defined *clearance profile*, e.g., shown in Figure 7.5 (b). These profiles form a volume along the track containing all structures and objects of interest for the mentioned applications. The requirements can be divided into the following subtasks:

- R1 - Filtering the input 3D point cloud and detecting outliers and duplicates in the data, which typically occur due to reflections of LiDAR rays.
- R2 - Detection of points belonging to objects inside a defined clearance profile.
- R3 - Categorization of obstacles to enable an application specific filtering, e.g., based on obstacle properties such as size, position, and height.
- R4 - Time-efficient processing workflow to analyze a set of clearance profiles for a track sections (e.g., 20 km) within a reasonable processing time (e.g., a few minutes).
- R5 - Real-time visualization of the input 3D point cloud, clearance profiles as well as processing and analysis results of R1 and R2.
- R6 - Exploration, interaction, and filtering techniques for inspection of obstacles in the context of the captured environment.

The input data is a mobile mapping point cloud with 825 million points for a track section of 68 kilometers and an average point density of 12.2 million pts/km. The clearance profile is a 2D profile composed of multiple lines and is illustrated in Figure 7.5 (b).

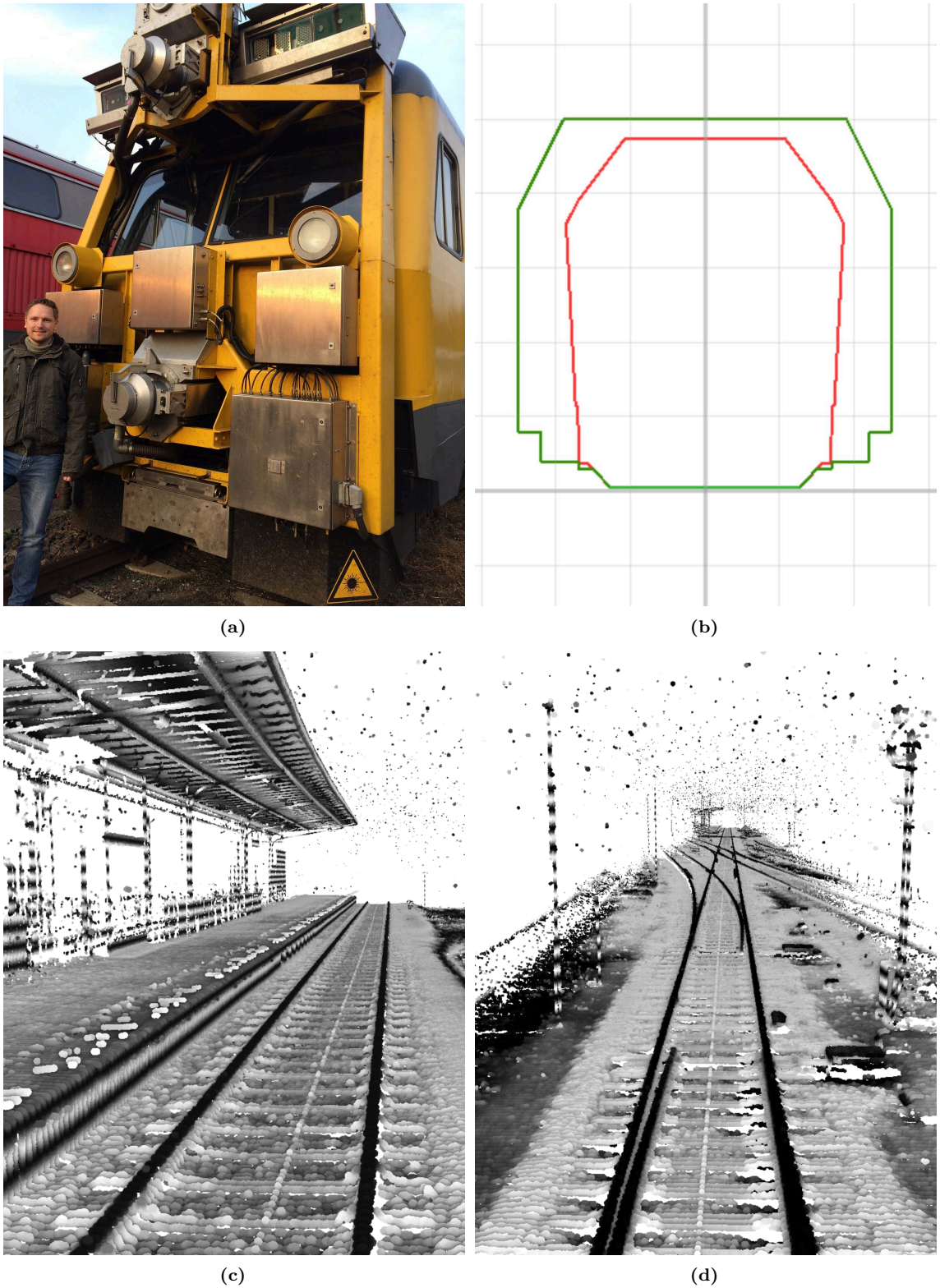


Figure 7.5: *(a) Measurement train "Limez III" of the DB Netz AG. (b) 2D clearance profile example. (c) 3D point cloud of Railroads of captured track station. (d) Typical railroad environment with a track signal.*

(a) *3D point cloud with highlighted outliers (red).* **(b)** *3D point cloud rendered without outliers.*

Figure 7.6: *Illustration of outlier detection results for the 3D point cloud of a track section.*

7.2.2 Clearance Profile Analysis

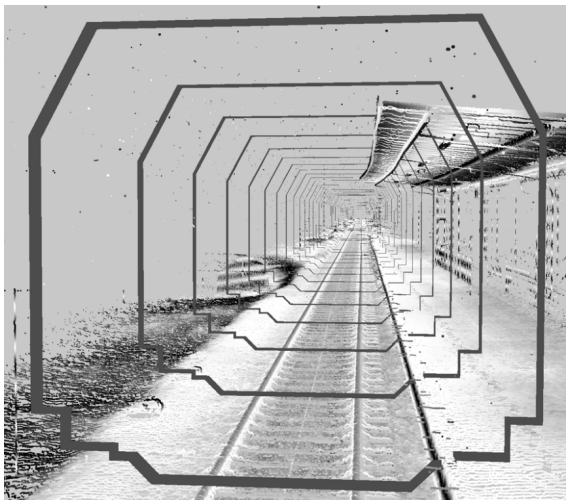
The clearance profile analysis can be divided into filtering, clearance profile comparison, and obstacle analysis tasks. First, an outlier and duplicate detection is performed to identify all points that need to be removed to clean up the data. The implemented outlier and duplicate detection process is similar to the presented approach in Section 3.3.1 and fulfills requirement R1. Figure 7.6 compares the input data with outliers and the filtered data. Second, all remaining points are compared with the clearance profile by determining the horizontal and vertical distance to a given profile (R2). This task requires the most computational effort and is performed with the change detection approach presented in Section 4 to fulfill requirement R4. In contrast to the point-to-point comparison a point-to-line computation has been implemented with a GPU-based approach. Figure 7.2.2 (a) and (b) show different clearance profiles. The results are stored as per-point attributes and can be visualized as shown in Figure 7.2.2 (c) and (d). Third, all points inside the clearance profile are determined and grouped based on the local proximity. These resulting segments are considered as obstacles that need to be categorized in the last stage. Each segment is analyzed and attributed with the following information to characterize the obstacle (requirement R3):

- #Points
- Volume
- Bounds
- Mean segment height
- Minimal distance to profile
- Maximal distance to profile

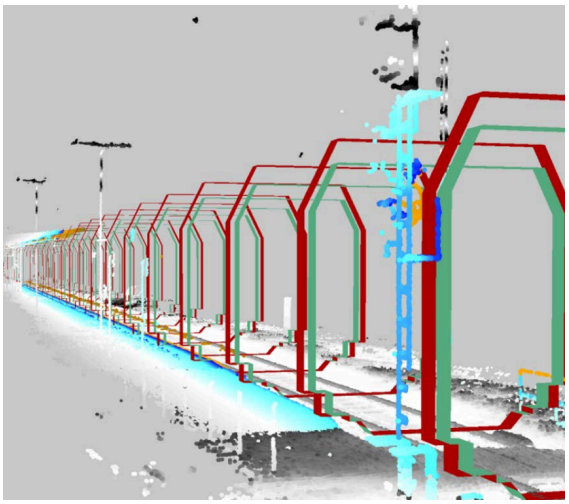
Table 7.2: *Processing performance of the clearance profile analysis of 3D point cloud Railroads.*

Length	Points	Calculation	Disk I/O	Total
1 km	10.96 mln	1.1 s	1.3 s	2.4 s
5 km	54.43 mln	5.6 s	6.0 s	11.6 s
68 km	825 mln	80 s	85 s	165 s
34.000 km	— mln	— s	— s	~23 h

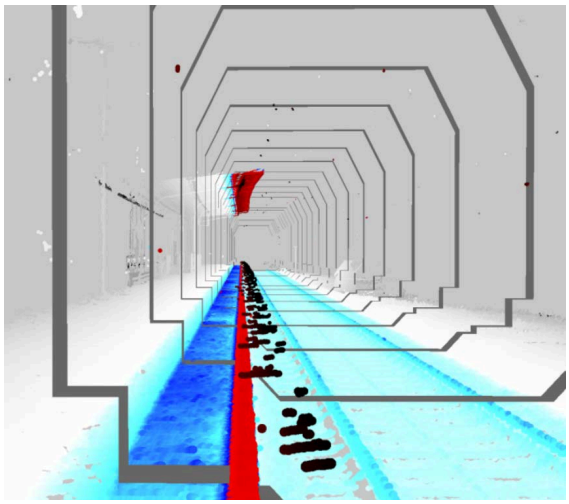
Exploration and inspection of the 3D point cloud and the processing results has been implemented based on the out-of-core real-time rendering system presented in Section 5.4 (R5). Each point is attributed with outlier, distance, and segment information as per-point attribute to enable filtering and highlighting during the interaction, shown in Figure 7.2.2 (c) and (d). Obstacle segments with related attributes are stored in a data structure used as input for the 3D visualization tool. Obstacles can be highlighted and inspected by selecting them in the 3D scene or a list view provided by the 3D visualization tool, illustrated in Figure 7.2.2 (e) and (f). The user interface provides features for filtering obstacles based on user defined requirements such as size, volume, and location. Several exploration features such as camera flights along the track or from one obstacle to another support the exploration of the data.



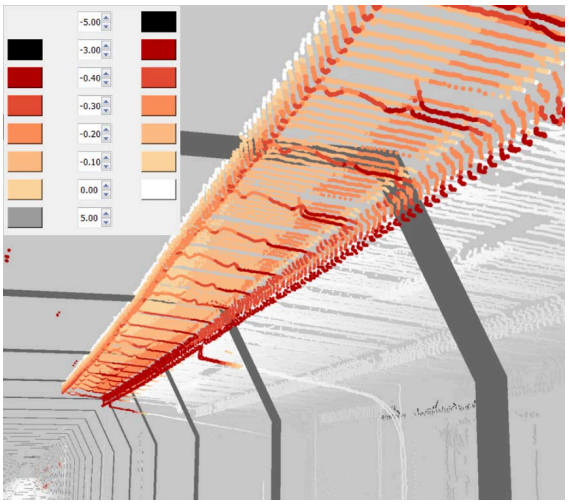
(a) *Clearance profile and 3D point cloud.*



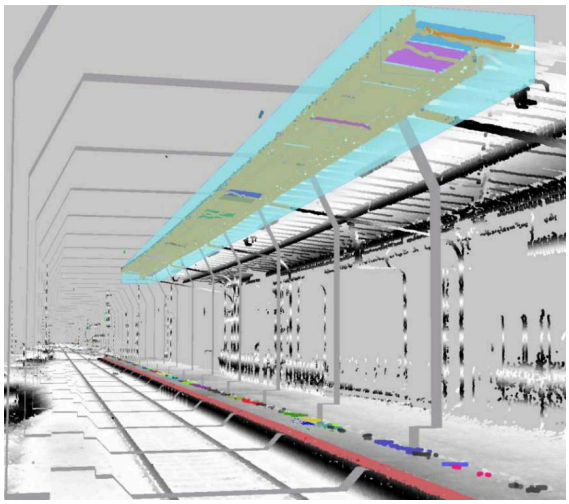
(b) *Comparison of different clearance profiles.*



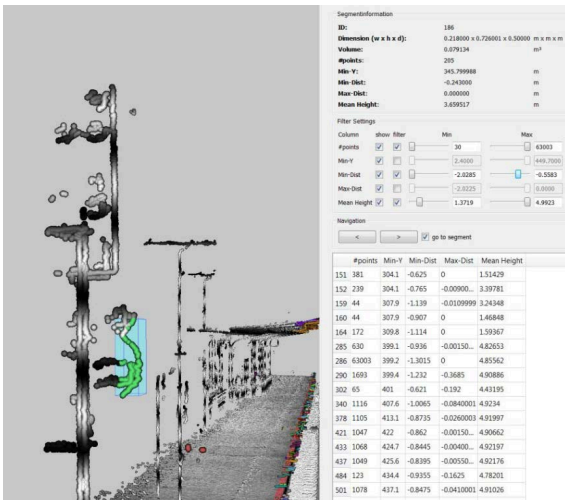
(c) *Distance analysis clearance profile and 3D point cloud.*



(d) *Visualization of points inside clearance profile.*



(e) *Segmentation of obstacles inside clearance profile.*



(f) *Obstacle property inspection.*

Figure 7.7: *Visualization of 3D point clouds from railroad track used for clearance profile analysis.*

7.3 Automatic Tree Detection and Visualization

Vegetation objects such as trees represent one main compositional element of digital 3D models of our environment required by a growing number of simulation, analysis, and visualization applications. However, a detailed representation of vegetation in 3D spatial models is generally not feasible due to the lack of up-to-date, object-based, and area-wide tree surveys.

Traditionally, geo-spatial datasets containing individual trees, called tree cadastres, are collected and maintained manually, which is an expensive and time-consuming process. To reduce the amount of work, these tree cadastres are often restricted to selected regions of interest, such as roadsides in the case of municipal tree cadastres. In addition, manually created tree cadastres frequently do not provide data about the real-world appearance and geometry of individual trees. Tree height and crown diameter, for example, are commonly omitted in a municipal tree cadastre as it would require frequent updates due to vegetation growth.

For these reasons, approaches for automatic detection, categorization, and visualization of trees within 3D point clouds are required as part of the digital transformation of related workflows. The automatic derivation of a tree cadastre based on dense 3D point clouds is a feasible and cost-efficient approach to integrate area-wide, object-based vegetation models into geographical virtual environments (e.g., 3D city models). This process does not only enhance the contents regarding vegetation, but also provides the prerequisites for further computational and simulation processes that can be built upon tree cadastres as the tree data is based on a uniform analysis without any subjective bias resulting from a manual data collection.

7.3.1 Application Requirements and Input Data

The application aims at automatically creating an area-wide tree cadastre. The input data are 3D point clouds from aerial LiDAR or dense image matching acquisition with classification results. The classification relies on points belonging to vegetation objects and excludes points belonging to other surface categories. The requirements can be divided into the following subtasks:

- R1 - Automation of data collection to create tree cadastres for arbitrarily large areas without any manual interaction by analyzing dense 3D point clouds together with aerial images.
- R2 - Single tree delineation to identify individual trees.
- R3 - Derivation of tree parameters such as tree position, height, crown diameter, volume, and typical leaf color.
- R4 - Visualization and real-time rendering of tree cadastres.

7.3.2 Automatic Tree Detection

In the context of this thesis, tree detection is a data processing task that generates an object-based tree cadastre for a geographic area based on a dense 3D point cloud (e.g., by LiDAR or dense image matching) and related four-channel orthoimages. Various approaches

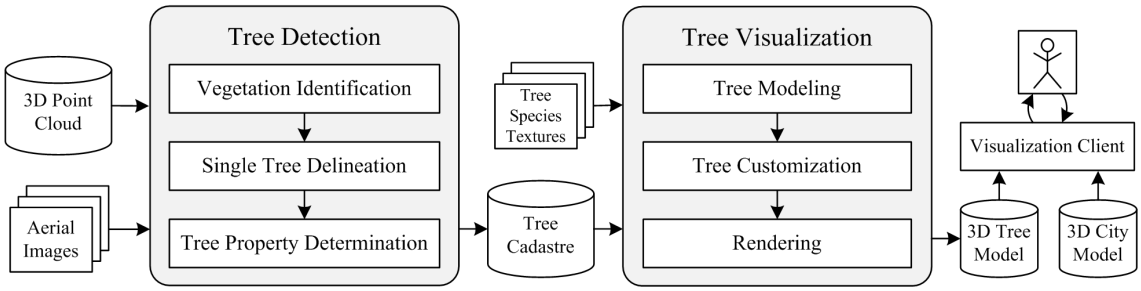


Figure 7.8: *Overview of tree detection and visualization pipeline.*

have been presented in recent years (Jakubowski *et al.* 2013) that automatically extract tree positions from 3D point clouds and aerial images, along with additional geometrical parameters (Edson & Wing 2011), such as tree height and crown diameter. Further, four-channel aerial images (RGBNIR) can be used to obtain information about tree species (Zhang & Hu 2012). However, the determination of species is out of the scope of this work. Tree detection involves three main consecutive steps:

1. Vegetation identification
2. Single tree delineation
3. Tree property determination

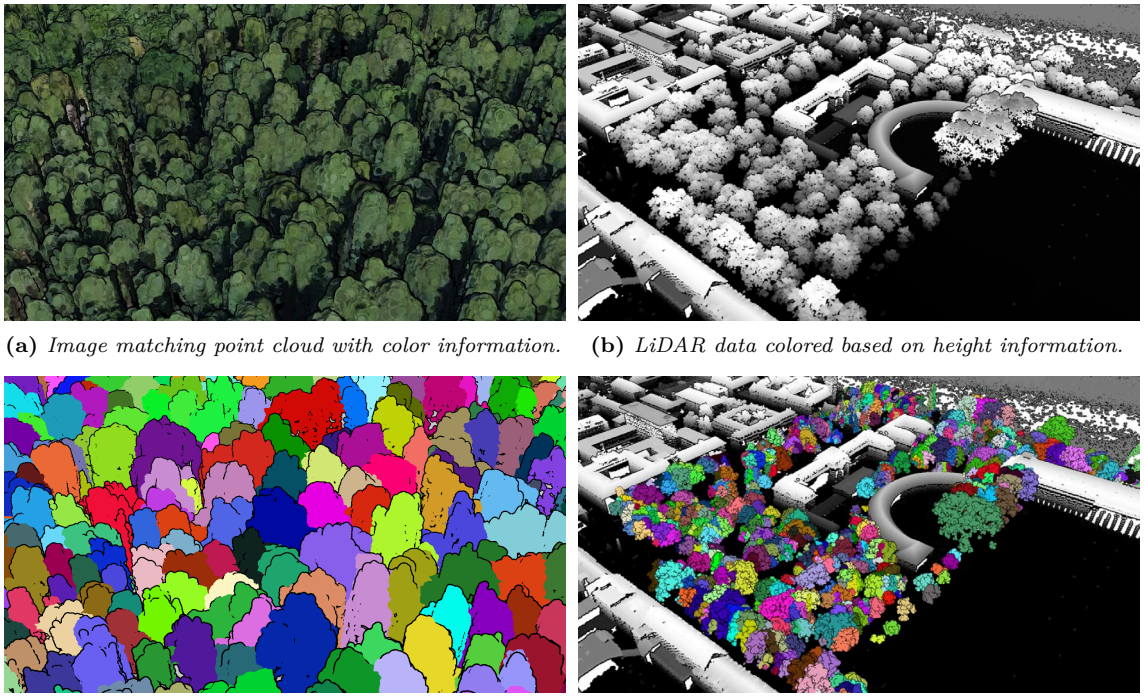
The common processing workflow is illustrated in Figure 7.8. Results can be used for a variety of applications such as urban monitoring, tree cadastre maintenance, analysis, or visualization.

Vegetation Identification

The first step is a classification of the input data, which is described in detail in Chapter 3 and in Richter et al. (2013). It serves to distinguish vegetation and non-vegetation points. The classification can be performed automatically for arbitrarily large urban areas (R1) and supports subsequent processing stages to operate on a subset of the input data, i.e., vegetation points. In addition, identified ground points are used to derive a digital terrain model used as an elevation filter as well es to determine the tree height. Typically, only vegetation points located at least 2.0 m above the ground are relevant and reliable for a tree detection.

Single Tree Delineation

The second step operates on detected and filtered vegetation points and separates individual tree objects from each other (R2). Existing methods for single tree delineation can be classified into image-based and point-based approaches. Image-based approaches convert the 3D point cloud into a 2D height map and apply image processing algorithms on this height map, such as inverse watershed segmentation (Reitberger *et al.* 2007) or edge detection (Chen & Zakhor 2009). In contrast, point-based approaches operate on the 3D point data itself and use geometric approaches to delineate single trees, such as RANSAC methods (Tittmann *et al.* 2011) or iterative, top-down classification based on horizontal spacing rules (Li *et al.* 2012b).



(a) *Image matching point cloud with color information.*

(b) *LiDAR data colored based on height information.*

(c) *Single trees visualized with a random color.*

(d) *Single trees visualized with a random color.*

Figure 7.9: *3D point clouds from dense image matching (left) and LiDAR (right) used for tree delineation. Illustration of data characteristic (top) and each identified tree visualized with a random color (bottom).*

The tree delineation algorithm presented in this section is based on the approach of Li et al. (2012). It is a new, point-based approach and requires only a single iteration over the 3D point cloud. During this iteration, each point in the 3D point cloud is assigned a tree ID, while all points belonging to the same tree are assigned the same tree ID. To take advantage of the horizontal spacing between treetops, the algorithm starts with the highest point in the 3D point cloud. For the current point p, a sub procedure searches for the closest tree point q in the local neighborhood, i.e., the closest point that has already been assigned a tree ID, while the radius of this neighborhood linearly depends on the elevation of p above the ground. If such a point q exists in the local neighborhood, p belongs to the same tree as q and is therefore assigned the same tree ID. Otherwise, if there is no such point q in the local neighborhood, p is assigned a new tree ID, which is typically the case for the highest point within a tree. Figure 7.9 shows tree delineation results for dense image matching and LiDAR data.

Tree Property Determination

For each detected tree, geometric and spectral properties are derived that describe the *individual tree characteristics*. These parameters (R3) are important for statistical analysis, monitoring, and visualization tasks (R4). Geometric properties are extracted from the 3D point cloud and include position, height (elevation of the treetop relative to the ground), crown diameter, and crown cover area for each tree. Spectral properties are extracted from the related aerial image of the tree crown and include mean band ratios, mean NDVI, and mean channels in HSV color space. The latter is used to identify the typical leaf color

of individual trees leading to more realistic virtual appearance. To minimize the effect of shadows a shadow index (SI), introduced by Ono et al. (2010), is applied to identify shadowed regions on the tree crown, and those areas are ignored during the computation of spectral properties. As an additional application, spectral properties are useful to predict individual tree species. Figure 7.10 illustrates a tree cadastre and related properties for an urban area.

7.3.3 Tree Cadastre Visualization

The visualization and exploration of tree cadastres by geographical virtual environments requires real-time rendering techniques (R4). To address this issue, *instancing* is used as optimization technique to render a large number of similar tree objects that are customized through their object-specific parameters (Bao *et al.* 2011).

The input of the tree visualization component is an *object-based tree cadastre* that specifies the position and characteristics of each tree and a set of 2D tree textures that contain the image of the tree from three different points of view (Figure 7.11 (a-c)). To generate a 3D tree model, these textures are mapped onto separate, semi-transparent, intersecting quads as geometric primitives, commonly denoted as crossed billboards (Zhang *et al.* 2006) (Figure 7.11 (d-e)). Crossed billboards leverage the capabilities of modern texturing hardware to create visually complex 3D models that can be efficiently rendered. However, if viewed at close range (Figure 7.11 (d-e)) or from an elevated position (Figure 7.11 (e)), crossed-billboard trees appear unrealistic due to the plane-like carrier structures of the quads which results in particular in parallax artifacts. To increase the visual complexity of nearby trees, billboard planes can be dynamically tessellated into a polygonal mesh that is deformed using vertex displacement. The resulting tree models (Figure 7.11 (f-g)) exhibit a more detailed, volumetric, and unstructured appearance leading to less parallax artifacts. Tessellation and vertex displacement are performed on the GPU and are only applied to tree models close to the viewer. To render large collections of 3D tree objects efficiently, instancing is used as a means of minimizing the communication overhead between CPU and GPU. Moreover, trees outside the view frustum are excluded from rendering to increase performance. To obtain a more diverse and realistic appearance, tree models are customized by their individual characteristics defined in the tree cadastre such as tree height and typical leaf color. Color transfer from the aerial image to the tree texture takes place in HSV color space and involves the following tasks:

- The tree detection process extracts the mean hue (*h1*), saturation (*s1*), and value (*v1*) channels from each tree crown in the related region of the aerial image; these results are stored as properties per-tree attributes (Figure 7.10).
- A preprocessing step extracts the mean value channel (*v2*) of all pixels belonging to leafs of a tree texture.
- The 3D rendering engine generates tree texture for each tree on-demand by using the per-tree attributes (mean hue (*h1*) and saturation (*s1*)) and the value channel of the tree species texture. The new pixel color is computed by *v1* – *v2* to have an appearance that looks like the tree texture with a characteristic color similar to the real tree.

(a) *2D illustration of tree cadastre. Tree positions are illustrated as red circles over the aerial image.*

ID	X	Y	Z	HEIGHT	AREA	VOLUME	diameter	NIR_ratio	R_ratio	G_ratio	B_ratio	meanNDVI	meanGRI	meanH	meanS	meanV
1	11077.859375	36589.867188	34.574821	24.174183	38.408363	6382.721680	12.225761	0.427899	0.171263	0.251439	0.149399	0.428998	0.190539	0.303833	0.407666	0.405436
2	11085.649414	36689.855469	34.470001	2.882927	4.825389	6393.087891	1.535969	0.392317	0.185831	0.260629	0.161223	0.362706	0.172655	0.303530	0.383946	0.477820
3	11068.016602	36590.792969	34.564438	26.158672	70.617294	7854.318848	22.478184	0.414167	0.182921	0.243466	0.159447	0.399067	0.154980	0.299156	0.351967	0.404004
4	11233.459961	36725.082031	34.972347	2.253052	7.995431	7859.227539	2.545025	0.341153	0.217324	0.278259	0.163264	0.243790	0.139808	0.260258	0.422685	0.563836
5	11079.750000	36700.539063	34.467731	14.357597	79.315971	8670.169922	25.247057	0.386286	0.221133	0.263300	0.129280	0.286038	0.099599	0.221488	0.515620	0.518830
6	11121.541016	36679.667969	34.555191	2.045975	11.771352	8685.430664	3.746938	0.366743	0.229204	0.272542	0.131511	0.242699	0.095736	0.221230	0.515979	0.541982
7	11143.336914	36670.730469	34.714821	2.139118	14.530262	8705.996094	4.625126	0.358058	0.240134	0.274457	0.127350	0.204564	0.069759	0.206778	0.531311	0.575910
8	11218.727539	36672.527344	35.054535	6.812069	5.398809	8727.123047	1.718494	0.381394	0.208315	0.264448	0.145842	0.306852	0.128806	0.247769	0.449924	0.517176
9	11045.197266	36572.085938	34.631935	2.188786	5.387990	8735.536133	1.715050	0.343590	0.243927	0.250081	0.162401	0.183518	0.022266	0.186747	0.360254	0.498356
10	11062.981445	36691.566406	34.429066	4.062965	6.340085	8743.706055	2.018112	0.291210	0.270966	0.266247	0.171577	0.043342	-0.002294	0.171361	0.373373	0.557975
11	11106.987305	36522.496094	34.700001	3.007000	16.272163	8771.252930	5.179591	0.393396	0.196877	0.267714	0.142013	0.342187	0.160482	0.271330	0.472763	0.475112
12	11147.090820	36567.812500	34.747456	9.266506	11.208751	8840.861328	3.567856	0.410167	0.169196	0.245800	0.174837	0.418512	0.185095	0.350201	0.328356	0.390292
13	11063.917969	36742.304688	34.428955	2.230885	8.633767	8848.923828	2.748214	0.291013	0.275507	0.263375	0.170105	0.028034	-0.022401	0.148081	0.382437	0.638909
14	11162.313477	36583.242188	34.804634	3.055672	5.193244	8856.110352	1.653061	0.412211	0.186417	0.258325	0.143047	0.387759	0.170864	0.282901	0.442696	0.452022
15	11057.179688	36506.812500	34.730053	6.163193	25.890482	8945.968750	8.241197	0.411570	0.188878	0.257155	0.142398	0.376145	0.154513	0.270884	0.439286	0.449937
16	11162.304688	36562.433594	34.806179	17.404655	94.657677	10002.708984	30.130474	0.416950	0.183096	0.255095	0.144859	0.393314	0.168147	0.280911	0.430597	0.436317
17	11000.956055	36696.753906	34.400002	5.129997	29.655584	10108.168945	9.439666	0.399821	0.189358	0.258412	0.152409	0.362802	0.158284	0.283488	0.409251	0.434778
18	11062.597656	36618.250000	34.499634	17.220921	29.331005	10459.604492	9.336349	0.419943	0.183670	0.233866	0.162521	0.402229	0.134058	0.292640	0.311922	0.393522
19	11123.910156	36544.949219	34.705967	3.376354	19.766783	10493.394531	6.291962	0.419393	0.185593	0.254087	0.140927	0.391272	0.159352	0.270644	0.442166	0.436626

(b) *Sample of a tree cadastre shapefile. Each line represents an individual tree with related attributes such as position, geometric properties, and color information.*

Figure 7.10: *Results of single tree delineation (a) and tree property determination (b) for the campus area of the Hasso Plattner Institute.*

(a) **(b)** **(c)** **(d)** **(e)** **(f)** **(g)**

Figure 7.11: *Basic principle of instancing-based tree rendering. The input is a small set of 2D tree textures (a-c) used to form a crossed-billboard tree (d-e). To prevent parallax artifacts at close range, trees are adaptively tessellated (f-g).*

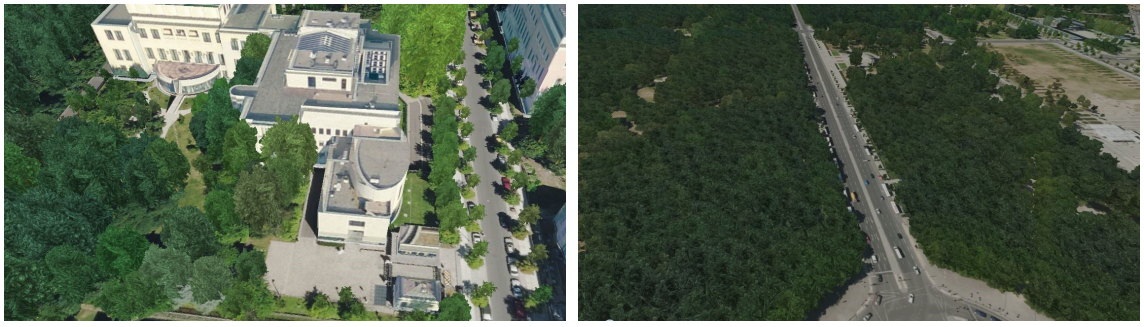
7.3.4 Results

The presented concepts and techniques have been implemented as part of the software framework for 3D point clouds. An evaluation of tree detection accuracy and visualization performance was performed on a desktop computer with an Intel Core i5 CPU at 3.30 GHz (4 cores and 4 threads), 8 GB of main memory, and a NVIDIA GeForce GTX 760 with 2 GB of video memory. The input 3D point cloud was generated through dense image matching and has a resolution of 100 points/m^2. The automatically generated tree cadastres were compared with manually classified reference data for two selected regions shown in Figure 7.12. The suburban area (Figure 7.12 (a)) contains a significant number of isolated, small-sized and medium-sized trees (up to 10 m) and some large, grouped trees. The urban forest (Figure 7.12 (b)) is mainly constituted by large, grouped trees (above 15 m). For three different height categories, the achieved precision, recall, and F-Measure were calculated as shown in Table 7.3. In both regions, small, isolated trees (below 10 m) are robustly detected, as indicated by the high recall of 97.4% for the suburban area and 92.3% for the urban forest. The least number of false positives occurred in the highest tree category (above 15 m), resulting in the highest precision of 80.3% for the suburban area and 91.1% for the urban forest. In the suburban test area, the precision is mainly affected by dense image matching errors associated with city furniture, such as poles and traffic signs, resulting in a number of false positives.

Figure 7.9 shows a visualization of single tree delineation results for a dense forest area. Most tree crowns were separated correctly, as indicated by different colors, but the automatic approach fails at separating tree crowns that are tightly interconnected, creating some degenerated large segments. This is especially due to the limited ability of 3D point clouds generated by dense image matching to describe the complete, three-dimensional structure of dense vegetation. For visualization purposes, however, the uncorrected tree cadastre may already be sufficient to derive a plausible and realistic model of the overall vegetation. Figure 7.13 shows two visualizations of an automatically generated tree cadastre for an urban area as part of a virtual 3D city model. Figure 7.13 (a) demonstrates how individual tree models are scaled to their individual height, as indicated by the group of old trees on the left and the small roadside trees on the right. Moreover, different leaf colors are observable that were extracted from the aerial image and mapped onto the individual tree crowns. Figure 7.13 (b) shows an urban forest from a distant view. Tree rendering performance has been evaluated on a synthetic, randomly generated tree cadastre, a window resolution of 1920 x 1080, and two fixed camera orientations. During the test, the amount of trees was gradually increased to investigate the impact on the achieved frame rate in frames per second (FPS), as illustrated in Figure 7.14. Three different render settings were compared to each other: First, all trees were rendered with full tessellation (294 triangles per tree) and without culling. Second, adaptive tessellation was enabled to reduce the geometric complexity for distant trees to a minimum of 6 triangles per tree. Third, adaptive tessellation was supplemented with individual tree culling, further reducing the rendering overhead for trees outside the view frustum. The first setting shows similar performance for both camera orientations, achieving a frame rate of 10 FPS for up to 60k.

(a) *Isolated trees.* **(b)** *Urban forest.*

Figure 7.12: *Overview of the two test areas with mostly isolated, urban trees (a) and urban forest (b).*



(a) *Isolated trees.* **(b)** *Urban forest.*

Figure 7.13: *Automatically generated tree models visualized by a virtual 3D city model.*

Table 7.3: *Tree detection accuracy measured as precision, recall, and F-Measure.*

Tree height (m)	Count	Prec.	Recall	F-Meas.
Suburban area				
$[2; 10)$	214	71.9%	97.4%	82.7%
$[10; 15)$	115	69.6%	84.5%	76.3%
$[15; \infty)$	160	80.3%	72.1%	76.0%
$[2; \infty)$	489	73.7%	85.3%	79.1%
Urban area				
$[2; 10)$	61	62.1%	92.3%	74.2%
$[10; 15)$	16	78.6%	84.6%	81.5%
$[15; \infty)$	341	91.1%	84.4%	87.6%
$[2; \infty)$	418	86.0%	85.3%	85.6%

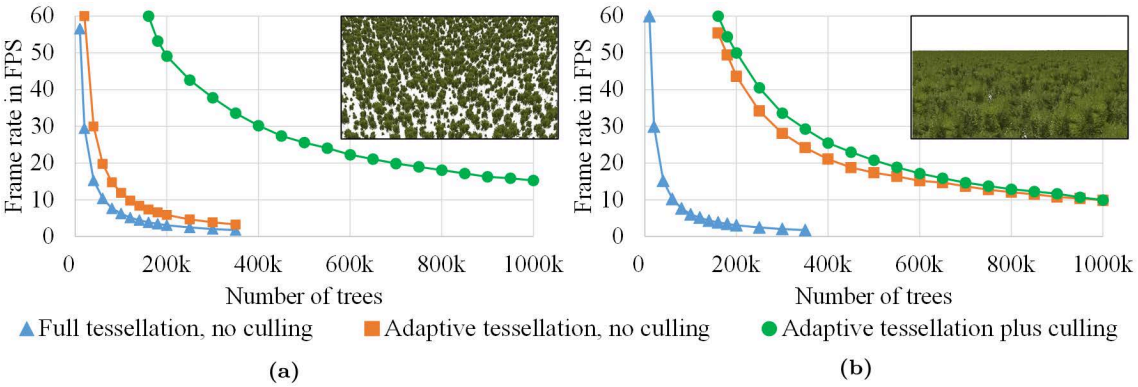


Figure 7.14: *Tree scene used for performance evaluation (a) and frame rate depending on the number of rendered trees (b).*

Chapter 8

Conclusions

How can physical surroundings be captured in an efficient way? How can 3D point clouds become the central source to generate digital twins? How can we use workflows based on 3D point clouds to document, monitor, and identify spatial objects and environments? How can spatial analytics based on 3D point clouds help us to draw conclusions and get insights?

This thesis contributes to these questions. It shows that 3D point clouds serve as a universal digital representation for spatial 3D models, provided that storage, processing, analysis, and visualization are based on efficient and reliable algorithms and data structures that reflect the specific characteristics of 3D point clouds. In that respect, this thesis presents a number of concepts and solutions to selected challenges, which contribute to establish 3D point clouds as a fundamental and essential category of 3D spatial data. Obviously, there was, is, and still will be demand for polygonal 3D modeling approaches (e.g., digital terrain models, 3D city models). Nevertheless, the main attraction and key strengths of 3D point clouds include their ability to represent *any geometry* – regardless of their shape, topology or morphology – and provide a *uniform digital representation*. A number of today's systems and applications use polygonal 3D models as a compromise because they cannot adequately handle raw 3D point data.

3D point clouds are already used in a variety of application domains and typically generated or captured by ground-based or airborne remote sensing systems at high precision and density. Novel technologies, including image-based 3D reconstruction, more and more allow for real-time 3D point cloud acquisition that will produce dense 3D point clouds by standard hardware. For example, smartphones will become de-facto handheld scanners to capture the environment in 3D using stereo cameras and image-matching; small and autonomous unmanned aerial vehicle (UAVs) will be used to capture sites and interiors fully automatically. Hand in hand with these developments, the increasing amount of data demands for *specialized storage, processing, and visualization software solutions* that are designed for and driven by the specific properties of 3D point clouds. Most traditional systems, such as geoinformation systems, will most likely not be able to fully cope with these specific needs as their data structures and processing paradigms as well as their storage and systems architectures have not been designed with 3D (or even 4D) point clouds in mind. Furthermore, there is not only a need for *parallel and GPU-accelerated processing as well as out-of-core processing concepts*, there is also a strong need for *distributed, service-oriented systems* to enable the seamless integration of 3D point cloud services into arbitrary IT workflows and services.

The framework developed throughout this thesis shows that components to manage, process, and visualize massive 3D point clouds can be efficiently implemented in terms of a

fine-grained service family. For their implementation, appropriate spatial data structures have been developed to cope with massive amounts of data as presented in Chapter 2. Concepts and techniques for the classification turned out to be essential to attribute point cloud data with semantics information. This way, applications and systems get a far more effective access to the information contained and represented by 3D point clouds – a key step to reduce their implementation complexity as they do not have to take care of the geo-specific preprocessing, processing, and classification phases (Chapter 3). Another main high-level functionality provided for 3D point clouds represents massive parallel processing using GPU-based processing, and, as key application, this allows for 3D difference analysis among point clouds taken for the same region at different points in time (Chapter 4). Change detection appears to have a still untapped potential for both traditional spatial as well as non-spatial fields of applications.

Complementary, the framework also developed a number of out-the-box visualization approaches for massive 3D point clouds that are based on the spatial data structures used to implement an out-of-memory data management. In particular, the developed 3D rendering engine enables to select, combine, and configure point-based rendering techniques. It allows to adjust the appearance of the captured environment (i.e., based on semantics information), leads to an improved visual representation, enhances recognition of objects within 3D point cloud depictions, and facilitates visual filtering and highlighting (Chapter 5). All concepts and techniques have been implemented and evaluated within the thesis' framework and demonstrate the feasibility and applicability for common real-world data sets (Chapter 6). In addition, the thesis verifies the relevance of classification, change detection, and visualization techniques with case studies and related projects (Chapter 7).

From a software engineering point of view, one essential requirement to build complex software systems based on 3D point clouds represent service-oriented architectures (SOA). The work in this thesis is based on a strong decomposition of functionality into components that can be deployed within SOA-based IT landscapes in terms of services. These components have been designed in a modular way to operate in arbitrary IT solutions, without assuming a specific GIS. This, in particular, is important as the bigger part of future applications will use 3D point clouds as spatial data source for purposes in completely different domains (e.g., insurance, facility management, agriculture, security), which commonly need that level of abstraction regarding geodata management. Insofar, the presented system architecture can serve as general framework for designing, implementing, and operating generic services based on 3D point clouds.

8.1 Future Work

The presented concepts and techniques focussing on 3D point clouds provide the basis for a number of future research directions.

- **Analytics for terrestrial 3D point clouds:** 3D point clouds from mobile mapping scans will vastly increase in the future. These datasets represent ground-aligned or near-ground assets, such as building façades, cars, traffic lights, and trees as well as static and non-static natural or manmade objects, which appear to be essential to achieve and operate many applications (e.g., monitoring, simulations, asset management). For that purpose, classification strategies for aerial 3D point clouds cannot directly be used due to different scales, data characteristics, and

relevant structures to be detected. Hence, there is a strong need for classification concepts and techniques specific to mobile mapping data to classify these "assets" of urban areas (Lehtomäki *et al.* 2016). A fusion of aerial and mobile 3D point clouds is promising and would enable to implement novel classification approaches. In addition, automotive sensor data, e.g., provided by cars, can provide spatio-temoral data (Voland & Asche 2017) for analysis tasks. Deep-learning techniques can be seen as an effective approach as they can be trained and as they are able to resolve the many ambiguities inherent to these assets and their partial representation in 3D point clouds.

- **Database technology for 3D point clouds:** A recurring task for almost all applications is efficient data access and management. A distributed and redundant data acquisition by many devices will lead to streams of 3D point clouds for overlapping areas and a large degree of redundancy. Further research should focus on infrastructures and spatial databases for massive 3D point clouds that could be used as a kind of "*black box*" for different applications (Cura, Perret & Paparoditis 2017; van Oosterom *et al.* 2015). Key requirements are the ability to store 3D point clouds from heterogeneous sources, captured at different points in time, with different characteristics and per-point attributes. Advantages of databases are transaction, backup, migration, concurrency, and user management mechanisms. In addition, the database must support a fast and parameterizable data selection based on different attribute layers and LoDs. A possible request could be to return all points within a defined area (e.g., along a street), belonging to a specified object class (e.g., vegetation), with a defined degree of change (e.g., 2 meter), captured at a period of time (e.g., between 2005 and 2007), with a specific NDVI value (e.g., larger than 0.5), etc. These kind of customized requests are essential to establish generic point cloud infrastructures for different industries. The general idea can be summarized in database technology for "4D point clouds".

- **Analytics for massive redundant 3D point clouds:** A continuous data acquisition, e.g., collected with sensors on vehicle fleets, results in streams of 3D points with a high degree of redundancy, for example in the case of a laser scanning system installed on garbage cars, UAVs, or trains, providing up-to-date or real-time 3D point clouds of entire cities or along infrastructure networks. A future research direction represent analytics functions that detect and handle massive redundancy, for example, by separating static and dynamic entities. In addition, a reliable detection and detailed classification of different assets such as vegetation structures (e.g., separation of leafs and brunches), city furniture, or construction site changes is required.

- **Service-based visualization and standardization:** Web-based solutions for aerial images, maps, and 3D city models benefit from OGC standards and provide interoperability. The unrestricted availability has massively promoted the use of maps by professional and private users (Engemaier & Asche 2011). Today, the first web-based rendering systems for 3D point clouds are available. These systems apply rendering capabilities of modern browsers using WebGL and OpenGL ES. The rapidly growing popularity and use of 3D point clouds with different formats, services, and applications require to develop and establish future standards for data storage, exchange, and provision. Hence, these standards will facilitate opening up

new markets and providing 3D point clouds to a large number of businesses and users.

- **Machine-learning for point cloud classification:** A key promising area for future extensions, which can be integrated in a canonical way into the presented system architecture developed in this thesis, are procedures to analyze 3D point clouds based on machine learning techniques (Qi *et al.* 2017; Yang, Zhang & Li 2017; Ao *et al.* 2017). They will allow us to derive more precise asset information, possibly with a high degree of detail and a high degree of robustness provided that appropriate training data sets are given.

In that respect, 3D point clouds exhibit their power of representation: they virtually have no limitations regarding model, geometry, structure, or topology of what is being represented. They are particularly suitable to interpret and extract those parts and objects that are of interest by an application or service. As a prerequisite, the various processing modules presented in this thesis along the extensible system architecture allow for future extensions both for generic as well as for application-specific purposes.

List of Publications

The work presented in this manuscript appeared previously in the following publications:

Journal Papers and Book Chapters

1. Discher, S., Richter, R., Trapp, M. & Döllner, J. *Service-Oriented Processing and Analysis of Massive Point Clouds in Geoinformation Management.* In *Service Oriented Mapping: Changing Paradigm in Map Production and Geoinformation Management* (eds Döllner, J., Jobst, M. & Schmitz, P.) in press (Springer, 2018). ISBN: 978-3-319-72433-1.

2. Richter, R., Behrens, M. & Döllner, J. Object class Segmentation of Massive 3D Point Clouds of Urban Areas Using Point Cloud Topology. *International Journal of Remote Sensing* **34**(23), 8408–8424. ISSN: 0143-1161 (2013).

3. Richter, R. & Döllner, J. Concepts and Techniques for Integration, Analysis and Visualization of Massive 3D Point Clouds. *Computers, Environment and Urban Systems* **45,** 114–124 (2014).

4. Richter, R. & Döllner, J. Integrierte Echtzeit-Visualisierung von massiven 3D-Punktwolken und georeferenzierten Texturdaten. *Photogrammetrie Fernerkundung Geoinformation (PFG)* **2011**(3), 145–154 (2011).

5. Richter, R., Kyprianidis, J. E. & Döllner, J. Out-of-Core GPU-based Change Detection in Massive 3D Point Clouds. *Transactions in GIS* **17**(5), 724–741. ISSN: 13611682 (2013).

6. Stojanovic, V., Richter, R., Döllner, J. & Trapp, M. Comparative Visualization Of Bim Geometry And Corresponding Point Clouds. *International Journal of Sustainable Development and Planning* **13**(1), 12 –23 (2018).

Conference Papers

7. Discher, S., Richter, R. & Döllner, J. *Interactive and View-Dependent See-Through Lenses for Massive 3D Point Clouds.* In *Advances in 3D Geoinformation 2016* (Springer International Publishing, 2016), 49–62.

8. Discher, S., Richter, R. & Döllner, J. *Konzepte für eine Service-basierte Systemarchitektur zur Integration, Prozessierung und Analyse von massiven 3D Punktwolken.* In *34. Wissenschaftlich-Technische Jahrestagung der DGPF* (2014).

9. Oehlke, C., Richter, R. & Döllner, J. *Automatic Detection and Large-Scale Visualization of Trees for Digital Landscapes and City Models based on 3D Point Clouds.* In *16th Conference on Digital Landscape Architecture (DLA 2015)* (2015), 151–160.

10. Richter, R., Discher, S. & Döllner, J. *Out-of-Core Visualization of Classified 3D Point Clouds.* In *3D Geoinformation Science: The Selected Papers of the 3D GeoInfo 2014* (2015), 227–242.

11. Richter, R. & Döllner, J. *Bestandsaktualisierung von 3D-Stadtmodellen durch Analyse von 3D-Punktwolken.* In *Tagungsband der 3-Ländertagung DGPF* (2010).

12. Richter, R. & Döllner, J. *Ein Ansatz für die Differenzanalyse zwischen 3D-Punktwolken und 3D-Referenzgeometrie.* In *31. Wissenschaftlich-Technische Jahrestagung der DGPF* (2011), 463–471.

13. Richter, R. & Döllner, J. *Out-of-core Real-time Visualization of Massive 3D Point Clouds.* In *7th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa* (2010), 121–128.

14. Richter, R. & Döllner, J. *Potentiale von massiven 3D-Punktwolkendatenströmen.* In *Geoinformatik 2012 - Mobilität und Umwelt* (2012), 215–222.

15. Richter, R. & Döllner, J. *Semantische Klassifizierung von 3D-Punktwolken für Stadtgebiete.* In *Terrestrisches Laserscanning 2012* (Wißner-Verlag, 2012), 127 –134.

Bibliography

16. Abdelhafiz, A & Niemeier, W. Developed technique for automatic point cloud texturing using multi images applied to a complex site. *IAPRS* **XXXVI,** 1–7 (2006).

17. Abdelhafiz, A, Riedel, B & Niemeier, W. Towards a 3d true colored space by the fusion of laser scanner point cloud and digital photos. *International workshop 3DArch* (2005).

18. Alexander, C., Tansey, K., Kaduk, J., Holland, D. & Tate, N. J. An Approach to Classification of Airborne Laser Scanning Point Cloud Data in an Urban Environment. *International Journal of Remote Sensing* **32**(24), 9151–9169 (2011).

19. America, M. & America, J. *Web Processing Service Best Practices Discussion Paper, Open Geospatial Consortium.* 2012.

20. Anil, E., Tang, P., Akinci, B. & Huber, D. *Assessment of Quality of As-is Building Information Models Generated from Point Clouds Using Deviation Analysis.* In *Proceedings of the SPIE Vol. 7864A, Electronics Imaging Science and Technology Conference (IS&T), 3D Imaging Metrology* (2011).

21. Ao, Z., Su, Y., Li, W., Guo, Q. & Zhang, J. One-Class Classification of Airborne LiDAR Data in Urban Areas Using a Presence and Background Learning Algorithm. *Remote Sensing* **9**(10), 1001. ISSN: 2072-4292 (2017).

22. Arabsheibani, R., Abedini, A. & Kanani Sadat, Y. Comparison of outlier detection at the edges of point clouds using statistical approach and fuzzy methodology: ground-based laser scanner field experiment and randomly simulated point cloud. *Geodesy and Cartography* **41**(3), 119–130. ISSN: 2029-6991 (2015).

23. Arikan, M., Schwärzler, M., Flöry, S., Wimmer, M. & Maierhofer, S. O-snap: Optimization-based Snapping for Modeling Architecture. *ACM Transactions on Graphics* **32**(1), 6:1–6:15. ISSN: 07300301 (2013).

24. Autodesk, *123D Catch.* http://www.123dapp.com/catch. (Last accessed 20.11.2017), 2017.

25. Awrangjeb, M. & Fraser, C. S. Automatic Segmentation of Raw LIDAR Data for Extraction of Building Roofs. *Remote Sensing* **6**(5), 3716–3751. ISSN: 2072-4292 (2014).

26. Bao, G., Meng, W, Li, H, Liu, J & Zhang, X. *Hardware instancing for real-time realistic forest rendering.* In *SIGGRAPH Asia 2011 Sketches* (2011), 6–7. ISBN: 9781450308076.

27. Barber, D. M., Holland, D. & Mills, J. P. Change detection for topographic mapping using three- dimensional data structures. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* **37**(B4), 1177–1182 (2008).

28. Barrett, E. C. & Curtis, L. F. *Introduction to Environmental Remote Sensing* 3rd. ISBN: 978-0412371707 (Springer, 1993).

29. Baumann, P. *Web Coverage Service (WCS) Interface Standard, Version 2.0.1 Open Open Geospatial Consortium Inc.* 2012.

30. Becker, C. and Häni, N. and Rosinskaya, E. and d'Angelo, E. and Strecha, C. Classification of Aerial Photogrammetric 3D Point Clouds. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* **IV-1/W1,** 3–10 (2017).

31. Beckmann, N., Kriegel, H.-P., Schneider, R. & Seeger, B. *The R*-tree: an efficient and robust access method for points and rectangles.* In *ACM SIGMOD* (1990), 322–331.

32. Bentley, J. L. Multidimensional binary search trees used for associative searching. *Communications of the ACM* **18**(9), 509–517 (1975).

33. Besl, P. J. & McKay, N. D. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **14**(2), 239–256 (1992).

34. Bettio, F., Gobbetti, E., Marton, F., Tinti, A., Merella, E. & Combet, R. *A point-based system for local and remote exploration of dense 3D scanned models.* In *The 10th International Symposium on Virtual Reality, Archaeology and Cultural Heritage VAST* (2009), 25–32.

35. Beutel, A., Mølhave, T., Agarwal, P. K., Boedihardjo, A. P. & Shine, J. A. *TerraNNI: natural neighbor interpolation on a 3D grid using a GPU.* In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (2011), 64–74. ISBN: 9781450310314.

36. Beutel, A., Mølhave, T. & Agarwal, P. K. *Natural neighbor interpolation based grid DEM construction using a GPU.* In *18th SIGSPATIAL International Conference on Advances in Geographic Information Systems* (2010), 172–181. ISBN: 9781450304283.

37. Blug, A., Baulig, C., Dambacher, M., Wölfelschneider, H. & Höfler, H. Novel platform for terrestrial 3D mapping from fast vehicles. *PIA07 - Photogrammetric Image Analysis* **36**(3/W49B)), 7–12 (2007).

38. Botsch, M. & Kobbelt, L. High-quality point-based rendering on modern GPUs. *Pacific Conference on Computer Graphics and Applications,* 335–343 (2003).

39. Botsch, M., Hornung, A., Zwicker, M. & Kobbelt, L. *High-quality surface splatting on today's GPUs.* In *Eurographics Symposium on Point-Based Graphics* (2005), 17–24. ISBN: 3-905673-20-7.

40. Boubekeur, T., Duguet, F. & Schlick, C. Rapid Visualization of Large Point-Based Surfaces. *The 6th International Symposium on Virtual Reality, Archaeology and Cultural Heritage VAST* (2005).

41. Bröring, A., Stasch, C. & Echterhoff, J. *Sensor Observation Service Interface Standard, Version 2.0 Open Geospatial Consortium Inc.* 2012.

42. Butkiewicz, T., Chang, R., Wartell, Z. & Ribarsky, W. Visual analysis and semantic exploration of urban LIDAR change detection. *Computer Graphics Forum* **27**(3), 903–910. ISSN: 01677055 (2008).

43. Callieri, M., Ponchio, F., Cignoni, P. & Scopigno, R. Virtual Inspector: a flexible visualizer for dense 3D scanned models. *IEEE Computer Graphics and Applications* **28,** 44–55 (2008).

44. Campbell, J. B. & Wynne, R. H. *Introduction to Remote Sensing* 5th. ISBN: 978-1609181765 (The Guilford Press, 2011).

45. Carlberg, M., Gao, P., Chen, G. & Zakhor, A. *Classifying Urban Landscape in Aerial Lidar Using 3D Shape Analysis.* In *16th IEEE International Conference on Image Processing (ICIP)* (2009), 1701–1704.

46. Carlberg, M., Gao, P., Chen, G. & Zakhor, A. Urban Landscape Classification System Using Airborne LiDAR (2008).

47. Cavegn, S., Haala, N., Nebiker, S., Rothermel, M. & Tutzauer, P. Benchmarking High Density Image Matching for Oblique Airborne Imagery. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* **XL-3**(September), 45–52. ISSN: 2194-9034 (2014).

48. Charaniya, A. P., Manduchi, R. & Lodha, S. K. *Supervised Parametric Classification of Aerial LiDAR Data.* In *Computer Vision and Pattern Recognition Workshop* (2004), 25–32.

49. Chauhan, I., Brenner, C., Garg, R. D. & Parida, M. A New Approach to 3D Dense LiDAR Data Classification in Urban Environment. *Journal of the Indian Society of Remote Sensing* **42**(3), 673–678. ISSN: 0255660X (2014).

50. Che, E. & J.Olsen, M. Fast ground filtering for TLS data via Scanline Density Analysis. *ISPRS Journal of Photogrammetry and Remote Sensing* **129,** 226–240 (2017).

51. Chen, G. & Zakhor, A. *2D Tree Detection in Large Urban Landscapes Using Aerial Lidar Data.* In *16th IEEE International Conference on Image Processing (ICIP)* (2009), 1693–1696.

52. Chen, Q., Wang, H., Zhang, H., Sun, M. & Liu, X. A Point Cloud Filtering Approach to Generating DTMs for Steep Mountainous Areas and Adjacent Residential Areas. *Remote Sensing* **8**(1) (2016).

53. Chen, Z., Gao, B. & Devereux, B. State-of-the-Art: DTM Generation Using Airborne LIDAR Data. *Sensors* **17**(1), 150. ISSN: 1424-8220 (2017).

54. Cheng, J., Grossman, M. & McKercher, T. *Professional CUDA C Programming* 1. Edition. ISBN: 978-1118739327 (John Wiley & Sons, 2015).

55. Clode, S., Kootsookos, P. & Rottensteiner, F. The automatic extraction of roads from LIDAR data. *ISPRS* **35**(3B), 231–236 (2004).

56. Clode, S. & Rottensteiner, F. *Classification of Trees and Powerlines from Medium Resolution Airborne Laserscanner Data in Urban Environments.* In *APRS Workshop on Digital Image Computing (WDIC)* (2005), 97 –102.

57. Coutinho-Rodrigues, J., Simão, A. & Antunes, C. H. A GIS-based multicriteria spatial decision support system for planning urban infrastructures. *Decision Support Systems* **51**(3), 720–726. ISSN: 01679236 (2011).

58. Cura, R., Perret, J. & Paparoditis, N. A scalable and multi-purpose point cloud server (PCS) for easier and faster point cloud data management and processing. *ISPRS Journal of Photogrammetry and Remote Sensing* **127,** 39–56 (2017).

59. Da Silva, C. R., Centeno, J. A. S. & Henriques, M. J. Automatic Road Extraction on Aerial Photo and Laser Scanner Data. *International Conference on Environmental and Computer Science* **19,** 99–103 (2011).

60. Dachsbacher, C., Vogelgsang, C. & Stamminger, M. Sequential point trees. *ACM Transactions on Graphics* **22,** 657–662 (2003).

61. D'Angelo, P. *Image matching and outlier removal for large scale DSM generation.* In *ISPRS Technical Commission I Symposium* (2010), 1–5.

62. Daniels, J, Ha, L., Ochotta, T & Silva, C. *Robust Smooth Feature Extraction from Point Clouds.* In *IEEE International Conference on Shape Modeling and Applications* (2007), 123 –136.

63. Dey, T., Li, G. & Sun, J. Normal estimation for point clouds: A comparison study for a Voronoi based method. *Eurographics Symposium on Point-Based Graphics,* 39–46 (2005).

64. Döllner, J., Baumann, K. & Buchholz, H. *Virtual 3D City Models as Foundation of Complex Urban Information Spaces.* In *11th international conference on Urban Planning and Spatial Development in the Information Society* (2006), 107–112. ISBN: 9783950213904.

65. Döllner, J., Hagedorn, B & Klimke, J. *Server-based rendering of large 3D scenes for mobile devices using G-buffer cube maps.* In *Web3D '12 Proceedings of the 17th International Conference on 3D Web Technology* (2012), 97–100.

66. Dong, W., Lan, J., Lianga, S., Yao, W. & Zhan, Z. Selection of LiDAR geometric features with adaptive neighborhood size for urban land cover classification. *International Journal of Applied Earth Observation and Geoinformation* **60,** 99–110 (2017).

67. Douglas, D. H. & Peucker, T. K. Algorithms for the Reduction of the Number of Points Required to Represent Digitized Line or its Caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization,* 112–122 (1973).

68. Douillard, B., Underwood, J., Kuntz, N., Vlaskine, V., Quadros, A., Morton, P. & Frenkel, A. *On the Segmentation of 3D LiDAR Point Clouds.* In *International Conference on Robotics and Automation* (2011), 2798–2805.

69. Duranleau, F., Beaudoin, P. & Poulin, P. *Multiresolution point-set surfaces.* In *Proceedings of Graphics Interface* (2008), 211–218.

70. Edson, C. & Wing, M. G. Airborne Light Detection and Ranging (LiDAR) for Individual Tree Stem Location, Height, and Biomass Measurements. *Remote Sensing* **3**(11), 2494–2528. ISSN: 2072-4292 (2011).

71. Eggert, D. & Schulze, E. C. Visualization of Mobile Mapping Data via Parallax Scrolling. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* **XL-3,** 89–94. ISSN: 2194-9034 (2014).

72. Eggert, D. & Sester, M. Multi-Layer Visualization of Mobile Mapping Data. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* **II-5/W2,** 73–78. ISSN: 2194-9050 (2013).

73. Elseberg, J., Borrmann, D. & Nüchter, A. *Efficient processing of large 3D point clouds.* In *XXIII International Symposium on Information, Communication and Automation Technologies* (2011), 1–7. ISBN: 978-1-4577-0746-9.

74. Engemaier, R. & Asche, H. *CartoService: A web service framework for quality on-demand geovisualisation.* In *ICCSA'11 Proceedings of the 2011 international conference on Computational science and its applications - Volume Part I* (2011), 329–341. ISBN: 9783642219276.

75. ESRI, *ArcGIS.* http://resources.arcgis.com/en/help/main/10.1/index.html. (Last accessed 19.01.2016), 2016.

76. ESRI, *ArcGIS System Requirements.* (Last accessed 10.11.2017), 2017.

77. ESRI, *ESRI Shapefile Technical Description.* 1998.

78. F. Samadzadegan, B. Bigdeli, M. H. Automatic road extraction from LIDAR data based on classifier fusion. *Laserscanning09,* 81–86 (2009).

79. Farber, R. *CUDA Application Design and Development.* ISBN: 978-0123884268 (Morgan Kaufmann Publishers, 2011).

80. Fernando, R. & Kilgard, M. J. *The CG Tutorial: The Definitive Guide to Programmable Real-Time Graphics.* ISBN: 78-0321194961 (Addison Wesley Pub Co Inc, 2003).

81. Filin, S. & Pfeifer, N. Segmentation of Airborne Laser Scanning Data Using a Slope Adaptive Neighborhood. *ISPRS Journal of Photogrammetry and Remote Sensing* **60**(2), 71–80. ISSN: 09242716 (2006).

82. Finkel, R. & Bentley, J. Quad trees a data structure for retrieval on composite keys. *Acta informatica* **4**(1), 1–9 (1974).

83. Ganovelli, F. & Scopigno, R. OCME: Out-of-Core Mesh Editing Made Practical. *Computer Graphics and Applications* **32**(3), 46–58. ISSN: 0272-1716 (2012).

84. Gao, Z., Nocera, L. & Neumann, U. *Visually-complete aerial LiDAR point cloud rendering.* In *20th International Conference on Advances in Geographic Information Systems* (2012), 289–298. ISBN: 9781450316910.

85. Gehrke, S., Morin, K., Downey, M., Boehrer, N. & Fuchs, T. Semi-global matching: An alternative to LIDAR for DSM generation? *International Archives of Photogrammetry and Remote Sensing* **38**(B1) (2010).

86. Girardeau-Montaut, D., Roux, M., Marc, R. & Thibault, G. Change detection on points cloud data acquired with a ground laser scanner. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Science* **36**(3/W19), 30–35 (2005).

87. Gobbetti, E., Kasik, D. & Yoon, S. Technical strategies for massive model visualization. *ACM Solid and Physical Modeling Symposium,* 405–415 (2008).

88. Gobbetti, E. & Marton, F. *Layered point clouds.* In *Eurographics Symposium on Point Based Graphics* (2004), 113–120.

89. Gobbetti, E. & Marton, F. Layered point clouds: a simple and efficient multiresolution structure for distributing and rendering gigantic point-sampled models. *Computers & Graphics* **28**(6), 815–826. ISSN: 00978493 (2004).

90. Goesele, M., Ackermann, J., Fuhrmann, S., Haubold, C., Klowsky, R., Steedly, D. & Szeliski, R. Ambient point clouds for view interpolation. *ACM Transactions on Graphics* **29**(4), 95:1–95:6. ISSN: 07300301 (2010).

91. Golovinskiy, A., Kim, G. V. & Funkhouser, T. *Shape-based Recognition of 3D Point Clouds in Urban Environments.* In *12th International Conference on Computer Vision* (2009), 2154–2161.

92. Goswami, P., Erol, F., Mukhi, R., Pajarola, R. & Gobbetti, E. An efficient multi-resolution framework for high quality interactive rendering of massive point clouds using multi-way kd-trees. *The Visual Computer* **29**(1), 69–83. ISSN: 0178-2789 (2013).

93. Goswami, P., Zhang, Y., Pajarola, R. & Gobbetti, E. *High Quality Interactive Rendering of Massive Point Models Using Multi-way kd-Trees.* In *18th Pacific Conference on Computer Graphics and Applications* (2010), 93–100. ISBN: 978-1-4244-8288-7.

94. Grilli, E., Menna, F. & Remondino, F. A review of point clouds segmentation and classification algorithms. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives* **42**(2W3), 339–344. ISSN: 16821750 (2017).

95. Gröger, G. & Plümer, L. Updating 3D city models: how to preserve geometric-topological consistency. *ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems,* 532–535 (2009).

96. Gross, M. & Pfister, H. *Point-based Graphics.* ISBN: 9780123706041 (Morgan Kaufmann Publishers Inc., 2007).

97. Guttman, A. *R-Trees: A Dynamic Index Structure for Spatial Searching.* In *ACM SIGMOD* (1984), 47–57.

98. Haala, N. *Multiray Photogrammetry and Dense Image Matching.* In *Photogrammetric Week* (2011), 185–195.

99. Haala, N. *The landscape of dense image matching algorithms.* In *Photogrammetric Week 2013* (ed Fritsch, D.) (Wichmann, Berlin/Offenbach, 2013), 271–284.

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

101. Haala, N. & Rothermel, M. Dense Multi-Stereo Matching for High Quality Digital Elevation Models. *PFG Photogrammetrie, Fernerkundung, Geoinformation* **2012**(4), 331 –343 (2012).

102. Haala, N. & Rothermel, M. Dense Multiple Stereo Matching of Highly Overlapping Uav Imagery. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* **XXXIX-B1,** 387–392 (2012).

103. Haala, N., Peter, M., Kremer, J. & Hunter, G. Mobile LiDAR mapping for 3D point cloud collection in urban areas - A performance test. *International Archieves of Photogrammetry and Remote Sensing* **62**(6), 415–433 (2008).

104. Hagedorn, B. *Web View Service Discussion Paper, Version 0.3.0 Open Geospatial Consortium Inc.* 2010.

105. Heinzel, S., Guennebaud, G., Botsch, M. & Gross, M. A hardware processing unit for point sets. *Graphics Hardware,* 21–31 (2008).

106. Höfle, B., Hollaus, M. & Hagenauer, J. Urban vegetation detection using radiometrically calibrated small-footprint full-waveform airborne LiDAR data. *ISPRS Journal of Photogrammetry and Remote Sensing* **67,** 134–147. ISSN: 09242716 (2012).

107. Hoppe, H. *Progressive meshes.* In *ACM SIGGRAPH* (1996), 99–108. ISBN: 0897917464.

108. Hoppe, H., DeRose, T., Duchamp, T., McDonald, J. & Stuetzle, W. Surface Reconstruction from Unorganized Points. *ACM SIGGRAPH Computer Graphics* **26**(2), 71–78. ISSN: 00978930 (1992).

109. Hu, X. & Yuan, Y. Deep-learning-based classification for DTM extraction from ALS point cloud. *Remote Sensing* **8**(9), 730. ISSN: 20724292 (2016).

110. Huang, H., Brenner, C. & Sester, M. A generative statistical approach to automatic 3D building roof reconstruction from laser scanning data. *ISPRS Journal of Photogrammetry and Remote Sensing* **79,** 29–43. ISSN: 09242716 (2013).

111. Huang, H., Wu, S., Gong, M., Cohen-Or, D., Ascher, U. & Zhang, H. R. Edge-aware point set resampling. *ACM Transactions on Graphics* **32**(1), 1–12. ISSN: 07300301 (2013).

112. Isenburg, M. & Lindstrom, P. *Streaming meshes.* In *Visualization* (2005), 231–238.

113. Jakubowski, M., Li, W., Guo, Q. & Kelly, M. Delineating Individual Trees from Lidar Data: A Comparison of Vector- and Raster-based Segmentation Approaches. *Remote Sensing* **5**(13), 4163–4186. ISSN: 2072-4292 (2013).

114. Jarzabek-Rychard, M. *Reconstruction of Building Outlines in Dense Urban Areas Based on LIDAR Data and Address Points.* In *Proceedings of the XXII ISPRS Congress* (2012), 121–126.

115. Jeschke, S., Wimmer, M. & Purgathofer, W. *Image-based representations for accelerated rendering of complex scenes.* In *EUROGRAPHICS 2005 State of the Art Reports* (2005), 1–20.

116. Ji, G., Shen, H.-W. & Gao, J. *Interactive Exploration of Remote Isosurfaces with Point-Based Non-Photorealistic Rendering.* In *IEEE Pacific Visualization Symposium* (2008), 25–32. ISBN: 978-1-4244-1966-1.

117. Jian, S. Y. A Method of Combing the Model of the Global Quadtree Index with Local KD- tree for Massive Airborne LiDAR Point Cloud Data Organization. *Geomatics and Information Science of Wuhan University* **39,** 918–922 (2014).

118. Jiang, J., Zhang, Z. & Ming, Y. Filtering of Airborne Lidar Point Clouds for Complex Cityscapes. *Geo-spatial Information Science* **11**(1), 21–25. ISSN: 1009-5020 (2008).

119. Jochem, A., Höfle, B., Wichmann, V., Rutzinger, M. & Zipf, A. Area-wide roof plane segmentation in airborne LiDAR point clouds. *Computers, Environment and Urban Systems* **36**(1), 54–64. ISSN: 01989715 (2012).

120. Kalasapudi, V. S., Turkan, Y. & Tang, P. *Toward Automated Spatial Change Analysis of MEP Components Using 3D Point Clouds and As-Designed BIM Models.* In *3D Vision (3DV)* (2014), 145–152.

121. Kaminsky, R., Snavely, N., Seitz, S. & Szeliski, R. *Alignment of 3D point clouds to overhead images.* In *Computer Vision and Pattern Recognition Workshops* (2009), 63–70. ISBN: 978-1-4244-3994-2.

122. Kang, Z. & Lu, Z. *The change detection of building models using epochs of terrestrial point clouds.* In *International Workshop on Multi-Platform/Multi-Sensor Remote Sensing and Mapping (M2RSM)* (2011), 1–6.

123. Katz, S., Tal, A. & Basri, R. Direct visibility of point sets. *ACM Transactions on Graphics* **26**(3), 24. ISSN: 07300301 (2007).

124. Kersten, T. P. & Lindstaedt, M. Automatic 3D Object Reconstruction from Multiple Images for Architectural, Cultural Heritage and Archaeological Applications Using Open-Source Software and Web Services. *Photogrammetrie - Fernerkundung - Geoinformation (PFG)* **2012**(6), 727–740. ISSN: 14328364 (2012).

125. Kim, E. & Medioni, G. Urban scene understanding from aerial and ground LIDAR data. *Machine Vision and Applications* **22**(4), 691–703. ISSN: 0932-8092 (2010).

126. Kim, H.-J., Öztireli, A. C., Gross, M. & Choi, S.-M. Adaptive surface splatting for facial rendering. *Computer Animation and Virtual Worlds* **23**(3-4), 363–373 (2012).

127. Kim, M.-K., Jack, C. C., Sohn, H. & Chang, C.-C. A framework for dimensional and surface quality assessment of precast concrete elements using BIM and 3D laser scanning. *Automation in Construction* **49**(Part B), 225–238 (2015).

128. Kohut, P., Mikrut, S., Pyka, K., Tokarczyk, R. & Uhl, T. Research on the Prototype of Rail Clearance Measurement System. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* **XXXIX-B4,** 385–389 (2012).

129. Kolbe, T. H. *Representing and Exchanging 3D City Models with CityGML.* In *3D geo-information sciences* (2009), 15–31.

130. König, S. & Gumhold, S. *Consistent Propagation of Normal Orientations in Point Clouds.* In (2009), 83–92.

131. Koutsoudis, A., Vidmar, B., Ioannakis, G., Arnaoutoglou, F., Pavlidis, G. & Chamzas, C. Multi-image 3D reconstruction data evaluation. *Journal of Cultural Heritage* **15**(1), 73–79 (2013).

132. Kreylos, O., Bawden, G. W. & Kellogg, L. H. *Immersive Visualization and Analysis of LiDAR Data.* In *4th International Symposium on Advances in Visual Computing* (2008), 846–855.

133. Kuntzsch, C., Sester, M. & Brenner, C. Generative models for road network reconstruction. *International Journal of Geographical Information Science* **30**(5), 1012–1039 (2016).

134. Lafarge, F. & Mallet, C. Creating Large-Scale City Models from 3D-Point Clouds: A Robust Approach with Hybrid Representation. *International Journal of Computer Vision* **99**(1), 69–85. ISSN: 0920-5691 (2012).

135. Lafarge, F., Descombes, X., Zerubia, J. & Pierrot-Deseilligny, M. Structural approach for building reconstruction from a single DSM. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **32**(1), 135–147. ISSN: 1939-3539 (2010).

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

137. Lee, D., Dinov, I., Dong, B., Gutman, B., Yanovsky, I. & Toga, A. W. CUDA optimization strategies for compute-and memory-bound neuroimaging algorithms. *Computer Methods and Programs in Biomedicine* **106**(3), 175–187 (2012).

138. Lehtomäki, M., Jaakkola, A., Hyyppä, J., Lampinen, J., Kaartinen, H., Kukko, A., Puttonen, E. & Hyyppä, H. Object Classification and Recognition From Mobile Laser Scanning Point Clouds in a Road Environment. *IEEE Transactions on Geoscience and Remote Sensing* **54**(2), 1226–1239. ISSN: 01962892 (2016).

139. Leite, P., Teixeira, J., de Farias, T., Teichrieb, V. & Kelner, J. *Massively parallel nearest neighbor queries for dynamic point clouds on the GPU.* In *Computer Architecture and High Performance Computing* (2009), 19–25. ISBN: 978-0-7695-3857-0.

140. Li, M, Li, W, Wang, J., Li, Q. & Nüchter, A. *Dynamic VeloSLAM - Preliminary Report on 3D Mapping of Dynamic Environments.* In *2012 IEEE Intelligent Vehicles Symposium (IV '12), Workshop on Navigation, Perception, Accurate Positioning and Mapping for Intelligent Vehicles* (2012), 1–6.

141. Li, W., Guo, Q., Jakubowski, M. K. & Kelly, M. A New Method for Segmenting Individual Trees from the Lidar Point Cloud. *Photogrammetric Engineering & Remote Sensing* **78**(1), 75–84 (2012).

142. Lillesand, T., Kiefer, R. W. & Chipman, J. *Remote Sensing and Image Interpretation* 7th. ISBN: 978-1-118-91947-7 (John Wiley & Sons, 2015).

143. Livny, Y., Kogan, Z. & El-Sana, J. Seamless patches for GPU-based terrain rendering. *The Visual Computer* **25**(3), 197–208. ISSN: 0178-2789 (2009).

144. Livny, Y., Pirk, S., Cheng, Z., Yanl, F., Deussen, O., Cohen-Or, D. & Chen, B. Texture-Lobes for Tree Modelling. *ACM Transactions on Graphics* **30**(4), 53:1–53:10 (2011).

145. Lodha, S. K., Fitzpatrick, D. M. & Helmbold, D. P. *Aerial Lidar Data Classification using AdaBoost.* In *Sixth International Conference on 3-D Digital Imaging and Modeling (3DIM)* (2007), 435–442. ISBN: 0-7695-2939-9.

146. Luebke, D., Harris, M., Krüger, J., Purcell, T., Govindaraju, N., Buck, I., Woolley, C. & Lefohn, A. *GPGPU: General Purpose Computation On Graphics Hardware.* In *SIGGRAPH 2005 Course Notes* (2005).

147. MacEachren, A. M. & Kraak, M. J. Research Challenges in Geovisualization. *Cartography and Geographic Information Science* **28**(1), 3–12 (2001).

148. Maltezos, E. & Ioannidis, C. Automatic Detection of Building Points From Lidar and Dense Image Matching Point Clouds. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* **II-3/W5,** 33–40. ISSN: 2194-9050 (2015).

149. Martinez-Rubi, O., Verhoeven, S., Meersbergen, M. V., Schütz, M., Oosterom, P. V., Goncalves, R. & Tijssen, T. *Taming the beast: Free and open-source massive point cloud web visualization.* In *Capturing Reality Forum 2015* (2015).

150. Masry, M. A. & Schwartzberg, P. Marine high-density data management and visualization. *Networks,* 53–58 (2009).

151. Matikainen, L. Automatic detection of changes from laser scanner and aerial image data for updating building maps. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Science* **35**(B2), 434–439 (2004).

152. Matikainen, L., Karila, K., Hyyppä, J., Litkey, P., Puttonen, E. & Ahokas, E. Object-based analysis of multispectral airborne laser scanner data for land cover classification and map updating. *ISPRS Journal of Photogrammetry and Remote Sensing* **128,** 298–313. ISSN: 09242716 (2017).

153. Mayer, H., Sester, M. & Vosselman, G. *Basic Computer Vision Techniques.* In *Manual of Photogrammetry. American Society for Photogrammetry and Remote Sensing (ASPRS)* 6th ed., 517–583 (Bethesda, 2013). ISBN: 1-57083-099-1.

154. Meagher, D. Geometric modeling using octree encoding. *Computer Graphics and Image Processing* **19**(2), 129–147 (1982).

155. Meixner, P., Leberl, F. & Brédif, M. *Planar Roof Surface Segmentation Using 3D Vision.* In *19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (2011), 9–15. ISBN: 9781450310314.

156. Mémoli, F. & Sapiro, G. *Comparing point clouds.* In *Symposium on Geometry Processing* (2004), 32–40.

157. Meng, X, Currit, N & Yang, X. *Object-oriented residential building land-use mapping using lidar and aerial photographs.* In *American society of photogrammetry and remote sensing 2010 annual conference* (2010), 26–30.

158. Meng, X., Currit, N. & Zhao, K. Ground Filtering Algorithms for Airborne LiDAR Data: A Review of Critical Issues. *Remote Sensing* **2**(3), 833–860. ISSN: 2072-4292 (2010).

159. Microsoft, *Photosynth.* https://photosynth.net/. (Last accessed 20.11.2017), 2015.

160. Mitra, N. J. & Nguyen, A. *Estimating surface normals in noisy point cloud data.* In *19th Annual Symposium on Computational Geometry* (2003), 322–328. ISBN: 1581136633.

161. Monserrat, O & Crosetto, M. Deformation measurement using terrestrial laser scanning data and least squares 3D surface matching. *ISPRS Journal of Photogrammetry and Remote Sensing* **63**(1), 142–154. ISSN: 09242716 (2008).

162. Morsy, S., Shaker, A. & El-rabbany, A. Multispectral LiDAR Data for Land Cover Classification of Urban Areas. *Sensors* **17**(5), 958. ISSN: 1424-8220 (2017).

163. NASA, *Remote Sensors.* (Last accessed 13.11.2017), 2017.

164. Nebiker, S., Bleisch, S. & Christen, M. Rich point clouds in virtual globes – A new paradigm in city modeling? *Computers, Environment and Urban Systems* **34**(6), 508–517. ISSN: 01989715 (2010).

165. Ni, H., Lin, X. & Zhang, J. Classification of ALS Point Cloud with Improved Point Cloud Segmentation and Random Forests. *Remote Sensing* **9**(3), 288. ISSN: 20724292 (2017).

166. NVIDIA Corporation, *CUDA GPUs.* (Last accessed 19.01.2016), 2016.

167. NVIDIA Corporation, *NVIDIA CUDA C Programming Guide v4.0* (2011).

168. Olson, M, Dyer, R., Zhang, H. & Sheffer, A. Point set silhouettes via local reconstruction. *Computers & Graphics* **35**(3), 500–509 (2011).

169. Ono, A, Kajiwara, K & Honda, Y. Development of new vegetation indexes, shadow index (SI) and water stress trend (WST). *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Science* **XXXVIII**(8), 710–714 (2010).

170. Pajarola, R. Stream-Processing Points. *IEEE Visualization,* 239–246 (2005).

171. Pajarola, R., Sainz, M. & Lario, R. *XSplat: External Memory Multiresolution Point Visualization.* In *IASTED Visualization, Imaging and Image Processing* (2005), 628–633.

172. Pan, J. & Manocha, D. *Fast GPU-based locality sensitive hashing for k-nearest neighbor computation.* In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems - GIS '11* (ACM Press, 2011), 211–200. ISBN: 9781450310314.

173. Paravati, G. & Sanna, A. An open and scalable architecture for delivering 3D shared visualization services to heterogeneous devices. *Concurrency and Computation: Practice and Experience* **23**(11), 1179–1195 (2011).

174. Park, S., Guo, X., Shin, H., Qin, H., KAIST, D. & Korea, S. *Shape and appearance repair for incomplete point surfaces.* In *Tenth IEEE International Conference on Computer Vision* **2** (2005), 1260–1267.

175. Pauly, M., Kobbelt, L. P. & Gross, M. Point-based multiscale surface representation. *ACM Transactions on Graphics* **25,** 177–193. ISSN: 07300301 (2006).

176. Pauly, M., Keiser, R., Kobbelt, L. & Gross, M. H. Shape modeling with point-sampled geometry. *ACM Transactions on Graphics* **22,** 641–650 (2003).

177. Pfister, H, Zwicker, M, Baar, J. V. & Gross, M. *Surfels: Surface elements as rendering primitives.* In *ACM SIGGRAPH* (2000), 335–342.

178. Poullis, C. & You, S. Photorealistic Large-Scale Urban City Model Reconstruction. *IEEE Transactions on Visualization and Computer Graphics* **15**(4), 654 –669 (2009).

179. Prandi, F, Soave, M, Devigili, F, Andreolli, M. & Amicis, R. D. Services Oriented Smart City Platform Based On 3d City Model Visualization. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* **II-4,** 59–64 (2014).

180. Preiner, R., Jeschke, S. & Wimmer, M. *Auto Splats: Dynamic Point Cloud Visualization on the GPU.* In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization* (2012), 139–148.

181. Puente, I., González-Jorge, H., Martínez-Sánchez, J. & Arias, P. Review of mobile mapping and surveying technologies. *Measurement* **46**(7), 2127–2145 (2013).

182. Qi, C. R., Su, H., Mo, K. & Guibas, L. J. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *Conference on Computer Vision and Pattern Recognition (CVPR).* eprint: 1612.00593v2 (2017).

183. Qiu, D., May, S. & Andreas, N. *GPU-accelerated Nearest Neighbor Search for 3D Registration.* In *International Conference on Computer Vision Systems (ICVS '09)* (2009), 194–203.

184. Rabbani, T., van den Heuvel, F. & Vosselman, G. *Segmentation of Point Clouds Using Smoothness Constraint.* In *ISPRS Image Engineering and Vision Metrology* (2006), 248 –253.

185. Raber, G. T., Jensen, J. R., Schill, S. R. & Schuckman, K. Creation of Digital Terrain Models Using an Adaptive Lidar Vegetation Point Removal Process. *Photogrammetric Engineering & Remote Sensing* **68**(12), 1307–1315 (2002).

186. Reitberger, J., Heurich, M., Krzystek, P. & Stilla, U. Single Tree Detection in Forest Areas With High-Density Lidar Data. *Remote Sensing and Spatial Information Sciences* **36**(3/W49B), 139–144 (2007).

187. Remondino, F. & Menna, F. Image-Based Surface Measurement for Close-range Heritage Documentation. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* **37**(B5), 199–206 (2007).

188. Remondino, F., Spera, M. G., Nocerino, E., Menna, F. & Nex, F. State of the art in high density image matching. *The Photogrammetric Record* **29**(146), 144–166. ISSN: 0031868X (2014).

189. Rodríguez, M. & Gobbetti, E. *Coarse-grained multiresolution structures for mobile exploration of gigantic surface models.* In *SIGGRAPH Asia Symposium on Mobile Graphics and Interactive Applications* (2013), 4:1–4:6.

190. Rothermel, M. & Haala, N. Potential of Dense Matching for the Generation of High Quality Digital Elevation Models. *International Society for Photogrammetry and Remote Sensing* **XXXVII**(4-W19) (2011).

191. Rusinkiewicz, S & Levoy, M. *QSplat: A multiresolution point rendering system for large meshes.* In *ACM SIGGRAPH* (2000), 343–352.

192. Rusinkiewicz, S. & Levoy, M. *Streaming QSplat: a viewer for networked visualization of large, dense models.* In *Proceedings of the 2001 symposium on Interactive 3D graphics - SI3D '01* (2001), 63–68. ISBN: 1581132921.

193. Rusu, R. B. & Cousins, S. *3D is here: Point Cloud Library (PCL).* In *IEEE International Conference on Robotics and Automation (ICRA)* (2011). ISBN: 978-1-61284-386-5.

194. Rutzinger, M., Höfle, B., Hollaus, M. & Pfeifer, N. Object-Based Point Cloud Analysis of Full-Waveform Airborne Laser Scanning Data for Urban Vegetation Classification. *Sensors* **8**(8), 4505–4528. ISSN: 1424-8220 (2008).

195. Sainz, M., Pajarola, R. & Lario, R. Points reloaded: Point-based rendering revisited. *Symposium on Point-Based Graphics,* 121–128 (2004).

196. Saito, T. & Takahashi, T. Comprehensible Rendering of 3-D Shapes. *SIGGRAPH Computer Graphics* **24**(4), 197–206. ISSN: 00978930 (1990).

197. Samadzadegan, F., Bigdeli, B. & Ramzi, P. *A Multiple Classifier System for Classification of LiDAR Remote Sensing Data Using Multi-class SVM.* In *Multiple Classifier Systems* (2010), 254–263.

198. Samadzadegan, F., Bigdeli, B. & Ramzi, P. Classification of Lidar Data Based on Multi-class SVM (2010).

199. Sanders, J. & Kandrot, E. *CUDA by Example: An Introduction to General-Purpose GPU Programming.* ISBN: 978-0131387683 (Addison-Wesley Professional, 2010).

200. Schachtschneider, J., Schlichting, A. & Brenner, C. Assessing Temporal Behavior in Lidar Point Clouds of Urban Environments. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* **42,** 543–550. ISSN: 2194-9034 (2017).

201. Scheiblauer, C, Zimmermann, N & Wimmer, M. Interactive Domitilla Catacomb Exploration. *Symposium on Virtual Reality, Archaeology and Cultural Heritage VAST* (2009).

202. Schilling, A. & Kolbe, T. *Draft for Candidate OpenGIS® Web 3D Service Interface Standard, Version 0.4.0 Open Geospatial Consortium Inc.* 2010.

203. Schnabel, R., Wahl, R. & Klein, R. Ransac Based Out-of-Core Point-Cloud Shape Detection for City-Modeling. *zugspitze.cs.uni-bonn.de,* 1–8.

204. Schneider, D. *Terrestrial laser scanning for area based deformation analysis of towers and water damns.* In *Proceedings of the 3rd IAG Symposium of Geodesy for Geotechnical and Structural Engineering and 12th FIG Symposium on Deformation Measurements* (2006), 22–24.

205. Schut, P. *OpenGIS Web Processing Service, Version 1.0.0 Open Geospatial Consortium Inc.* 2007.

206. Schütz, M. *Rendering Large Point Clouds in Web Browsers Multi-Resolution Octree.* In *CESCG 2015: The 19th Central European Seminar on Computer Graphics* (2015).

207. Secord, J. & Zakhor, A. Tree Detection in Urban Regions Using Aerial LiDAR and Image Data. *Geoscience and Remote Sensing Letters* **4**(2), 196–200. ISSN: 1545-598X (2007).

208. Shaffer, C. A. & Samet, H. Optimal Quadtree Construction Algorithms. *Computer Vision, Graphics and Image Processing* **37**(3), 402–419. ISSN: 0734189X (1987).

209. Shao, Y. & Chen, L. Automated Searching of Ground Points from Airborne Lidar Data Using a Climbing and Sliding Method. *Photogrammetric Engineering and Remote Sensing* **74**(5), 625–635 (2008).

210. Shapovalov, R. & Velizhev, A. Cutting-Plane Training of Non-associative Markov Network for 3D Point Cloud Segmentation. *International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission,* 1–8 (2011).

211. Shapovalov, R., Velizhev, A. & Barinova, O. Non-Associative Markov Networks for 3D Point Cloud Classification. *Photogrammetric Computer Vision and Image Analysis* **38**(3A), 103–108 (2010).

212. Shreiner, D., Sellers, G., Kessenich, J. M. & Licea-Kane, B. M. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.3* 8th (Addison-Wesley, 2013).

213. Sibbing, D., Sattler, T., Leibe, B. & Kobbelt, L. SIFT-Realistic Rendering. *International Conference on 3D Vision,* 56–63 (2013).

214. Sithole, G. & Vosselman, G. Experimental Comparison of Filter Algorithms for Bare-Earth Extraction from Airborne Laser Scanning Point Clouds. *ISPRS Journal of Photogrammetry and Remote Sensing* **59**(1-2), 85–101. ISSN: 09242716 (2004).

215. Sithole, G. & Vosselman, G. *Filtering of Airborne Laser Scanner Data Based on Segmented Point Clouds.* In *Laser Scanning 2005* (2005), 66–71.

216. Snavely, N., Seitz, S. & Szeliski, R. Photo tourism: Exploring photo collections in 3D. *ACM Transactions on Graphics* **25**(3), 835–846 (2006).

217. Snavely, N., Garg, R., Seitz, S. M. & Szeliski, R. Finding Paths Through the World's Photos. *ACM Transactions on Graphics* **27**(3), 11–21. ISSN: 07300301 (2008).

218. Sotoodeh, S. Outlier detection in laser scanner point clouds. *International Archives of Photogrammetry and Remote Sensing* **XXXVI**(5), 297–302 (2006).

219. Stein, C., Limper, M. & Kuijper, A. *Spatial data structures for accelerated 3D visibility computation to enable large model visualization on the web.* In *Proceedings of the 19th International ACM Conference on 3D Web Technologies* (2014), 53–61. ISBN: 9781450330152.

220. Thoeni, K., Giacomini, a., Murtagh, R. & Kniest, E. A comparison of multi-view 3D reconstruction of a rock wall using several cameras and a laser scanner. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* **XL-5**(June), 573–580. ISSN: 2194-9034 (2014).

221. Tittmann, P., Shafii, S., Hartsough, B. & Hamann, B. *Tree Detection and Delineation from LiDAR point clouds using RANSAC.* In *SilviLaser* (2011), 1–23.

222. Tran, P., Shaw, R., Chantry, G. & Norton, J. GIS and local knowledge in disaster management: a case study of flood risk mapping in Viet Nam. *Disasters* (2009).

223. Trapp, M., Glander, T., Buchholz, H. & Döllner, J. *3D Generalization Lenses for Interactive Focus + Context Visualization of Virtual City Models.* In *12th International Conference on Information Visualisation* (2008), 356–361. ISBN: 978-0-7695-3268-4.

224. Trinder, J. & Salah, M. Airborne LiDAR as a tool for disaster monitoring and management. *GeoInformation for Disaster Management* (2011).

225. Tucker, C. Red and Photographic Infrared Linear Combinations for Monitoring Vegetation. *Remote Sensing of Environment* **8**(2), 127–150 (1979).

226. Tuttas, S., Braun, A., Borrmann, A. & Stilla, U. Acquisition and Consecutive Registration of Photogrammetric Point Clouds for Construction Progress Monitoring Using a 4D BIM. *Journal of Photogrammetry, Remote Sensing and Geoinformation Science* **85**(1), 3–15 (2017).

227. Vaaraniemi, M., Freidank, M. & Westermann, R. *Enhancing the visibility of labels in 3D navigation maps.* In *Lecture Notes in Geoinformation and Cartography* (2012), 23–40.

228. Van Oosterom, P., Martinez-Rubi, O., Ivanova, M., Horhammer, M., Geringer, D., Ravada, S., Tijssen, T., Kodde, M. & Gonçalves, R. Massive point cloud data management: Design, implementation and execution of a point cloud benchmark. *Computers & Graphics* **49,** 92–125 (2015).

229. Van Gosliga, R., Lindenbergh, R. & Pfeifer, N. Deformation analysis of a bored tunnel by means of terrestrial laser scanning. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* **36**(5), 167–172 (2006).

230. Varcholik, P. *Real-Time 3D Rendering with DirectX 11 and HLSL: A Practical Guide to Graphics Programming (Game Design and Development).* ISBN: 978-0321962720 (Addison Wesley, 2014).

231. Virtanen, J.-p., Kukko, A., Kaartinen, H., Turppa, T., Hyyppä, H. & Hyyppä, J. Nationwide Point Cloud - The Future Topographic Core Data. *ISPRS International Journal of Geo-Information* **6**(8), 243 (2017).

232. Voland, P. & Asche, H. *Geospatial Visualization of Automotive Sensor Data: A Conceptual and Implementational Framework for Environment and Traffic-Related Applications.* In *Computational Science and Its Applications - ICCSA 2017* (2017), 626–637. ISBN: 978-3-319-62406-8.

233. Vosselman, G. Point cloud segmentation for urban scene classification. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* **XL-7/W2**(November), 257–262. ISSN: 2194-9034 (2013).

234. Vosselman, G., Gorte, B., Sithole, G. & Rabbani, T. Recognising Structure in Laser Scanner Point Clouds. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* **46**(8), 33–38 (2004).

235. Vretanos, P. P. A. *OpenGIS Web Feature Service 2.0 Interface Standard, Version 2.0.0 Open Geospatial Consortium Inc.* 2010.

236. Wahl, R., Guthe, M. & Klein, R. Identifying planes in point-clouds for efficient hybrid rendering. *The 13th Pacific Conference on Computer Graphics and Applications,* 1–8 (2005).

237. Wald, I., Dietrich, A. & Slusallek, P. An Interactive Out-of-Core Rendering Framework for Visualizing Massively Complex Models. *Eurographics Symposium on Rendering* (2004).

238. Wald, I., Mark, W. & Günther, J. State of the Art in Ray Tracing Animated Scenes. *Computer Graphics Forum* **28**(6), 1691–1722 (2009).

239. Wand, M, Berner, A, Bokeloh, M, Fleck, A, Hoffmann, M, Jenke, P, Maier, B, Staneker, D & Schilling, A. Interactive Editing of Large Point Clouds. *Eurographics Symposium on Point-Based Graphics,* 37–46 (2007).

240. Wang, J & Oliveira, M. Filling holes on locally smooth surfaces reconstructed from point clouds. *Image and Vision Computing* **25**(1), 103–113. ISSN: 02628856 (2007).

241. Wang, L., Xu, Y. & Li, Y. Aerial Lidar Point Cloud Voxelization with its 3D Ground Filtering Application. *Photogrammetric Engineering & Remote Sensing* **83**(2), 95–107 (2017).

242. Wang, S., Wilkins-Diehr, N. R. & Nyerges, T. L. CyberGIS - Toward synergistic advancement of cyberinfrastructure and GIScience: A workshop summary. *Journal of Spatial Information Science* **4**(4), 125–148. ISSN: 1948-660X (2012).

243. Wang, Y., Xu, H., Cheng, L., Li, M., Wang, Y., Xia, N., Chen, Y. & Tang, Y. Three-Dimensional Reconstruction of Building Roofs from Airborne LiDAR Data Based on a Layer Connection and Smoothness Strategy. *Remote Sensing* **8**(5), 415. ISSN: 20724292 (2016).

244. Wehr, A. & Lohr, U. Airborne Laser Scanning - an Introduction and Overview. *ISPRS Journal of Photogrammetry and Remote Sensing* **54**(2-3), 68–82 (1999).

245. Weinmann, M., Weinmann, M., Mallet, C. & Brédif, M. A Classification-Segmentation Framework for the Detection of Individual Trees in Dense MMS Point Cloud Data Acquired in Urban Areas. *Remote Sensing* **9**(3), 277. ISSN: 20724292 (2017).

246. Wenzel, K., Abdel-Wahab, M., Cefalu, A. & Fritsch, D. High-resolution surface reconstruction from imagery for close range cultural Heritage applications. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Science* **XXXIX-B5,** 133–138 (2012).

247. Wimmer, M. & Scheiblauer, C. *Instant points: Fast rendering of unprocessed point clouds.* In *Eurographics Symposium on Point-Based Graphics* (2006), 129–137.

248. Wolff, D. *OpenGL 4 Shading Language Cookbook* 2nd Revise. ISBN: 978-1782167020 (Packt Publishing, 2013).

249. Wolff, M. & Asche, H. *Geospatial Modelling of Urban Security: A Novel Approach with Virtual 3D City Models.* In *Computational Science and Its Applications - ICCSA 2008* (2008), 42–51.

250. Wu, J. & Kobbelt, L. Optimized Sub-Sampling of Point Sets for Surface Splatting. *Computer Graphics Forum* **23**(3), 643–652. ISSN: 0167-7055 (2004).

251. Xu, H., Nguyen, M. X., Yuan, X. & Chen, B. Interactive Silhouette Rendering for Point-Based Models. *Eurographics Symposium on Point-Based Graphics,* 13–18 (2004).

252. Yang, B., Huang, R., Li, J., Tian, M., Dai, W. & Zhong, R. Automated Reconstruction of Building LoDs from Airborne LiDAR Point Clouds Using an Improved Morphological Scale Space. *Remote Sensing* **9**(1) (2017).

253. Yang, R., Guinnip, D. & Wang, L. View-dependent textured splatting. *The Visual Computer* **22**(7), 456–467. ISSN: 0178-2789 (2006).

254. Yang, Y. B., Zhang, N. N. & Li, X. L. Adaptive slope filtering for airborne Light Detection and Ranging data in urban areas based on region growing rule. *Survey Review* **49**(353), 139–146 (2017).

255. Yang, Z., Jiang, W., Xu, B., Zhu, Q., Jiang, S. & Huang, W. A convolutional neural network-based 3D semantic labeling method for ALS point clouds. *Remote Sensing* **9**(9), 936. ISSN: 20724292 (2017).

256. Yao, W. & Fan, H. *Automated detection of 3D individual trees along urban road corridors by mobile laser scanning systems.* In *International Symposium on Mobile Mapping Technology (MMT) 2013* (2013).

257. Yu, J. & Turk, G. Reconstructing surfaces of particle-based fluids using anisotropic kernels. *ACM Transactions on Graphics* **32**(1), 5:1–5:12. ISSN: 07300301 (2013).

258. Yunfei, B., Guoping, L., Chunxiang, C., Xiaowen, L., Hao, Z., Qisheng, H., Linyan, B. & Chaoyi, C. *Classification of LIDAR point cloud and generation of DTM from LIDAR height and intensity data in forested area.* In *International Society for Photogrammetry and Remote Sensing Congress* (2008), 313–318.

259. Zhang, J., Duan, M., Yan, Q. & Lin, X. Automatic Vehicle Extraction from Airborne LiDAR Data Using an Object-Based Point Cloud Analysis Method. *Remote Sensing* **6**(9), 8405–8423. ISSN: 2072-4292 (2014).

260. Zhang, J., Duan, M., Yan, Q. & Lin, X. Automatic Vehicle Extraction from Airborne LiDAR Data Using an Object-Based Point Cloud Analysis Method. *Remote Sensing* **6**(9), 8405–8423. ISSN: 2072-4292 (2014).

261. Zhang, K., Yan, J. & Chen, S. *Automatic construction of building footprints from airborne LIDAR data.* In *IEEE Transactions on Geoscience and Remote Sensing* (2006), 2523 –2533.

262. Zhang, K. & Hu, B. Individual Urban Tree Species Classification Using Very High Spatial Resolution Airborne Multi-Spectral Imagery Using Longitudinal Profiles. *Remote Sensing* **4**(6), 1741–1757. ISSN: 2072-4292 (2012).

263. Zhang, Y.-K., Teboul, O., Zhang, X.-P. & Deng, Q.-Q. Image Based Real-Time and Realistic Forest Rendering and Forest Growth Simulation. *2006 Second International Symposium on Plant Growth Modeling and Applications,* 323–327 (2006).

264. Zhang, Chengyuan and Zhang, Ying and Zhang, Wenjie and Lin, X. *Inverted Linear Quadtree: Efficient Top K Spatial Keyword Search.* In *International Conference on Data Engineering* **28** (2013), 901–912.

265. Zhao, W., Nister, D. & Hsu, S. Alignment of continuous video onto 3D point clouds. *IEEE transactions on pattern analysis and machine intelligence* **27,** 1305–1318. ISSN: 0162-8828 (2005).

266. Zhou, Q.-Y. & Neumann, U. *2.5D Building Modeling by Discovering Global Regularities.* In *Computer Vision and Pattern Recognition* (2012), 326–333.

267. Zhou, Q.-Y. & Neumann, U. *Fast and Extensible Building Modeling from Airborne Lidar Data.* In *16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (2008), 1–8.

268. Zhu, Q., Gong, J. & Zhang, Y. An efficient 3D R-tree spatial index method for virtual geographic environments. *ISPRS Journal of Photogrammetry and Remote Sensing* **62**(3), 217–224 (2007).

269. Zwicker, M & Gotsman, C. Meshing point clouds using spherical parameterization. *Proc. Eurographics Symp. Point-Based Graphics* (2004).

270. Zwicker, M., Pauly, M., Knoll, O. & Gross, M. Pointshop 3D: an interactive system for point-based surface editing. *ACM Transactions on Graphics,* 322–329. ISSN: 07300301 (2002).

271. Zwicker, M., Pfister, H., van Baar, J. & Gross, M. H. *Surface splatting.* In *ACM SIGGRAPH* (2001), 371–378.

Eidesstattliche Erklärung
Declaration of Academic Honesty

I hereby declare in lieu of an oath that this thesis has been written by myself without any external unauthorized help, that it has been neither presented to any institution for evaluation nor previously published in its entirety or in parts.

Potsdam, March 23, 2018

Rico Richter